# Hyperbolic Graph Diffusion Model

**Lingfeng Wen[1], Xuan Tang[2], Mingjie Ouyang[1], Xiangxiang Shen[1], Jian Yang[3], Daxin Zhu[4], Mingsong Chen[1], Xian Wei[1]***

[1]MoE Engineering Research Center of Hardware/Software Co-design Technology and Application, East China Normal University

[2]School of Communication and Electronic Engineering, East China Normal University

[3]Information Engineering University; [4]Quanzhou Normal University

xwei@sei.ecnu.edu.cn

## Abstract

Diffusion generative models (DMs) have achieved promising results in image and graph generation. However, real-world graphs, such as social networks, molecular graphs, and traffic graphs, generally share non-Euclidean topologies and hidden hierarchies. For example, the degree distributions of graphs are mostly power-law distributions. The current latent diffusion model embeds the hierarchical data in a Euclidean space, which leads to distortions and interferes with modeling the distribution. Instead, hyperbolic space has been found to be more suitable for capturing complex hierarchical structures due to its exponential growth property. In order to simultaneously utilize the data generation capabilities of diffusion models and the ability of hyperbolic embeddings to extract latent hierarchical distributions, we propose a novel graph generation method called, Hyperbolic Graph Diffusion Model (HGDM), which consists of an auto-encoder to encode nodes into successive hyperbolic embeddings, and a DM that operates in the hyperbolic latent space. HGDM captures the crucial graph structure distributions by constructing a hyperbolic potential node space that incorporates edge information. Extensive experiments show that HGDM achieves better performance in generic graph and molecule generation benchmarks, with a $48\%$ improvement in the quality of graph generation with highly hierarchical structures.

## Introduction

Graph representation learning and graph generation are important machine learning tasks that have wide applications in fields such as social networks, drug discovery, and recommendation systems. While real-world graphs, such as social networks, molecular graphs, user-item interactions, and traffic graphs, generally have non-Euclidean topologies and hidden hierarchies (**???**), e.g., the degree distribution of the nodes are mostly power-law distributions (**?**). It is common practice in graph representation learning to embed node representations into Euclidean space (**?**). However, the polynomially growing capacity of Euclidean space struggles to maintain the intrinsic distances of the huge number of nodes located in long-tailed regions and differentiate between them. Therefore, embedding these graphs into Euclidean space may produce distortions (**?**). Especially for the graphs of drug molecules, changes in a few atoms or chemical bonds can have a significant effect on the properties of the molecules (**?**). On the contrary, it has been found that the capacity of hyperbolic spaces grows exponentially (**?**) (see Figure 1). This property makes hyperbolic spaces well-suited for embedding tree-like or scale-free graphs (**?**). It has been observed that the complex hierarchical structures in hyperbolic embedding space become more prominent and easier to be captured (**??**). Some existing hyperbolic graph neural networks (**??**) show excellent performance in graph representation learning tasks such as node classification and edge prediction. **??** also demonstrate superior performance over Euclidean methods in predicting long-tail items of user-item interaction for collaborative filtering in recommendation systems.

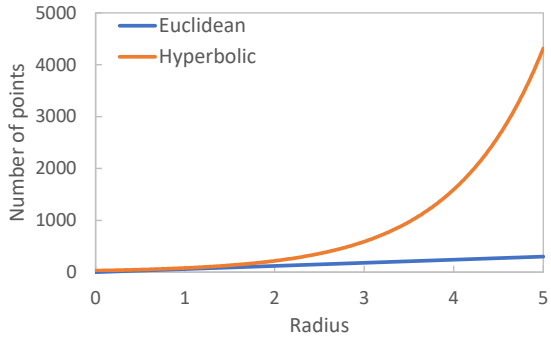


Figure 1: The number of points in 2D hyperbolic and Euclidean space that can be placed at radius $r$ from the center point while maintaining a distance of at least $s = 0.1$ from each other.

Furthermore, regarding generative models, diffusion models have achieved amazing results in synthesizing images (**?**). The diffusion model perturbs the data distribution gradually during the forward diffusion process and then learns to recover the data distribution from noise through the reverse diffusion process. The Latent Diffusion Model (**?**) embeds images into a low-dimensional latent space to reduce the computational complexity. **?** unified *Score matching with Langevin dynamics* (SMLD) (**?**) and *Denoising diffusion probabilistic modeling* (DDPM) (**?**) into a system of stochastic differential equations (SDEs), referred as score-

*Corresponding author

based generative models, which model the gradient of the log probability density of the data, the so-called "score". GDSS (**?**) extended score-based generative models to graph generation tasks and proved that score-based models also outperform existing graph generative methods. However, this method performs in Euclidean space, which is inconsistent with the natural properties of graphs, does not adequately model the distribution of graph data, and still has room for improvement. In recent years, efforts have been made to extend the diffusion model to different manifolds to take full advantage of the underlying structure of the data. Torsional diffusion (**?**) performs better for generating molecular conformers by applying diffusion processes on the hypertorus to the torsion angles. Riemannian diffusion models such as RDM (**?**) and RSGM (**?**) extended the diffusion model to arbitrary compact and non-compact Riemannian manifolds. However, they only test a few simple predefined distributions on manifolds and do not adequately evaluate the Riemannian diffusion model on larger real-world datasets.

In order to simultaneously utilize the excellent data generation capability of the diffusion model and the ability of the hyperbolic space to embed hierarchical distributions, we propose a two-stage graph diffusion model based on the hyperbolic latent space. Firstly, we train a hyperbolic graph variational auto-encoder (HVAE) to learn the hyperbolic representation of nodes. Then, we train two score models simultaneously to learn the reverse diffusion process in hyperbolic space (for nodes) and Euclidean space (for adjacency matrices), which also model the dependency between nodes and adjacency matrices. In addition, since diffusion models are more time-consuming for sampling compared to VAEs or GANs, in order to avoid another increase in time-consumption by using hyperbolic methods, we also design a *Hyperbolic Graph Attention (HGAT)* layer as a basic module for HVAE and score-based models to maintain the performance without significantly increasing in time spent sampling. Our approach effectively utilizes the properties of hyperbolic space and the generative quality advantages of the diffusion model.

Our main contributions are summarized as follows:

- We propose a novel approach for graph generation, the Hyperbolic Graph Diffusion Model (HGDM). To the best of our knowledge, HGDM is the first hyperbolic diffusion model in graph generation.
- We further design a simple and novel hyperbolic graph attention layer that utilizes the better representational capabilities of hyperbolic space without significantly increasing the computational time.
- We consider both generic graph and molecular graph generation tasks. Experiments show that HGDM significantly outperforms all baselines in most metrics.

## Related Work

### Graph Generation

**?** proposed a method called GraphVAE for generating small graphs using a variational auto-encoder (VAE). MolGAN (**?**) combines generative adversarial networks (GANs) and reinforcement learning objectives to encourage the generation of molecules with specific desired chemical properties. (**??**) generate molecules in a flow-based fashion. Graphdf (**?**) is an autoregressive flow-based model that utilizes discrete latent variables. **?** proposed an energy-based model to generate molecular graphs.

### Hyperbolic Graph Neural Network

Hyperbolic Graph Neural Networks (HGNNs) (**????**) utilize the characteristics of hyperbolic space to better capture the hierarchical structure in graph data. The core idea is to embed node representations into hyperbolic space and use hyperbolic distance to measure their relationships, enabling better handling of graph data with a hierarchical structure. In the research of hyperbolic graph neural networks, **?** introduced the Hyperbolic Graph Convolutional Network (HGCN), which achieves new state-of-the-art results in learning embeddings for real-world graphs with hierarchical and scale-free features. Through experiments with hyperbolic auto-encoders, **?** find that utilizing hyperbolic geometry improves the performance of unsupervised tasks on graphs, such as node clustering and link prediction. **?** extended VAE into the hidden space of a Poincar ball manifold as P-VAE, which shows better generalization to unseen data and the ability to qualitatively and quantitatively recover hierarchical structures.

### Diffusion Models

After the denoising diffusion probabilistic model (**?**) was proposed, a large number of related works were carried out to improve this type of model such as DDIM (**?**), score-based diffusion (**?**). Some works extended diffusion models to graph generation tasks. **?** proposed a novel score-based graph generation model, which can generate both node features and adjacency matrices via the system of SDEs by introducing the diffusion process of graphs. **?** proposed a novel generative model named GEODIFF for molecular conformation prediction. In addition, some works have extended the diffusion model to non-Euclidean manifolds to adequately model the data distribution. **?** proposed torsional diffusion, a novel diffusion framework that operates on the space of torsion angles via a diffusion process on the hypertorus and an extrinsic-to-intrinsic score model. RDM (**?**) and RSGM (**?**) extended the diffusion model to arbitrary compact and non-compact Riemannian manifolds, but have not been fully evaluated on more real-world datasets.

## Preliminaries

### Riemannian Manifold

A Riemannian manifold $(\mathcal{M}, g^{\mathcal{M}})$ is a smooth manifold $\mathcal{M}$ equipped with a Riemannian metric $g^{\mathcal{M}}$ on the tangent space $\mathcal{T}_x\mathcal{M}$ at every point $x \in \mathcal{M}$. $\mathcal{T}_x\mathcal{M}$ is the vector space formed by all tangent vectors at $x$, which is locally a first-order approximation of the hyperbolic manifold at $x$. The Riemannian manifold metric $g^{\mathcal{M}}$ assigns a positive definite inner product $\langle \cdot, \cdot \rangle : \mathcal{T}_x\mathcal{M} \times \mathcal{T}_x\mathcal{M} \to \mathbb{R}$ on

the tangent space, which makes it possible to define several geometric properties. For a curve $\gamma : [a, b] \to \mathcal{M}$, the length of $\gamma$ is defined as $L(\gamma) = \int_a^b \|\gamma'(t)\|_g \, \mathrm{d}t$,where $\|\cdot\|_g$ denotes the norm induced by the Riemannian metric $g^{\mathcal{M}}$, i.e., $\|v\|_g = \sqrt{g(v, v)}$ for any $v \in T_x\mathcal{M}$. A geodesic is the analogue of a straight line in Euclidean geometry, being the shortest distance between two points $x, y$ on manifold $\mathcal{M}$: $d_{\mathcal{M}}(x, y) = \inf L(\gamma)$ where $\gamma$ is a curve with $\gamma(a) = x, \gamma(b) = y$. Exponential and logarithmic mappings are often described as additions and subtractions on Riemannian manifolds. The exponential map at point $x \in \mathcal{M}$, denoted as $\exp_x : T_x\mathcal{M} \to \mathcal{M}$, takes a tangent vector $v \in T_x\mathcal{M}$ to the point $y \in \mathcal{M}$ obtained by "following" the geodesic starting at $x$ in the direction of $v$ for a unit amount of time. The logarithmic map is the inverse of the exponential map $\log_x = \exp_x^{-1} : \mathcal{M} \to T_x\mathcal{M}$.

**Poincar Ball Model.** A hyperbolic manifold is a Riemannian manifold with a constant negative curvature c $(c < 0)$. There exist multiple isomorphic hyperbolic models, including the Poincar ball model, Lorentz model and Klein model. The Poincar ball model is denoted as $(\mathcal{B}_c^n, g_x^{\mathcal{B}})$, where $\mathcal{B}_c^n = \{x \in \mathbb{R}^n : \|x\|^2 < -1/c\}$ is an open $n$-dimensional ball with radius $1/\sqrt{|c|}$. Its metric tensor is $g_x^{\mathcal{B}} = (\lambda_x^c)^2 g_x^E$, with conformal factor $\lambda_x^c = 2/(1 + c\,\|x\|^2)$ and Euclidean metric $g^E$, i.e., $\mathbf{I}_n$. Given 2 points $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}_c^n$, the induced geodesic distance on Poincar ball is

$$d_{\mathcal{B}}^c(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{\sqrt{|c|}} \cosh^{-1}\left(1 - \frac{2c\|\boldsymbol{x}-\boldsymbol{y}\|^2}{(1+c\|\boldsymbol{x}\|^2)\,(1+c\|\boldsymbol{y}\|^2)}\right) \quad (1)$$

**?** derived closed-form formulations for the exponential map as

$$\exp_{\boldsymbol{x}}^c(\boldsymbol{v}) = \boldsymbol{x} \oplus_c \left(\tanh\left(\sqrt{|c|}\frac{\lambda_{\boldsymbol{x}}^c\|\boldsymbol{v}\|}{2}\right)\frac{\boldsymbol{v}}{\sqrt{|c|}\|\boldsymbol{v}\|}\right) \quad (2)$$

where $v \in T_x\mathcal{B}$. The logarithm map is given by

$$\log_{\boldsymbol{x}}^c(\boldsymbol{y}) = \frac{2}{\sqrt{|c|}\lambda_{\boldsymbol{x}}^c} \tanh^{-1}\left(\sqrt{|c|}\|-\boldsymbol{x}\oplus_c \boldsymbol{y}\|\right)\frac{-\boldsymbol{x}\oplus_c \boldsymbol{y}}{\|-\boldsymbol{x}\oplus_c \boldsymbol{y}\|} \quad (3)$$

The parallel transport $PT_{x\to y} : T_x\mathcal{B} \to T_y\mathcal{B}$ defines the movement of a vector from one tangent space to another along a curve without changing itself, and is given by

$$PT_{\mathbf{x}\to\mathbf{y}}^c(\mathbf{v}) = \frac{\lambda_x^c}{\lambda_{\mathbf{y}}^c}\,\mathrm{gyr}[\mathbf{y}, -\mathbf{x}]\mathbf{v} \quad (4)$$

where $\oplus_c$ and $\mathrm{gyr}[\cdot,\cdot]v$ are Mbius addition (**?**) and gyration operator (**?**), respectively.

## Graph Diffusion Model

Diffusion models perturb the data by adding progressively larger noise and then learn to reverse it. Recently, SMLD (**?**) and DDPM (**?**) are unified into the SDE system (**?**). The Diffusion model perturbs the data in the forward SDE and estimates the score function (that is, the gradient of the log probability density with respect to data). Sampling is performed by using the estimated score function in the reverse SDE. GDSS (**?**) extended the diffusion model to graph generation tasks. Graph diffusion models perturb both node features and adjacency matrix, modeling their dependencies to learn how to convert from noise to graph.

A graph $\boldsymbol{G}$ with $N$ nodes is defined by its node features $\boldsymbol{X} \in \mathbb{R}^{N\times F}$ and the weighted adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$as $\boldsymbol{G} = (\boldsymbol{X}, \boldsymbol{A}) \in \mathbb{R}^{N\times F} \times \mathbb{R}^{N\times N} := \mathcal{G}$, where $F$ is the dimension of the node features. Formally, the diffusion process can be represented as the trajectory of random variables$\{\boldsymbol{G}_t = (\boldsymbol{X}_t, \boldsymbol{A}_t)\}_{t\in[0,T]}$ in a fixed time horizon $[0, T]$, where $\boldsymbol{G}_0$ is a graph from the data distribution $p_{data}$. The SDE of diffusion process is given by:

$$\mathrm{d}\boldsymbol{G}_t = \boldsymbol{f}_t(\boldsymbol{G}_t)\,\mathrm{d}t + \boldsymbol{g}_t(\boldsymbol{G}_t)\,\mathrm{d}\mathbf{w}, \quad \boldsymbol{G}_0 \sim p_{data} \quad (5)$$

where $\boldsymbol{f}_t(\cdot) : \mathcal{G} \to \mathcal{G}$ is the linear drift coefficient, $\boldsymbol{g}_t(\cdot) : \mathcal{G} \to \mathcal{G}\times\mathcal{G}$ is the diffusion coefficient, and $\mathbf{w}$ is the standard Wiener process.

The reverse-time SDE is separated into two parts in (**?**):

$$\begin{cases} \mathrm{d}\boldsymbol{X}_t = \left[\boldsymbol{f}_{1,t}(\boldsymbol{X}_t) - \boldsymbol{g}_{1,t}^2\nabla_{\boldsymbol{X}_t}\log p(\boldsymbol{X}_t, \boldsymbol{A}_t)\right]\mathrm{d}t + \boldsymbol{g}_{1,t}\,\mathrm{d}\overline{\mathbf{w}}_1 \\ \mathrm{d}\boldsymbol{A}_t = \left[\boldsymbol{f}_{2,t}(\boldsymbol{A}_t) - \boldsymbol{g}_{2,t}^2\nabla_{\boldsymbol{A}_t}\log p(\boldsymbol{X}_t, \boldsymbol{A}_t)\right]\mathrm{d}t + \boldsymbol{g}_{2,t}\,\mathrm{d}\overline{\mathbf{w}}_2 \end{cases} \quad (6)$$

where $\boldsymbol{f}_{1,t}$ and $\boldsymbol{f}_{2,t}$ are linear drift coefficients satisfying $\boldsymbol{f}_t(\boldsymbol{X}, \boldsymbol{A}) = \big(\boldsymbol{f}_{1,t}(\boldsymbol{X}), \boldsymbol{f}_{2,t}(\boldsymbol{A})\big)$, $\boldsymbol{g}_{1,t}$ and $\boldsymbol{g}_{2,t}$ are scalar diffusion coefficients, and $\overline{\mathbf{w}}_1$, $\overline{\mathbf{w}}_2$ are reverse-time standard Wiener processes. The $\nabla_{\boldsymbol{X}_t}\log p(\boldsymbol{X}_t, \boldsymbol{A}_t)$ and $\nabla_{\boldsymbol{A}_t}\log p(\boldsymbol{X}_t, \boldsymbol{A}_t)$ are named as the partial score functions.

# Hyperbolic Graph Diffusion Model

In this section, we introduce the proposed Hyperbolic Graph Diffusion Model (HGDM). Firstly, we introduce the hyperbolic wrapped normal distribution (HWN) and its sampling procedures. Secondly, we describe how to perturb data in the hyperbolic space and the training objective. Thirdly, we introduce a novel hyperbolic graph attention layer and the architecture of our model. Finally, we present the sampling methods for the inverse diffusion process in the hyperbolic space.

## Probability Distributions on Hyperbolic Space

As an alternative to the normal distribution in Euclidean space, the hyperbolic wrapped normal distribution (**??**) is easy to sample and has a well-defined density function. For the Poincar ball manifold, its density function is given by:

$$\begin{aligned} &\mathcal{N}_{\mathcal{B}_c^d}^{\mathrm{W}}(\boldsymbol{z} \mid \boldsymbol{\mu}, \Sigma) = \tfrac{d\nu^{\mathrm{W}}(\boldsymbol{z}|\boldsymbol{\mu},\Sigma)}{d\mathcal{M}(\boldsymbol{z})} \\ &= \mathcal{N}\left(\lambda_{\boldsymbol{\mu}}^c \log_{\boldsymbol{\mu}}(\boldsymbol{z}) \mid \mathbf{0}, \Sigma\right)\left(\tfrac{\sqrt{c}d_p^c(\boldsymbol{\mu},\boldsymbol{z})}{\sinh\left(\sqrt{c}d_p^c(\boldsymbol{\mu},\boldsymbol{z})\right)}\right)^{d-1} \end{aligned} \quad (7)$$

and its log density function is

$$\log p(\boldsymbol{z}) = \log p(\boldsymbol{v}) + (d-1)\log\left(\frac{\sqrt{c}d_p^c(\boldsymbol{\mu},\boldsymbol{z})}{\sinh\left(\sqrt{c}d_p^c(\boldsymbol{\mu},\boldsymbol{z})\right)}\right) \quad (8)$$

where $\boldsymbol{z} \in \mathcal{B}_c^d$, $p(\boldsymbol{v})$ is the normal distribution in the tangent space of origin $\mathcal{T}_o\mathcal{B}$.

This distribution can be constructed by defining Gaussian distribution on the tangent space at the origin of the hyperbolic space and projecting the distribution onto hyperbolic

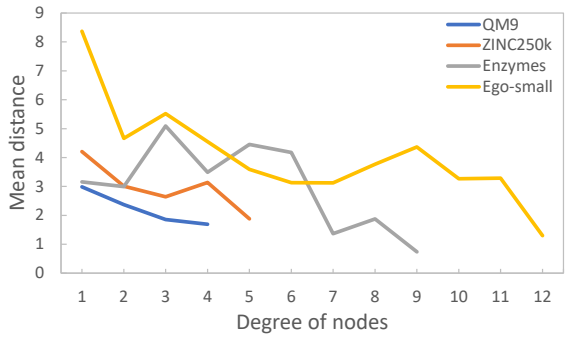


Figure 2: The average distance of the hyperbolic embedding of the nodes from the origin in different datasets with the degree of the nodes.

space after transporting the tangent space to a desired location in the space, using Eq. (4) and Eq. (2). We use the HWN as the prior distribution on the hyperbolic manifold and it is convenient to compute gradient using Eq. (8).

## Perturbing Data on Hyperbolic Space

We consider two types of noise perturbations, the Variance Exploding (VE) and the Variance Preserving (VP) given in (**?**). For Euclidean features, we have:

$$p(\mathbf{x}(t) \mid \mathbf{x}(0)) = \begin{cases} \mathcal{N}\left(\mathbf{x}(0), \left[\sigma^2(t) - \sigma^2(0)\right]\mathbf{I}\right), & \text{(VE SDE)} \\ \mathcal{N}\left(\mathbf{x}(0) e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, \mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s}\right), & \text{(VP SDE)} \end{cases} \tag{9}$$

where $\sigma(t) = \sigma_{\min}\left(\frac{\sigma_{\max}}{\sigma_{\min}}\right)^t$ for $t \in (0,1]$ and $\beta(t) = \bar{\beta}_{\min} + t\left(\bar{\beta}_{\max} - \bar{\beta}_{\min}\right)$ for $t \in [0,1]$.

In order to parallelly train on different time steps, we directly give the hyperbolic distribution $p(x_t|x_0)$:

$$p(\mathbf{x}(t) \mid \mathbf{x}(0)) = \begin{cases} \mathcal{N}^{\mathrm{W}}_{\mathcal{B}^d_c}\left(\mathbf{x}(0), \left[\sigma^2(t) - \sigma^2(0)\right]\mathbf{I}\right), & \text{(VE SDE)} \\ \mathcal{N}^{\mathrm{W}}_{\mathcal{B}^d_c}\left(e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s} \otimes_c \mathbf{x}(0), \mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s}\right), & \text{(VP SDE)} \end{cases} \tag{10}$$

where $\otimes_c$ is the Mbius scalar multiplication (**?**).

## Training Objective

The training process consists of two stages. Firstly, we train a hyperbolic graph variational auto-encoder (HVAE) to generate the hyperbolic representation of nodes. The training objective of the HVAE is as follows:

$$\mathcal{L}_{VAE} = \mathcal{L}_{rec} + \mathcal{L}_{kl} + \mathcal{L}_{edge} \tag{11}$$

which is a reconstruction loss combined with two regularization terms. The Kullback-Leibler (KL) regularization prevents latent embeddings from arbitrarily high variance and empirically we find it helpful for learning potential inverse diffusion processes. We also add a link prediction regularization objective, to encourage embedding to preserve the graph structure and implicitly constrain nodes with high degrees to be close to the origin (see Figure 2). We use the Fermi-Dirac decoder to compute probability scores for edges:

$$p\left(\boldsymbol{A}_{i,j} \neq 0 \mid \boldsymbol{x}_i^{\mathcal{B}}, \boldsymbol{x}_j^{\mathcal{B}}\right) = \left[e^{\left(d_{\mathcal{B}}^c\left(\boldsymbol{x}_i^{\mathcal{B}}, \boldsymbol{x}_j^{\mathcal{B}}\right)^2 - r\right)/t} + 1\right]^{-1} \tag{12}$$

Secondly, following **?**, we train two neural networks $\boldsymbol{s}_{\theta,t}$ and $\boldsymbol{s}_{\phi,t}$ simultaneously to estimate the partial score functions. But differently, given graph $\boldsymbol{G} = (\boldsymbol{X}, \boldsymbol{A}) \in \mathbb{R}^{N\times F} \times \mathbb{R}^{N\times N}$ from data, we first use a hyperbolic encoder to generate the hyperbolic latent feature $\boldsymbol{X}_0^{\mathcal{B}} \in \mathcal{B}_c^F$ of nodes, which combine with original adjacency matrix $\boldsymbol{A}$ forms $\bar{\boldsymbol{G}}_0 = (\boldsymbol{X}_0^{\mathcal{B}}, \boldsymbol{A}_0)$. Then, we sample $\bar{\boldsymbol{G}}_t = (\boldsymbol{X}_t^{\mathcal{B}}, \boldsymbol{A}_t)$ using Eq. (9) and Eq. (10).

We use $\boldsymbol{s}_{\theta,t}$ and $\boldsymbol{s}_{\phi,t}$ to estimate the gradients of the joint log-density so that $\boldsymbol{s}_{\theta,t}(\bar{\boldsymbol{G}}_t) \approx \nabla_{\boldsymbol{X}_t^{\mathcal{B}}} \log p\left(\boldsymbol{X}_t^{\mathcal{B}} \mid \boldsymbol{X}_0^{\mathcal{B}}\right)$ and $\boldsymbol{s}_{\phi,t}(\bar{\boldsymbol{G}}_t) \approx \nabla_{\boldsymbol{A}_t} \log p\left(\boldsymbol{A}_t \mid \boldsymbol{A}_0\right)$. It is worth noticing that $\nabla_{\boldsymbol{X}_t^{\mathcal{B}}} \log p\left(\boldsymbol{X}_t^{\mathcal{B}} \mid \boldsymbol{X}_0^{\mathcal{B}}\right)$ is on $T_{\boldsymbol{X}_t^{\mathcal{B}}}\mathcal{B}$.

The training objectives are given by

$$\begin{matrix} \min_\theta \mathbb{E}_t\left\{\lambda_1(t)\mathbb{E}_{\bar{\boldsymbol{G}}_0}\mathbb{E}_{\bar{\boldsymbol{G}}_t|\bar{\boldsymbol{G}}_0}\left\|\boldsymbol{s}_{\theta,t}(\bar{\boldsymbol{G}}_t) - \nabla_{\boldsymbol{X}_t^{\mathcal{B}}} \log p\left(\boldsymbol{X}_t^{\mathcal{B}} \mid \boldsymbol{X}_0^{\mathcal{B}}\right)\right\|_2^2\right\} \\ \min_\phi \mathbb{E}_t\left\{\lambda_2(t)\mathbb{E}_{\bar{\boldsymbol{G}}_0}\mathbb{E}_{\bar{\boldsymbol{G}}_t|\bar{\boldsymbol{G}}_0}\left\|\boldsymbol{s}_{\phi,t}(\bar{\boldsymbol{G}}_t) - \nabla_{\boldsymbol{A}_t} \log p(\boldsymbol{A}_t \mid \boldsymbol{A}_0)\right\|_2^2\right\} \end{matrix} \tag{13}$$

where $\lambda_1(t)$ and $\lambda_2(t)$ are positive weighting functions and $t$ is uniformly sampled from $[0, T]$.

## Hyperbolic Graph Attention Layer

We propose the *Hyperbolic Graph Attention (HGAT)* layer as the basic building block of HVAE and the node score predicting model $\boldsymbol{s}_{\theta,t}$, which is a variant of GAT[1] and has smaller computational complexity than HGCN (**?**) (see Table 3). The input of our layer is a set of hyperbolic node feature $\mathbf{h}^{\mathcal{B}} = \{\mathbf{h}_1^{\mathcal{B}}, \mathbf{h}_2^{\mathcal{B}}, ..., \mathbf{h}_N^{\mathcal{B}}\}, \mathbf{h}_i^{\mathcal{B}} \in \mathcal{B}^d$, where $N$ is the number of nodes, and $d$ is the dimension of features.

We first map $\mathbf{h}^{\mathcal{B}}$ into the tangent space of origin $T_o\mathcal{M}$ using Eq. (3) that $\mathbf{h}^E = \log_{\boldsymbol{o}}^c(\mathbf{h}^{\mathcal{B}})$ to perform the following attention aggregation mechanism (**?**).

Next, we employ multi-head attention which is similar with (**?**), but we add adjacent feature $A_{ij}$ when computing attention $e_{ij}^k$ coefficients where $j \in \mathcal{N}_i$

$$e_{ij}^k = \text{leakyReLU}\left(\mathbf{W}_0^k[\mathbf{h}_i^E, \mathbf{h}_j^E, \mathbf{A}_{ij}]\right) \tag{14}$$

We normalize them across all choices of $j$ using the softmax function as follows:

$$\alpha_{ij}^k = \text{softmax}_j(e_{ij}^k) \tag{15}$$

Then, we concatenate features from $K$ head after computing the linear combination of the features corresponding to the normalized attention coefficients.

$$\mathbf{h}_i'^E = \Big\|_{k=1}^{K}\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \mathbf{W}_1^k \mathbf{h}_j^E\right) \tag{16}$$

[1] https://github.com/gordicaleksa/pytorch-GAT

We map the output $\mathbf{h}_i'^E$ of attention mechanism to the input $\mathbf{h}_i^B$ by using an exponential function, which can be regarded as a residual connection (**?**) on the hyperbolic manifold, and finally we apply a manifold-preserving activation function $\sigma$ (e.g., ReLU or leaky ReLU) (**?**).

$$\mathbf{h}_{i,\text{out}}^B = \sigma(\exp_{\mathbf{h}_i^B}^c(PT_{\mathbf{o}\to\mathbf{h}_i^B}^c(\mathbf{h}_i'^E))) \tag{17}$$

## Hyperbolic Score-based Model

We implement a hyperbolic variational auto-encoder to generate the hyperbolic representation $\boldsymbol{X}^{\mathcal{B}}$ of nodes.

**Encoder.** The encoder first utilizes a hyperbolic embedding layer to convert input feature$\boldsymbol{X}^E$ of nodes into hyperbolic embeddings $\boldsymbol{X}^{0,\mathcal{B}}$. $(\boldsymbol{X}^{0,\mathcal{B}}, \boldsymbol{A})$ are then put into a $n$-layer HGAT to learn the structural information of the graph. Then the output $\boldsymbol{X}^{n,\mathcal{B}}$ is mapped to Euclidean space using Eq. (3) and fed into an MLP to produce mean $\mu$ and distortion $\sigma$. We use Eq. (2) to map $\mu$ to Poincar ball and obtain $\boldsymbol{X}^{\mathcal{B}}$.

**Decoder.** Firstly, we obtain the hyperbolic node feature $\boldsymbol{X}^{\mathcal{B}} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{|V|}\}$ where $\boldsymbol{x}_i \in \mathcal{B}$ via reparameterization. After passing $(\boldsymbol{X}^{\mathcal{B}}, \boldsymbol{A})$ to $k$ layers of HGAT, we get the output $\boldsymbol{X}^{k,\mathcal{B}}$. Then, the decoder uses a centroid-distance layer (**?**) to convert $\boldsymbol{X}^{k,\mathcal{B}}$ into the Euclidean space. The centroid-distance layer is equipped with a list of learnable centroids $\mathcal{C} = [\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_{|\mathcal{C}|}]$ with each $\boldsymbol{c}_i \in \mathcal{B}$. By computing the pairwise distance $\psi_{ij}$ between $c_i$ and $x_j$, we obtain the Euclidean features while utilizing the hyperbolic metric and use it to recover $\boldsymbol{X}^E$.

**Score Model.** We implement the hyperbolic score-based model $\boldsymbol{s}_{\theta,t}(\bar{\boldsymbol{G}}_t)$ to estimate $\nabla_{\boldsymbol{X}_t^{\mathcal{B}}} \log p\left(\boldsymbol{X}_t^{\mathcal{B}}, \boldsymbol{A}_t\right)$. The computing procedures of $\boldsymbol{s}_{\theta,t}$ consists:

$$\begin{aligned} \boldsymbol{V}' &= \textbf{MLP}\left(\left[\left\{\log_{\boldsymbol{o}}^c\left(\textbf{HGAT}\left(\boldsymbol{H}_{\boldsymbol{i}}^{\boldsymbol{\mathcal{B}}}, \boldsymbol{A}_{\boldsymbol{t}}\right)\right)\right\}_{i=0}^{K}\right]\right) \\ \boldsymbol{V} &= \log_{\boldsymbol{X}_t^{\mathcal{B}}}^c\left(\exp_o^c\left(\boldsymbol{V}'\right)\right) * \textbf{MLP}\left(t, \lambda_{\boldsymbol{X}_t^{\mathcal{B}}}^c\right) \end{aligned} \tag{18}$$

where $\boldsymbol{V} = \boldsymbol{s}_{\theta,t}(\bar{\boldsymbol{G}}_t)$ is in the tangent space of $\boldsymbol{X}_t^{\mathcal{B}}$, $\boldsymbol{H}_{i+1}^{\mathcal{B}} = \text{HGAT}(\boldsymbol{H}_i^{\mathcal{B}}, A_t)$ with $\boldsymbol{H}_0^{\mathcal{B}} = \boldsymbol{X}_t^{\mathcal{B}}$ being given, $[\cdot]$ denotes the concatenation operation, and $K$ denotes the number of HGAT layers. Unlike the score in Euclidean space, we find that the norm of hyperbolic scores is related to the Poincar metric $g_x^{\mathcal{B}} = (\lambda_x^c)^2 g_x^E$, but the metric varies with $x$, which makes it difficult to match the score. We use an MLP to rescale the output according to the time step and the conformal factor $\lambda_x^c$, which empirically helps the model capture the norm of the true score.

We use the score-based model $\boldsymbol{s}_{\phi,t}(\bar{\boldsymbol{G}}_t)$ to estimate $\nabla_{\boldsymbol{A}_t} \log p\left(\boldsymbol{X}_t^{\mathcal{B}}, \boldsymbol{A}_t\right)$ as follows:

$$\boldsymbol{s}_{\phi,t}\left(\bar{\boldsymbol{G}}_t\right) = \textbf{MLP}\left(\left[\{(\boldsymbol{H}_{\boldsymbol{i}}^{\boldsymbol{E}})\}_{i=0}^{L}\right]\right) \tag{19}$$

where $\boldsymbol{H}_{i+1}^E = \text{GNN}(\boldsymbol{H}_i^E, A_t)$ with $\boldsymbol{H}_0^E$ obtained by a centroid-distance layer and $L$ is the number of GNN layers.

Algorithm 1: Hyperbolic PC sampling (VE SDE)

$x_N \sim \mathcal{N}_{\mathcal{B}_c^d}^{\text{W}}(0, \sigma_{max}^2\mathbf{I})$
**for** $i = N-1$ **to** $0$ **do**
  $x_i' \leftarrow \exp_{\boldsymbol{x_{i+1}}}^c((\sigma_{i+1}^2 - \sigma_i^2)\boldsymbol{s}_\theta(x_{i+1}, A_{i+1}))$
  $x_i \sim \mathcal{N}_{\mathcal{B}_c^d}^{\text{W}}(x_i', \sqrt{\sigma_{i+1}^2 - \sigma_i^2}\mathbf{I})$
  **for** $j = 1$ **to** $M$ **do**
    $x_i'' \leftarrow \exp_{\boldsymbol{x_i}}^c(\epsilon_i \boldsymbol{s}_\theta(x_i, A_i))$
    $x_i \sim \mathcal{N}_{\mathcal{B}_c^d}^{\text{W}}(x_i'', \sqrt{2\epsilon_i}\mathbf{I})$
  **end for**
**end for**
**return** $x_0$

Algorithm 2: Hyperbolic PC sampling (VP SDE)

$x_N \sim \mathcal{N}_{\mathcal{B}_c^d}^{\text{W}}(0, \mathbf{I})$
**for** $i = N-1$ **to** $0$ **do**
  $x_i' \leftarrow (2 - \sqrt{1-\beta_{i+1}}) \otimes_c \boldsymbol{x_{i+1}}$
  $x_i' \leftarrow \exp_{\boldsymbol{x_i'}}^c(\beta_{i+1}\boldsymbol{s}_\theta(x_{i+1}, A_{i+1}))$
  $x_i \sim \mathcal{N}_{\mathbb{B}_c^d}^{\text{W}}(x_i', \sqrt{\beta_{i+1}}\mathbf{I})$
  **for** $j = 1$ **to** $M$ **do**
    $x_i'' \leftarrow \exp_{\boldsymbol{x_i}}^c(\epsilon_i \boldsymbol{s}_\theta(x_i, A_i))$
    $x_i \sim \mathcal{N}_{\mathcal{B}_c^d}^{\text{W}}(x_i'', \sqrt{2\epsilon_i}\mathbf{I})$
  **end for**
**end for**
**return** $x_0$

## Reverse Diffusion Sampling on Hyperbolic Space

We implement the Predictor-Corrector samplers (**?**) in hyperbolic space (see Algorithms 1 and 2). We use the Mbius scalar multiplication $\otimes_c$ (**?**) and the exponential map in Eq. (2) to keep the reverse diffusion process on the manifold. $\{\epsilon_i\}_{i=0}^{N-1}$ are step sizes for Langevin dynamics. After sampling, a pre-trained decoder from HVAE is used to recover $x_0$ back into the original node features, such as atom types.

# Experimental Results

In this section, we evaluate our method on generic graph datasets and molecular datasets, then compare it with the baselines.

## Generic Graph Generation

**Experimental Setup** We validate HGDM on four generic graph datasets: (1) **Ego-small**, 200 small ego graphs drawn from larger Citeseer network dataset (**?**), (2) **Community-small**, 100 randomly generated community graphs (**?**), (3) **Enzymes**, 587 protein graphs which represent the protein tertiary structures of the enzymes from the BRENDA database (**?**), and (4) **Grid**, 100 standard 2D grid graphs (**?**). We use the maximum mean discrepancy (MMD) to compare the distributions of graph statistics between the same number of generated and test graphs. Following **?**, we measure the distributions of degree, clustering coefficient, and the number of occurrences of orbits with 4 nodes.

| Dataset | Ego-small | | | | Community-small | | | | Enzymes | | | | Grid | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Info. | Real, $4 \leq \|V\| \leq 18$, $\delta = 0.25$ | | | | Synthetic, $12 \leq \|V\| \leq 20$, $\delta = 1$ | | | | Real, $10 \leq \|V\| \leq 125$, $\delta = 1.26$ | | | | Synthetic, $100 \leq \|V\| \leq 400$ , $\delta = 9.76$ | | | |
| Method | Deg. | Clus. | Orbit | Avg. | Deg. | Clus. | Orbit | Avg. | Deg. | Clus. | Orbit | Avg. | Deg. | Clus. | Orbit | Avg. |
| DeepGMG | 0.040 | 0.100 | 0.020 | 0.053 | 0.220 | 0.950 | 0.400 | 0.523 | - | - | - | - | - | - | - | - |
| GraphRNN | 0.090 | 0.220 | 0.003 | 0.104 | 0.080 | 0.120 | 0.040 | 0.080 | **0.017** | 0.062 | 0.046 | 0.042 | **0.064** | 0.043 | **0.021** | **0.043** |
| GraphAF | 0.03 | 0.11 | **0.001** | 0.047 | 0.18 | 0.20 | 0.02 | 0.133 | 1.669 | 1.283 | 0.266 | 1.073 | - | - | - | - |
| GraphDF | 0.04 | 0.13 | 0.01 | 0.060 | 0.06 | 0.12 | 0.03 | 0.070 | 1.503 | 1.061 | 0.202 | 0.922 | - | - | - | - |
| GraphVAE | 0.130 | 0.170 | 0.050 | 0.117 | 0.350 | 0.980 | 0.540 | 0.623 | 1.369 | 0.629 | 0.191 | 0.730 | 1.619 | **0.0** | 0.919 | 0.846 |
| GNF | 0.030 | 0.100 | **0.001** | 0.044 | 0.200 | 0.200 | 0.110 | 0.170 | - | - | - | - | - | - | - | - |
| EDP-GNN | 0.052 | 0.093 | 0.007 | 0.051 | 0.053 | 0.144 | 0.026 | 0.074 | 0.023 | 0.268 | 0.082 | 0.124 | 0.455 | 0.238 | 0.328 | 0.340 |
| GDSS | 0.021 | 0.024 | 0.007 | 0.017 | 0.045 | 0.086 | 0.007 | 0.046 | 0.026 | 0.061 | 0.009 | **0.032** | 0.111 | 0.005 | 0.070 | 0.062 |
| HGDM (ours) | **0.015** | **0.023** | 0.003 | **0.014** | **0.017** | **0.050** | **0.005** | **0.024** | 0.045 | **0.049** | **0.003** | **0.032** | 0.137 | 0.004 | 0.048 | 0.063 |
| Improve. over GDSS | 28.6% | 4.2% | 57.1% | 17.6% | 62.2% | 41.9% | 28.6% | 47.8% | -73.1% | 19.7% | 66.7% | 0.0% | -23.4% | 20.0% | 31.4% | -1.6% |

Table 1: Generation results on the generic graph datasets. Results of the baselines are taken from published papers (**???**). Hyphen (-) denotes that the results are not provided in the original paper. The best results are highlighted in bold and the underline denotes the second best. (lower is better). We report the mean graph hyperbolicity values $\delta$ of all datasets. Due to the space limitation, we provide the standard deviations in Appendix.

We also report mean graph hyperbolicity values $\delta$ (lower is more hyperbolic) of the datasets by computing Gromovs $\delta$-hyperbolicity (**?**), a notion from group theory that measures how tree-like a graph is. The lower $\delta$, the more hyperbolic the graph dataset, and $\delta = 0$ for trees.

**Baselines** We compare our proposed method against the following generative models. **GraphVAE** (**?**) is a VAE-based model. **DeepGMG** (**?**) and **GraphRNN** (**?**) are autoregressive RNN-based models. **GNF** (**?**) is a one-shot flow-based model. **GraphAF** (**?**) is an autoregressive flow-based model. **EDP-GNN** (**?**) and **GDSS** (**?**) are score-based models. **GraphDF** (**?**) is an autoregressive flow-based model that utilizes discrete latent variables.

**Results** Table 1 shows that HGDM significantly outperforms all baselines including GDSS, achieving optimal results in most metrics. In particular, HGDM outperforms GDSS with a 37.1% decrease of MMD on average in Ego-small and Community-small, indicating that it has unique advantages in generating graphs with low hyperbolicity ($\delta \leq 1$). Although the graphs in Enzymes deviate from a power-law distribution, HGDM still maintains the same results as GDSS in terms of average statistics. The graphs in Grid deviate more from the hierarchical structure ($\delta = 9.76$) and are closer to the Euclidean topology with 0 curvature, in which case the autoregressive model GraphRNN has an advantage over the score-based models such as GDSS. HGDM still manages to achieve comparable results to GDSS on average statistics and outperforms the other baselines except GraphRNN. This demonstrates the scalability of HGDM, which still has good modeling capability for graphs with high hyperbolicity. In general, we find that our model performs better in generating graphs with small $\delta$-hyperbolicity.

## Molecule Generation

**Datasets** Following the GDSS (**?**), we tested our model on the **QM9** (**?**) and **ZINC250k** (**?**) datasets to assess its ability to learn molecular distributions. QM9 dataset contains 134k stable small molecules with up to 9 heavy atoms (CONF). ZINC250k datasets encompass a total of 250k chemical compounds with drug-like properties. On average, these molecules are characterized by a larger size (with an average of 23 heavy atoms) and possess greater structural complexity when compared to the molecules in QM9. Following previous works (**??**), the molecules are kekulized by the RDKit library (**?**) with hydrogen atoms removed. We also report the mean graph hyperbolicity values $\delta$.

**Metrics** We sample $10,000$ molecules using our model and evaluate their quality with the following metrics. **Frchet ChemNet Distance (FCD)** (**?**) evaluates the distance between the training sets and generated sets by analyzing the activations of the penultimate layer of the ChemNet. **Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) MMD** (**?**) computes MMD between the generated molecules and test molecules while considering both node and edge features for evaluation. It is important to note that FCD and NSPDK MMD are crucial metrics that evaluate the ability to learn the distribution of the training molecules by measuring the proximity of the generated molecules to the distribution. Specifically, FCD assesses the ability in the context of chemical space, whereas NSPDK MMD measures the ability in terms of the graph structure. **Validity w/o correction**, used for fair comparing with (**?**), is the proportion of valid molecules without valency correction or edge resampling. **Time** measures the time for generating $10,000$ molecules in the form of RDKit molecules.

**Baselines** We use **GDSS** (**?**) as our main baseline. We also compare the performance of HGDM against **GraphAF** (**?**), **GraphDF** (**?**), MoFlow (**?**), **EDP-GNN** (**?**) and **GraphEBM** (**?**).

**Results** Table 2 shows that our method also exhibits excellent performance on molecular generation tasks. In particular, on the QM9 dataset, our method is optimal in all metrics and significantly outperforms our main comparator, GDSS. HGDM achieves the highest validity without the use of post-hoc valency correction. It shows that HGDM can effectively learn the valency rules of the molecules. HGDM also outperforms all baselines in NSPDK MMD and FCD, which indicates that not only does the HGDM efficiently learn the

| Dataset | QM9 | | | | ZINC250k | | | |
|---|---|---|---|---|---|---|---|---|
| Info. | $\delta = 0.7$ | | | | $\delta = 1$ | | | |
| Method | Val. w/o corr. (%) ↑ | NSPDK MMD ↓ | FCD ↓ | time (s) ↓ | Val. w/o corr. (%)↑ | NSPDK MMD ↓ | FCD ↓ | time (s) ↓ |
| GraphAF | 67 | 0.020 | 5.268 | $2.52e^3$ | 68 | 0.044 | **16.289** | $5.80e^3$ |
| GraphDF | 82.67 | 0.063 | 10.816 | $5.35e^4$ | 89.03 | 0.176 | 34.202 | $6.02e^3$ |
| MoFlow | 91.36 | 0.017 | 4.467 | **4.60** | 63.11 | 0.046 | 20.931 | $\mathbf{2.45e^1}$ |
| EDP-GNN | 47.52 | 0.005 | 2.680 | $4.40e^3$ | 82.97 | 0.049 | 16.737 | $9.09e^3$ |
| GraphEBM | 8.22 | 0.030 | 6.143 | $\underline{3.71e^1}$ | 5.29 | 0.212 | 35.471 | $\underline{5.46e^1}$ |
| GDSS* | <u>95.79</u> | <u>0.003</u> | <u>2.813</u> | $1.14e^2$ | **95.90** | <u>0.019</u> | <u>16.621</u> | $2.02e^3$ |
| HGDM (ours) | **98.04** | **0.002** | **2.131** | $1.23e^2$ | <u>93.51</u> | **0.016** | 17.69 | $2.23e^3$ |

Table 2: Generation results on the QM9 and ZINC250k dataset. Results of the baselines are taken from (**?**). The method with $*$ denotes that results are obtained by running open-source codes. The best results are highlighted in bold and the underline denotes the second best. We report the mean graph hyperbolicity values $\delta$ of the two datasets. Due to the space limitation, we show the results of validity, uniqueness, and novelty in the Appendix.

| Method | Val. w/o corr. (%)↑ | NSPDK MMD ↓ | FCD ↓ | time (s) ↓ |
|---|---|---|---|---|
| GDSS | 95.79 | 0.003 | 2.813 | $\mathbf{1.14e^2}$ |
| GDSS+AE | 95.95 | 0.003 | 2.615 | $1.17e^2$ |
| HGDM+hgcn(ours) | 96.64 | **0.002** | 2.331 | $1.48e^2$ |
| HGDM (ours) | **98.04** | **0.002** | **2.131** | $1.23e^2$ |

Table 3: Generation results of the variants of GDSS and HGDM on the QM9 dataset.

distribution of the graph structure from the hyperbolic latent space and benefit from it, but the generated molecules are also close to the data distribution in the chemical space. We also report the generation results of ZINC250k in Table 2. As we observed in the generic graph datasets, the performance of HGDM in FCD and validity is slightly degraded due to the fact that the ZINC250k dataset deviates more from the hyperbolic structure and has more complex chemical properties compared to QM9. However, HGDM still maintains the advantage of generation from a geometric structure perspective that outperforms all the baselines in NSPDK MMD. Overall HGDM achieves similar performance for GDSS on the ZINC250k dataset and outperforms all other baselines. The superior performance of HGDM on the molecule generation task validates the ability of our method to efficiently learn the underlying distribution of molecular graphs with multiple nodes and edge types.

## Ablation Study

We implement a variant of GDSS called GDSS+AE. It incorporates an autoencoder to generate the Euclidean embedding of nodes and uses GDSS to estimate the score function in Euclidean latent space. We use GDSS+AE to compare the effect of Euclidean and hyperbolic hidden spaces on graph diffusion. In addition, we tested HGDM using the HGCN layer as a building block, called HGDM+hgcn. The results of the above-mentioned variants in QM9 are provided in Table 3

**Necessity of Hyperbolic Hidden Space** We find that GDSS+AE is slightly improved in metrics compared to GDSS. It can be considered that the dense embedding generated by the auto-encoder helps to learn the distribution of the graph, but it is not comparable to our hyperbolic approach due to the restricted capacity growth of Euclidean space.

**Necessity of HGAT Layers** HGDM+hgcn is comparable to our model in most metrics, but the increase in time cost is higher compared to GDSS, whereas our method has only a slight increase in time, making it more suitable for scaling up to the task of generating large graphs. This demonstrates the importance of using our proposed HGAT layer.

## Conclusion

In this work, we proposed a two-stage Hyperbolic Graph Diffusion Model (HGDM) to learn better the distribution of graph data and a simple hyperbolic graph attention layer that reduces the extra time spent associated with hyperbolic methods. HGDM overcomes the limitations of previous graph generation methods and is closer to the nature of graphs. We have found experimentally that learning the distribution in the hyperbolic space is beneficial for the quality of generated graph with the power-law distribution. Experimental results in generic graph and molecule generation show that HGDM outperforms existing graph generation methods in most metrics, demonstrating the importance of learning the underlying manifold for graph generation tasks. Code is available at https://github.com/LF-WEN/HGDM


## Acknowledgments

This work is supported by the National Natural Science Foundation of China (42130112,62272170). This work is also supported by the "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software (22510750100), the General Program of Shanghai Natural Science Foundation (23ZR1419300) and KartoBit Research Network(KRN2201CA).


Ex dolorum quae, est ipsam ratione porro blanditiis totam quibusdam?Inventore nostrum veritatis asperiores deleniti alias architecto voluptatibus fugit, rem tenetur adipisci quam assumenda ipsa obcaecati iure nostrum distinctio eligendi, odio quis reprehenderit ratione architecto fuga odit, dignissimos quod tempore quasi ex?Quas commodi temporibus atque fugiat rerum pariatur tempore quia odit dolorum a, in quaerat eaque quam incidunt facere quae quisquam placeat,

recusandae deserunt natus?Vero maiores deleniti nisi iure incidunt unde et temporibus quos veritatis,