GEOGAE: SCALABLE GRAPH-LEVEL AUTOENCODING VIA HYPERBALL CLOUD REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Embedding structured objects into Euclidean spaces has enabled a wide range of successful machine learning applications. Such objects include words, documents, image patches, time series, and graph nodes. In contrast, embedding entire graphs remains a challenging problem. Existing methods either sustain the original order of the graph nodes or match the output nodes to the input ones, both of which create scalability issues. In this work, we propose a graph representation as a cloud of hyperballs, which allows us to define a specific—typically unique—node ordering. Based on this representation, we propose GeoGAE, an autoencoder, in which the Transformer encoder translates a hyperball cloud into a graph-level embedding, and the Transformer decoder translates the graph-level embedding back into the graph. This formulation enables the model to capture both the global graph structure and local relational patterns. We evaluate our method on multiple graph datasets, spanning various domains. The results demonstrate effectiveness of our method in encoding and reconstructing graphs from their embeddings.

1 INTRODUCTION

Fixed-size embeddings have proven very useful in a number of domains. Token embeddings are invaluable for natural language processing (Su et al., 2024). Image embeddings are indispensable for image generation (Ramesh et al., 2022). Time series embeddings are a useful tool in prediction (Foumani et al., 2024). Embeddings are especially useful if any object in its domain can be reconstructed from its fixed-size vector representation. Then, the generation, transformation and optimization of these objects boil down to an equivalent operation in $\mathbb{R}^m$, for a fixed m.

The field of graph neural networks (GNNs; Corso et al. 2024; Khemani et al. 2024) provides a plethora of methods for conditional graph generation (Ayadi et al., 2024; Bian et al., 2024; Gao

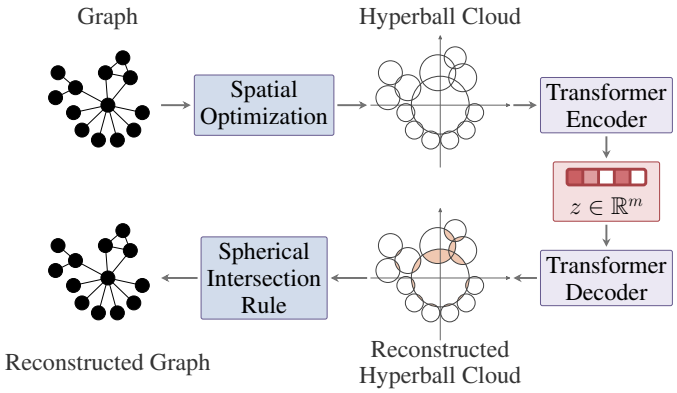

Figure 1: **Overview of GeoGAE.** The discrete input graph is first mapped into a continuous hyperball cloud representation via spatial optimization and deterministic node ordering. A Transformer encoder then compresses this sequence into a fixed-size graph-level embedding $z \in \mathbb{R}^m$. In the decoding path, a Transformer decoder reconstructs the hyperball cloud from the latent vector, and the final discrete graph topology is exactly recovered using a static spherical intersection rule.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

et al., 2025; Hou et al., 2024; Ketata et al., 2025; Li et al., 2025; Liu et al., 2025; Wang et al., 2025; Wen & Yu, 2025; Wesego, 2025; You et al., 2024; Zhou et al., 2024). However, the main purpose of these methods is typically to generate a diverse set of graphs possessing certain properties, rather than to reconstruct a specific, unique graph from a given latent embedding. While some of these methods (Wesego, 2025; You et al., 2024; Zhou et al., 2024) use node (and edge) embeddings as an auxiliary graph representation that could be padded to form a graph-level vector, this approach establishes a strict limit on the graph size.

Few works focus on a reversible, scalable transformation of fixed-size vectors into graphs. Methods proposed by Małkowski et al. (2023) and Bresson et al. (2026) include the node order in the graph embedding. In contrast, Winter et al. (2021) and Krzakala et al. (2025) ensure the graph embedding is independent of the node order, but at the cost of requiring complex matching mechanisms between the input and output graph nodes.

In this paper, we bridge this gap by designing a graph representation as a cloud of hyperballs in Euclidean space. This allows us to define a unique order of nodes. Our proposed graph-level autoencoder, intuitively described in Figure 1, uses the Transformer encoder to transform the sequence of nodes represented that way into a fixed-size embedding, and the Transformer decoder to transform the embedding back to the sequence of hyperballs. The hyperballs are translated into the discrete graph structure with a static rule. While representing nodes in 2- or 3-dimensional Euclidean space is a technique applied for molecular graph generation (Ketata et al., 2025), we generalize the dimensionality to any $d \in \mathbb{N}$.

Contributions. The main contributions of this work are threefold:

1. **Continuous Hyperball Representation:** We introduce a continuous graph representation as hyperball clouds in Euclidean space, enabling canonical and deterministic node ordering.
2. **Scalable Graph Autoencoder:** We propose GeoGAE, a Transformer-based autoencoder that compresses variable-sized hyperball clouds into fixed-size latent vectors $z \in \mathbb{R}^m$ and reconstructs original graph topologies using a static geometric rule.
3. **Empirical Superiority:** We conduct extensive evaluations across diverse benchmark domains, demonstrating that GeoGAE achieves state-of-the-art topology reconstruction and scales to large graphs where existing baselines suffer from out-of-memory errors or severe degradation.

2 Related work

Graph to vector models. Graph-level representation learning aims to encode an entire graph into a fixed-dimensional vector for tasks such as classification or regression (Khoshraftar & An, 2024). A standard approach involves processing the graph through message-passing GNN layers (Veličković et al., 2018; Xu et al., 2019; Chen et al., 2020) and applying a readout or pooling operation to aggregate node-level features into a global embedding. This includes simple pooling (e.g., sum, mean) Kipf & Welling (2016), attention-based pooling (Lee et al., 2019), or hierarchical coarsening strategies (Ying et al., 2018; Zhang et al., 2019; Ma et al., 2019). An alternative strategy introduces virtual nodes connected to all other nodes to directly capture global graph properties (Li et al., 2017; Xu et al., 2019; Brossard et al., 2021).

While these methods excel at predictive downstream tasks, their aggregation mechanisms inherently compress and entangle structural information, creating a severe information bottleneck. This irreversible loss of precise topological detail renders standard pooling and virtual node architectures poorly suited for exact graph reconstruction from the latent space. In contrast, our proposed GeoGAE framework replaces lossy aggregation with a reversible hyperball cloud representation, explicitly designed to preserve exact graph topology within a fixed-size vector.

Graph generation. While the primary focus of GeoGAE is autoencoding, our decoder effectively functions as a continuous-to-discrete graph generator. Most contemporary graph generation methods aim to learn the data distribution to sample novel structures, rather than exactly reconstructing specific inputs. Early approaches, such as GraphRNN (You et al., 2018), generated adjacency matrices sequentially using recurrent architectures. Recently, diffusion models have dominated the field

(Liu et al., 2023), generating adjacency matrices similarly to images (Liu et al., 2025), or utilizing autoregressive denoising steps (Kong et al., 2023). Several methods also generate node coordinates in Euclidean (Gao et al., 2025; You et al., 2024) or non-Euclidean spaces (Fu et al., 2024), while others apply continuous-time diffusion over discrete states (Xu et al., 2024). However, unlike our framework, which establishes a deterministic and reversible mapping between a unique graph and its latent embedding, these generative models primarily map prior noise distributions to graph distributions, making them unsuitable for exact topology preservation.

Graph autoencoders with embeddings in $\mathbb{R}^{n\times d}$. These architectures implement the general structure of Variational Autoencoder (VAE: Kingma & Welling 2014), with an encoder that transforms the input graph into a matrix of embeddings of its nodes. A decoder performs the reverse transformation. Both encoder and decoder can take the form of message-passing GNN (Imrie et al., 2020; Huang et al., 2022; Hou et al., 2024).

Graph autoencoders with embeddings in $\mathbb{R}^m$. Our goal in this paper is to transform graphs of various sizes to embeddings of fixed size, m, and transform these embeddings back to graphs. A simple way to achieve it is to assume that the number of vertices is bounded by a fixed $n_{\max}$ and one of the autoencoders from the previous paragraph with an embedding of size $n_{\max}\times d$ with some kind of padding for smaller graphs. This idea is applied in GraphVAE (Simonovsky & Komodakis, 2018) for small graphs with $n_{\max}$ up to 38. Hy & Kondor (2021) introduced MGVAE – an autoencoder whose encoder recursively identifies clusters in the graph, and replace them with nodes of a higher order graph. Eventually the input graph is reduced to a fixed size embedding. The decoder recursively unpacks this embedding to the input graph. Małkowski et al. (2023) presented ReGAE, an autoencoder where the encoder recursively rolls up growing parts of the adjacency matrix into a vector in $\mathbb{R}^m$, and the decoder, also recursively, unrolls the input matrix. While this architecture is able to reconstruct graphs with even thousands nodes, it is not node permutation equivariant which potentially leads to accuracy losses. In PIGVAE (Winter et al., 2021) and GRALE (Krzakala et al., 2025), the Transformer encoder produces permutation equivariant graph-level embedding and a similar decoder reconstructs the graph. These designs raise the problem of assigning the output nodes to the input ones. PIGVAE leverages a learnable assigning modules fed with the graph embedding. GRALE extends this idea by combining an optimal-transport-based reconstruction loss with a Sinkhorn-based matcher which significantly improved stability and expressivity. GraViti (Bresson et al., 2026) use a similar Transformer based architecture and simply resigns from node ordering equivariance. In this paper, we dodge the problem of node order equivariance by introducing an usually unique node order. Also, we use iteratively the transformer decoder to produce the output graph of an arbitrary size.

3 METHOD

3.1 FORMAL PROBLEM DESCRIPTION

We consider undirected, unweighted graphs without self-loops. Formally, a graph is defined by a set of vertices, V, and a set of edges $E \subseteq \{\{v,u\}|v,u\in V, v\neq u\}$. We denote $n=|V|$ and optionally consider features of vertices and edges. Given a set $\mathcal{G}$ of such graphs and a target dimension $m\in\mathbb{N}$, our objective is to learn an encoder : $\mathcal{G}\to\mathbb{R}^m$ that compresses the graph into a fixed-size vector $z\in\mathbb{R}^m$, and a decoder : $\mathbb{R}^m\to\mathcal{G}$ that reconstructs this graph. Both mappings are end-to-end trained neural networks to minimize a reconstruction error, generalizing effectively to unseen graphs.

3.2 MODEL OVERVIEW

Our proposed framework, GeoGAE, is illustrated in Figure 1. The pipeline first maps an input graph into a continuous hyperball cloud representation (details described in Section 3.3), where each hyperball corresponds to a graph node. Next, a Transformer encoder compresses this geometric sequence into a fixed-size latent vector $z\in\mathbb{R}^m$, constructed by concatenating C learned class (`[CLS]`) tokens. Because these class tokens attend to the graph nodes without inter-`[CLS]` attention, each token is encouraged to specialize in capturing distinct topological properties.

In the decoding phase, a Transformer decoder interprets the latent embedding z to sequentially reconstruct the hyperball cloud. Finally, the reconstructed cloud is mapped back into a discrete graph using the static spherical intersection rule (Section 3.3). The entire architecture is trained end-to-end to encourage exact topological reconstruction.

3.3 HYPERBALL CLOUD

Let us consider representing graphs in a Euclidean space (Agrawal et al., 2021; Nowak et al., 2024). In this formulation, graph nodes are mapped to spatial objects whose geometric distances reflect the underlying topological distances. The following proposition demonstrates that such a representation can be topologically lossless.

Proposition 1. *A graph with n vertices can be losslessly represented as n hyperballs of equal size in an n^2-dimensional space, such that two nodes are adjacent if and only if their corresponding hyperballs intersect.*

The proof of Proposition 1 is provided in Appendix A.

While lossless representation using equal-sized hyperballs is theoretically guaranteed, it requires an excessively high-dimensional space (n^2). This dimensionality bottleneck can be dramatically reduced by allowing hyperball radii to vary. Consider the hub topology in Figure 1, where a central node connects to 10 neighbors, with only one pair of neighbors being mutually adjacent. If all 11 nodes were represented by equal-sized hyperballs, packing 10 balls around the central one in low dimensions would geometrically force unwanted overlaps among the neighbors. Allowing variable radii bypasses this packing constraint, enabling exact, lossless representation even in low-dimensional space.

Translation of a Graph to a Hyperball Cloud. Graph nodes can be embedded in an Euclidean space as follows. Let $d \in \mathbb{N}$ be the dimension of this space, and $x_1^g, \ldots, x_n^g \in \mathbb{R}^d$ be the geometric embeddings of the nodes. Let x^g denote the matrix that gathers them. An embedding $x_i^g = [x_{i,1}^c, \ldots, x_{i,d-1}^c, r_i]$ represents a hyperball in $(d-1)$-dimensional space with center x_i^c and radius $f(r_i)$, where $f : \mathbb{R} \to \mathbb{R}_+$ is a strictly increasing function (e.g., Softplus). Adjacent nodes correspond to intersecting balls, whereas non-adjacent nodes correspond to disjoint ones. The geometric embeddings are obtained by minimizing the loss function $\mathcal{L}_{\text{sphere}}$:

$$x^g = \arg\min_{x^g} \mathcal{L}_{\text{sphere}}(x^g) \tag{1}$$

$$p_{i,j} = \sigma\left(\frac{0.75\,(f(r_i) + f(r_j)) - \|x_i^c - x_j^c\|}{T}\right) \tag{2}$$

$$\mathcal{L}_{\text{sphere}}(x^g) = \sum_{i,j} -\widehat{\alpha}_{i,j}\,(1 - \widehat{p}_{i,j})^{\gamma} \log(\widehat{p}_{i,j}), \tag{3}$$

where $\|\cdot\|$ denotes the standard L_2 Euclidean norm, σ is the sigmoid function, $T > 0$ is a temperature scaling constant. The topological distance between nodes i and j is denoted by $D_{i,j}$. The target probability $\widehat{p}_{i,j}$ and class weight $\widehat{\alpha}_{i,j}$ are defined as $\widehat{p}_{i,j} = p_{i,j}$ and $\widehat{\alpha}_{i,j} = \alpha$ for adjacent nodes ($D_{i,j} = 1$), and $\widehat{p}_{i,j} = 1 - p_{i,j}$ and $\widehat{\alpha}_{i,j} = 1 - \alpha$ otherwise. The loss $\mathcal{L}_{\text{sphere}}$ treats hyperball intersection as a binary edge classification task. The class weight α mitigates class imbalance by prioritizing positive (adjacent) pairs, while the focusing parameter γ down-weights well-classified node pairs to concentrate optimization on topologically ambiguous cases.

The optimization objective (3) is invariant to rigid Euclidean transformations (translations and rotations); hence, its solution is not unique. To establish a canonical, permutation-invariant representation, we normalize the resulting geometric embeddings through the following steps:

1. We center the point cloud by shifting the coordinates such that the mean of all hyperball centers is zero ($\sum_i x_i^c = 0$).
2. We perform Singular Value Decomposition (SVD) on the centered spatial coordinates to find the principal axes, and re-express the embeddings within this new coordinate system.
3. To resolve reflectional ambiguities along the principal axes, we evaluate the third moment (skewness) of the projected coordinates. Specifically, for any axis $k \in \{1, \ldots, d-1\}$, if $\sum_i (x_{i,k}^c)^3 < 0$, we negate the k-th coordinate across all node embeddings.

Barring highly symmetrical graph topologies, this deterministic normalization yields a unique set of geometric embeddings for any given graph. In our experiments we choose the smallest geometric embedding size which scores 100% F1 on all dataset graphs.

Translation of Hyperball Cloud to Graph. Geometric embeddings that minimize (3) enable determining if there is an edge between the i-th and j-th node using a direct intersection rule. Given the hyperball centers x_i, x_j and their raw radii r_i, r_j, the margin is calculated as the difference between the scaled sum of the radii and the Euclidean distance between the centers:

$$m_{i,j} = 0.75(f(r_i) + f(r_j)) - \|x_i - x_j\|_2$$

where f denotes the softplus function. The decoding rule is then strictly defined by the sign of the margin:

$$\text{if } m_{i,j} > 0, \text{ then there is an edge; else, not.} \tag{4}$$

During training, the margin $m_{i,j}$ is scaled and forms logits for the binary cross-entropy and focal loss terms. At inference, the rule explicitly relies on the geometric intersection ($m_{i,j} > 0$), eliminating the need to optimize an arbitrary threshold on the training set.

3.4 Node ordering and bundles

Node ordering. The node ordering procedure begins by computing the geometric center (mean) of all hyperball centers. The first node in the sequence is chosen as the one whose hyperball center is closest to this global center. The consecutive nodes are then selected greedily by minimizing a distance to the previously selected node, and an additional penalty. Specifically, the Euclidean distance between ball centers is scaled by the average of their radii, and incremented by a scaled base-2 logarithm of the candidate node's distance to the global center. This ensures that nodes are ordered primarily by their relative spatial proximity and sizes, with a slight preference for nodes closer to the center of the entire structure. Algorithm 1 in Appendix B presents the details of this procedure. Its result is denoted by $(y_1^g, \ldots, y_n^g)$.

Bundles. To ensure scalability and mitigate the quadratic computational complexity inherent to the self-attention mechanism in Transformers, consecutive nodes from the ordered sequence are grouped into contiguous blocks called *bundles* of size $b \in \mathbb{N}$. When the total number of nodes N is not evenly divisible by b, the remaining slots in the final bundle are padded by repeating the geometric embedding of the last node. Each bundle of b geometric embeddings in $\mathbb{R}^{b \times d}$ is flattened and projected into a single d_{tok}-dimensional sequence token using a trainable Multilayer Perceptron (MLP) (Bundle $\rightarrow$ Token). This enables nodes within a single bundle to be processed by the encoder and generated by the decoder simultaneously. For smaller datasets where scalability is not a bottleneck, we set $b = 1$, which bypasses bundling and represents each node with an individual token. See Appendix B.2 for exact tensor dimensions and decoder de-bundling details.

3.5 Architecture

Encoder. The encoder of GeoGAE (Figure 2) compresses the input graph into a fixed-size latent representation through a streamlined pipeline. First, the graph is mapped into a hyperball cloud, deterministically ordered, and grouped into $T = \lceil n/b \rceil$ contiguous bundles of size b (Sections 3.3–3.4). An MLP, denoted ϕ, maps each bundle into a sequence token (details provided in Appendix B.2). To preserve relative positional information without explicit absolute position embeddings, Rotary Positional Embeddings (RoPE) are applied directly to queries and keys within each self-attention layer. The sequence is prepended with C trainable `[CLS]` tokens and processed by the Transformer encoder (attention masking details are provided in Appendix B.3). Finally, the output representations of the C `[CLS]` tokens are extracted and concatenated to form the global graph embedding $z \in \mathbb{R}^m$, optionally followed by a VAE reparameterization.

Decoder. The Transformer decoder (Figure 3) reconstructs the hyperball cloud by conditioning on the latent vector z via cross-attention over sequence queries. Instead of a standard linear-softmax head, a prediction module ψ (detailed in Appendix B.2) processes each decoder output token to yield two distinct outputs: (i) a matrix of b geometric node embeddings containing their spatial centers and radii, and (ii) b independent binary stopping logits. Each logit corresponds to an individual node

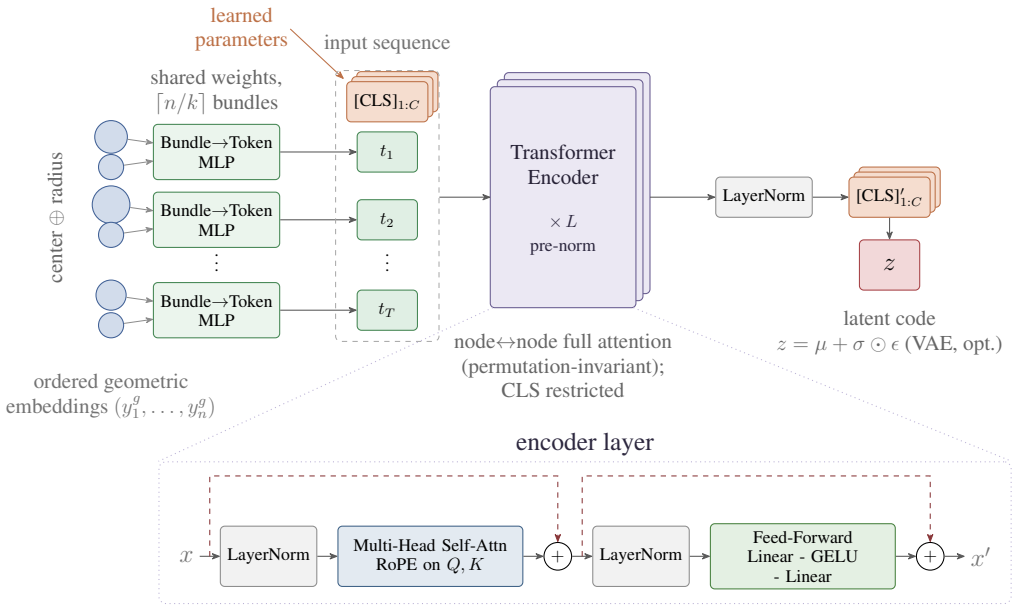

Figure 2: Geometric graph transformer encoder. The input graph is represented as an ordered sequence of hyperballs and mapped into sequence tokens. A full, permutation-invariant self-attention mechanism processes these tokens alongside C dedicated `[CLS]` tokens. The output representations of the `[CLS]` tokens are concatenated to yield the global graph embedding $z \in \mathbb{R}^m$.

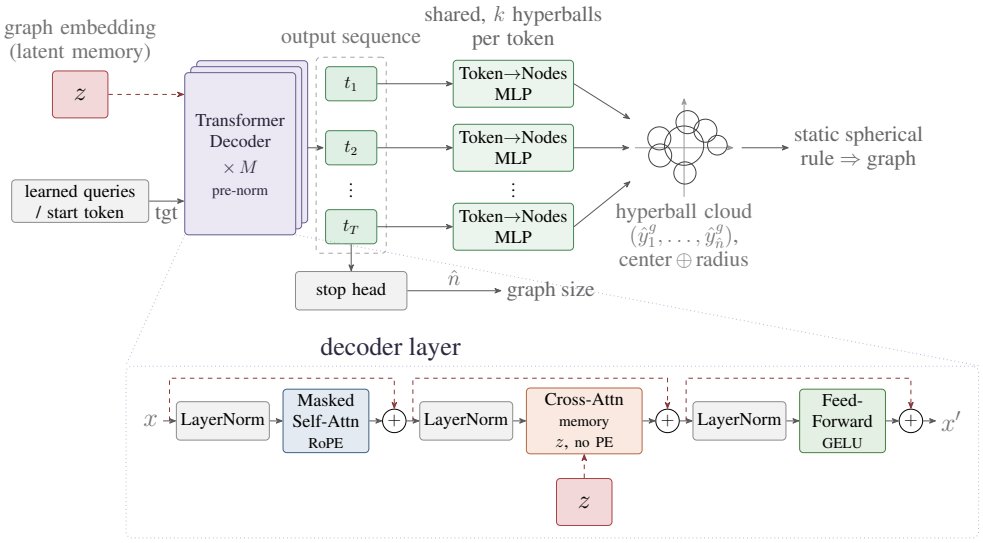

Figure 3: Geometric graph transformer decoder. Acts as a generative module that reconstructs the graph. It uses the latent vector z as context (latent memory) within its *cross-attention* layers, combining it with learned queries. The decoder generates a sequence of output tokens. Next, Multilayer Perceptron heads (Token→Nodes MLP) map these tokens back into a hyperball cloud. Concurrently, a dedicated stop head predicts the target graph size, and a geometric mechanism (static spherical rule) reconstructs the final edges between the nodes.

within the bundle and predicts whether it terminates the graph. This per-node stopping mechanism provides exact node-level precision, allowing the model to halt generation and cleanly truncate the final bundle when the graph size is not a multiple of b.

Dataset	Graphs	Avg. nodes	Avg. edges
MUTAG	188	17.90	19.80
AIDS	2,000	15.69	16.20
IMDB-BIN	1,000	19.77	96.53
QM9	129,433	18.03	18.63
SYNTHETIC NEW	300	100.00	196.25
COLLAB	5,000	74.49	2,457.78
REDDIT-BIN	2,000	429.63	497.75

Table 1: Basic statistics of datasets used in our benchmark.

3.6 Multi-layer teacher forcing

Transformer decoders are typically trained via teacher forcing, utilizing the right-shifted ground truth sequence—achieved by prepending a special 'start' vector—to predict the full target sequence terminating with an 'end' vector. However, this paradigm is notoriously susceptible to exposure bias; the absence of ground truth data during inference causes cascading errors when the model is conditioned on its own flawed predictions.

To counteract this unwanted byproduct, we propose an iterative refinement strategy by stacking the decoder $K \in \mathbb{N}$ times (with weight sharing of the instances). The base instance is conditioned on the right-shifted target sequence, while each successive instance takes the output sequence of the underlying layer, similarly right-shifted by prepending the 'start' vector, as its input. All K instances are jointly supervised to output the complete sequence ending with the 'end' vector, effectively bridging the gap between the training and inference regimes.

3.7 Node and edge attributes

When present, node attributes are concatenated into each node's geometric embedding before bundling – this effectively widens the input that ϕ consumes. On the other side, the decoder produces output tokens, which are then consumed by a separate MLP trained to reconstruct each node's attributes.

Edge attributes are consumed directly by the encoder's self-attention, in every layer, rather than as a per-token input channel. When a pair of tokens is connected via an edge, the edge attribute is concatenated into the token's key, and then consumed by a dedicated per-layer linear projection that is taught to correct the key, and the value only for those pairs. In the inference mode, the decoder, after deconstructing the adjacency matrix of the output graph uses a separate MLP, which for each connected pairs of tokens, takes a vector of their sum and absolute difference as an input, and produces a vector of their edge attribute. During the training process, the decoder uses the ground truth graph edges.

The total loss minimized in the training of our proposed architecture is specified in Appendix B.

4 Experimental study

We evaluate GeoGAE across seven benchmark datasets spanning chemical, social, and synthetic domains. Our experiments assess topological and feature reconstruction fidelity, scalability on large graphs, and the impact of key architectural choices via ablations.

Benchmarks. We use seven datasets of graphs of various nature, including social, chemical and synthetic ones. Along with their basic statistics, they are listed in Table 1 and characterized more broadly in Appendix C. These datasets provide the raw graph structures, which we process to benchmark our model's ability to generate graph-level embeddings and reconstruct the input graphs. We divide these datasets into training, validation, and test subsets in a 70:15:15 ratio.

Baselines. Among the autoencoders capable of generalizing to unseen graphs—meaning they can be trained on one dataset and evaluated on another—are PIGVAE (Winter et al., 2021), ReGAE

Dataset	tok-d	emb	geom-d	encod	decod	lr	enc-h	dec-h	batch
MUTAG	64	128	4	1024×4	256×2	1e-3	8	1	2
AIDS	64	192	6	1024×4	512×2	1e-3	8	1	32
IMDB-BIN	64	192	6	512×8	1024×6	1e-3	2	1	32
QM9	64	192	4	1024×6	1024×4	1e-3	4	4	256
SYNTH. NEW	64	192	6	2048×8	1024×2	5e-4	8	4	8
COLLAB	96	576	9	256×8	2048×4	1e-3	4	1	32
REDDIT-BIN	144	1728	9	512×4	1024×4	1e-3	6	4	16

Table 2: Hyperparameters of GeoGAE: tok-d – token dimension, emb – graph embedding size (integer multiple of tok-d), geom-d – dimension of the geometric embedding, encod – width and depth (width $\times$ number of layers) of the encoder hidden layers, decod – width and depth (width $\times$ number of layers) of the decoder hidden layers, lr – learning rate, enc-h – number of attention heads in the encoder, dec-h – number of attention heads in the decoder, batch – batch size. All datasets except REDDIT-BINARY use a bundle size of 1; REDDIT-BINARY uses a bundle size of 4.

(Małkowski et al., 2023), and GRALE (Krzakala et al., 2025). Our experiments aim to train these models to encode and reconstruct graph topologies using a training subset, verifying their performance on a held-out test subset.

4.1 Training

For each dataset, graph embedding dimensions were scaled proportionally with average graph complexity, with GeoGAE latent sizes aligned directly with ReGAE to ensure a fair comparison on high-capacity architectures (Table 2). Hyperparameters for all baselines were systematically optimized via dataset-specific random search, as detailed in Tables 2, 9, 10, and 11. For PIGVAE and GRALE, embedding sizes were explicitly included in hyperparameter tuning; larger latent dimensions degraded performance or induced training instability. Throughout all experiments, the temperature parameter in Eq. 2 was fixed at $T = 0.4$.

To guarantee statistical reliability, all models were trained and evaluated across five independent runs using distinct fixed seeds and pre-computed data splits. We report the mean and standard deviation across these runs.

4.2 Results

Table 3 summarizes the edge reconstruction F1 scores, weighted by vertex count across test graphs. GeoGAE consistently matches or outperforms baseline architectures on smaller graphs while demonstrating superior scalability on larger benchmarks. On complex, high-density datasets such as REDDIT-BINARY and COLLAB, GeoGAE achieves 56.5% and 90.9% F1, respectively, whereas PIGVAE, ReGAE, and GRALE suffer from training instability or degenerate into trivial predictions (e.g., predicting all 0s or 1s).

Table 4 reports joint topology and feature reconstruction performance, using F1 for discrete attributes and RMSE for continuous properties. While baselines suffer severe trade-offs, such as GRALE's high feature error on QM9 (5.4 node RMSE) or PIGVAE's topological collapse (25.6% F1), GeoGAE maintains a robust balance. On AIDS, GeoGAE achieves superior feature fidelity across both categorical (85.0% edge F1) and continuous attributes (0.8 node RMSE) without sacrificing topology (80.4% F1). Comprehensive ablation studies evaluating key architectural components are provided in Appendix F.

Unlike existing baselines that assume fixed node bounds, GeoGAE dynamically predicts graph cardinality. As detailed in Table 5, the resulting node-count errors remain minimal across all datasets (e.g., 0.00 on QM9 and 0.23 on AIDS). To strictly penalize dimension mismatches, predicted or target adjacency matrices with incorrect sizes are zero-padded prior to computing the F1 score.

5 Conclusions

In this paper, we introduced GeoGAE, a scalable graph-level autoencoder grounded in a novel hyperball cloud representation. By mapping discrete topologies into continuous geometric space with

Dataset\method	PIGVAE [%]	ReGAE [%]	GRALE [%]	GeoGAE [%]
MUTAG	43.5 ± 14.9	60.1 ± 3.5	6.9 ± 3.8	$\mathbf{61.3 \pm 5.4}$
AIDS	60.2 ± 4.9	83.6 ± 2.6	1.2 ± 0.6	$\mathbf{84.1 \pm 1.7}$
IMDB-BIN	87.3 ± 2.0	90.0 ± 2.6	54.8 ± 7.7	$\mathbf{94.0 \pm 1.0}$
QM9	0.0 ± 0.0	$\mathbf{99.9 \pm 0.0}$	98.6 ± 0.5	99.4 ± 0.2
SYNTHETIC NEW	0.4 ± 0.6	16.1 ± 3.0	7.3 ± 0.7	$\mathbf{50.1 \pm 0.4}$
COLLAB	52.2 ± 5.0	78.0 ± 2.4 †	OOM	$\mathbf{90.9 \pm 0.7}$
REDDIT-BIN	OOM	53.0 ± 2.0 ‡	OOM	$\mathbf{56.5 \pm 0.5}$

Table 3: Comparison of methods: F1 score on the test set. After $\pm$ we put the standard deviation. †Score from 3 seeds, other 2 got NaN loss. ‡Score taken as reported in the original paper, due to training instability.

Method	Dataset	Topo F1 [%]	Node F1 [%]	Node RMSE	Edge F1 [%]	Edge RMSE
PIGVAE	MUTAG	18.0 ± 9.7	66.3 ± 9.5	N/A	$\mathbf{71.2 \pm 22.8}$	N/A
	AIDS	41.9 ± 2.9	56.7 ± 14.6	1.5 ± 0.3	77.7 ± 6.3	N/A
	QM9	25.6 ± 15.9	N/A	$\mathbf{0.5 \pm 0.0}$	N/A	$\mathbf{0.1 \pm 0.0}$
GRALE	MUTAG	20.1 ± 18.3	$\mathbf{76.6 \pm 25.1}$	N/A	56.3 ± 15.7	N/A
	AIDS	0.1 ± 0.1	8.3 ± 1.1	4.3 ± 0.3	35.5 ± 1.1	N/A
	QM9	$\mathbf{94.1 \pm 0.7}$ †	N/A	5.4 ± 0.0 †	N/A	6.8 ± 0.3 †
GeoGAE	MUTAG	$\mathbf{62.3 \pm 3.0}$	$\mathbf{75.5 \pm 5.9}$	N/A	58.7 ± 2.8	N/A
	AIDS	$\mathbf{80.4 \pm 2.6}$	$\mathbf{62.2 \pm 5.3}$	$\mathbf{0.8 \pm 0.1}$	$\mathbf{85.0 \pm 1.7}$	N/A
	QM9	91.3 ± 1.2	N/A	$\mathbf{0.5 \pm 0.0}$	N/A	0.3 ± 0.1

Table 4: Comparison of methods: Topological, Node Features, and Edge Features reconstruction quality on the test set. F1 (higher is better) scores the categorical (one-hot) part of a feature block; RMSE (lower is better) scores its continuous part. N/A indicates that a metric is not applicable due to the nature of the features in a given dataset: F1 is N/A for purely continuous features (e.g., QM9 nodes and edges), while RMSE is N/A for purely categorical features (e.g., MUTAG node features, and edge features for MUTAG and AIDS). After $\pm$ we put the standard deviation. † denotes score obtained from 4 seeds, 1 seed got NaN loss.

Dataset	Mean size error	Dataset	Mean size error
MUTAG	0.12 ± 0.08	AIDS	0.23 ± 0.09
IMDB-BIN	0.27 ± 0.05	QM9	0.01 ± 0.00
SYNTHETIC NEW	0.00 ± 0.00	COLLAB	1.45 ± 0.34
REDDIT-BIN	16.25 ± 1.50		

Table 5: GeoGAE: Mean size error — the average absolute size difference between the target and predicted graphs over the average target graph size. Calculated for experiments for pure graph topology deconstruction, without features encoding and decoding enabled.

deterministic node ordering, GeoGAE resolves the longstanding node-matching bottleneck and size limitations of existing models. Integrated with a Transformer architecture, our framework successfully compresses variable-sized graphs into fixed-dimensional vectors $z \in \mathbb{R}^m$ while enabling high-fidelity topological reconstruction. Across diverse benchmarks, GeoGAE matches or exceeds state-of-the-art performance on most datasets, demonstrating unprecedented scalability on large graphs where prior methods fail. This establishes a robust foundation for latent-space graph generation, transformation, and optimization.

Limitations. Mapping a discrete graph to a hyperball cloud requires solving a continuous optimization problem per graph, which adds a pre-processing overhead. This, however is done only once per dataset, and can be considered a new node's feature. While our canonical ordering effectively mitigates the permutation ambiguity for most topologies, highly symmetrical graphs can remain sensitive to minor spatial perturbations during SVD normalization and greedy ordering. Scaling to large graphs introduces challenges in precise graph size prediction and topology preservation under node bundling. Addressing end-to-end differentiable hyperball initialization and symmetry-aware ordering remains a promising direction for future work.

Reproducibility Statement

We have made all efforts to ensure the full reproducibility of both our theoretical claims and empirical results.

Theoretical Reproducibility: The mathematical guarantee that any graph can be losslessly represented as a set of intersecting hyperballs is formally stated in Proposition 1, with a constructive proof provided in Appendix A. The deterministic node ordering mechanism is explicitly detailed in Algorithm 1 (Appendix B.1).

Empirical Reproducibility:

- **Hyperparameters & Training:** All hyperparameters for GeoGAE and the baseline methods (PIGVAE, ReGAE, GRALE) are comprehensively documented in Tables 2, 9, 10, and 11, respectively. We explicitly report batch sizes, embedding dimensions, network depths, learning rates, and weight decays.
- **Compute & Infrastructure:** The exact hardware specifications (including NVIDIA A100 GPUs and AMD EPYC processors) and software environments utilized across four computational clusters are detailed in Appendix D (Table 7). To aid in reproducing our training pipelines, we report the estimated end-to-end execution times per dataset on a single A100 GPU in Table 8 (e.g., $\sim$38 hours for QM9 with feature decoding).
- **Codebase:** The complete source code, environment specifications, and training scripts with execution instructions are provided in the supplementary materials.

LLM Usage Statement

In accordance with the ICLR 2027 Policy on AI Assistance, we disclose the use of Large Language Models (LLMs) during the preparation of this work:

- **Writing and Editing:** LLMs were utilized as writing assistants to refine grammar, improve sentence flow, format LaTeX tables, and ensure stylistic consistency across the manuscript.
- **Literature Discovery:** LLMs were employed to assist in discovery and literature search for relevant baseline methodologies and related work in geometric deep learning. All retrieved citations and statements were manually verified for accuracy by the authors.
- **Code Execution & Implementation:** LLMs assisted in the implementation phase by suggesting code refactoring for PyTorch modules, generating boilerplate code for data loading, and aiding in the debugging of minor implementation details.

Author Responsibility: We emphasize that no LLMs were used to generate the core scientific contributions, mathematical proofs (including Proposition 1), or conceptual designs. All baseline comparisons, experimental evaluations, and final interpretations were exclusively conceived, executed, and verified by the authors. The authors take full responsibility for the original content, code correctness, and scientific validity of this article.

References

Akshay Agrawal, Alnur Ali, and Stephen Boyd. Minimum-distortion embedding. *Foundations and Trends in Machine Learning*, 14(3):211–378, 2021.

Sirine Ayadi, Leon Hetzel, Johanna Sommer, Fabian Theis, and Stephan Günnemann. Unified guidance for geometry-conditioned molecular generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 138891–138924, 2024.

Tian Bian, Yifan Niu, Heng Chang, Divin Yan, Junzhou Huang, Yu Rong, Tingyang Xu, Jia Li, and Hong Cheng. Hierarchical graph latent diffusion model for conditional molecule generation. In *International Conference on Information and Knowledge Management*, pp. 130–140, 2024.

Roman Bresson, Konstantinos Divriotis, Johannes F. Lutzeyer, Iakovos Evdaimon, and Michalis Vazirgiannis. GraViti: Graph-level variational autoencoders with relaxed permutation invariance, 2026. arXiv:2605.16668.

Rémy Brossard, Oriel Frigo, and David Dehaene. Graph convolutions that can finally model local structure, 2021.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning (ICML)*, 2020.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations (ICLR)*, 2016.

Gabriele Corso, Hannes Stark, Stefanie Jegelka, et al. Graph neural networks. *Nature Reviews Methods Primers*, 4(1):17, 2024.

Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds: Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991. doi: 10.1021/jm00106a046.

Peter C. Fishburn. On the sphericity and cubicity of graphs. *Journal of Combinatorial Theory, Series B*, 35:309–318, 1983.

Navid Mohammadi Foumani, Chang Wei Tan, Geoffrey I. Webb, Hamid Rezatofighi, and Mahsa Salehi. Series2vec: similarity-based self-supervised representation learning for time series classification. In *Data Mining and Knowledge Discovery (ECML PKDD)*, 2024.

Xingcheng Fu, Yisen Gao, Yuecen Wei, Qingyun Sun, Hao Peng, Jianxin Li, and Xianxian Li. Hyperbolic geometric latent diffusion model for graph generation. In *International Conference on Machine Learning (ICML)*, pp. 14102–14124, 2024.

Yisen Gao, Xingcheng Fu, Qingyun Sun, Jianxin Li, and Xianxian Li. Toward a unified geometry understanding: Riemannian diffusion framework for graph generation and prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

D. Hou et al. Dag-aware variational autoencoder for social propagation graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Yinan Huang, Xingang Peng, Jianzhu Ma, and Muhan Zhang. 3dlinker: An E(3) equivariant variational autoencoder for molecular linker design. In *International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9280–9294, 2022.

Truong Son Hy and Risi Kondor. Multiresolution equivariant graph variational autoencoder, 2021. arXvi:2106.00967.

Fergus Imrie, Anthony R. Bradley, Mihaela van der Schaar, and Charlotte M. Deane. Deep generative models for 3d linker design. *Journal of Chemical Information and Modeling*, 60(4):1983–1995, 2020.

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

Bhavesh Khemani, Sagar Patil, Ketan Kotecha, et al. A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, 11(1):18, 2024.

Shima Khoshraftar and Aijun An. A survey on graph representation learning methods. *ACM Transactions on Intelligent Systems and Technology*, 15(1), 2024.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *IEEE Transactions on Neural Networks*, 5(1):61–80, 2016.

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

Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1), 2020.

Paul Krzakala, Gabriel Melo, Charlotte Laclau, Florence d'Alché Buc, and Rémi Flamary. The quest for the GRAph Level autoEncoder (GRALE). In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning (ICML)*, 2019.

Junying Li, Deng Cai, and Xiaofei He. Learning graph-level representation for drug discovery, 2017.

Mufei Li, Viraj Shitole, Eli Chien, Changhai Man, Zhaodong Wang, Srinivas, Ying Zhang, Tushar Krishna, and Pan Li. LayerDAG: A layerwise autoregressive diffusion model for directed acyclic graph generation. In *International Conference on Learning Representations (ICLR)*, 2025.

Chengyi Liu, Wenqi Fan, Yunqing Liu, Jiatong Li, Hang Li, Hui Liu, Jiliang Tang, and Qing Li. Generative diffusion models on graphs: Methods and applications. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 6702–6711, 2023.

Xinyang Liu, Yilin He, Bo Chen, and Mingyuan Zhou. Advancing graph generation through beta diffusion. In *International Conference on Learning Representations (ICLR)*, 2025.

Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *International Conference on Knowledge Discovery & Data Mining*, pp. 723–731, 2019.

Hiroshi Maehara. On the sphericity of the graphs of semiregular polyhedra. *Discrete Mathematics*, 58:311–315, 1986.

Adam Małkowski, Jakub Grzechociński, and Paweł Wawrzyński. Regae: Graph autoencoder based on recursive neural networks. In *International Conference on Neural Information Processing*, pp. 263–274, 2023.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

Radosław Nowak, Adam Małkowski, Daniel Cieślak, Piotr Sokół, and Paweł Wawrzyński. Graph vertex embeddings: Distance, regularization and community detection. In *International Conference on Computational Science (ICCS)*, pp. 43–57, 2024.

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

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022. arXiv:2204.06125.

Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Structural, Syntactic, and Statistical Pattern Recognition*, volume 5342 of *Lecture Notes in Computer Science*, pp. 287–297. Springer, 2008.

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

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. ISSN 0925-2312.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Zichong Wang, Zhipeng Yin, and Wenbin Zhang. A unified framework for fair graph generation: Theoretical guarantees and empirical advances. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Weihuang Wen and Tianshu Yu. HyperPLR: Hypergraph generation through projection, learning, and reconstruction. In *International Conference on Learning Representations (ICLR)*, 2025.

Daniel Wesego. Graph representation learning with diffusion generative models. In *Advances in Neural Information Processing Systems*, 2025.

Robin Winter, Frank Noé, and Djork-Arné Clevert. Permutation-invariant variational autoencoder for graph-level representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 9559–9573, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Zhe Xu, Ruizhong Qiu, Yuzhong Chen, Huiyuan Chen, Xiran Fan, Menghai Pan, Zhichen Zeng, Mahashweta Das, and Hanghang Tong. Discrete-state continuous-time diffusion for graph generation. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems (NeurIPS 2024)*, pp. 79704–79740, 2024.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning (ICML)*, pp. 5694–5703, 2018.

Yuning You, Ruida Zhou, Jiwoong Park, Haotian Xu, Chao Tian, Zhangyang Wang, and Yang Shen. Latent 3d graph diffusion. In *International Conference on Learning Representations (ICLR)*, 2024.

Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning, 2019.

Cai Zhou, Xiyuan Wang, and Muhan Zhang. Unifying generation and prediction on graphs with latent graph diffusion. In *Advances in Neural Information Processing Systems*, volume 37, pp. 61963–61999, 2024.

A Proof of Proposition 1

The proof is by construction. The radius of each ball is $0.5\sqrt{2n-1}$. The center of each ball is a vector of 0s and exactly n 1s. If two nodes are adjacent, they share 1 at a single coordinate; if they are not, they do not share any 1s; only two balls may share 1 at the same coordinate. The center of the first ball is $[1, \ldots, 1, 0, \ldots, 0]$. The center of i-th ball is constructed as follows. For adjacent balls $1, \ldots, i-1$, we copy their 1s at lowest coordinates at which they do not share 1 with other balls. The remaining 1s are set at coordinates starting from $(i \cdot n + 1)$-th.

For this coordination we need at most n^2 coordinates.

If the nodes are adjacent, the distance between their corresponding ball centers is $\sqrt{2n-2}$, thus they intersect. If they are not adjacent, the distance is $\sqrt{2n}$, thus they do not intersect. □

For more results on graph sphericity, see (Fishburn, 1983; Maehara, 1986).

B Details of the GeoGAE architecture

B.1 Ordering nodes

Algorithm 1 presents details of the node ordering. Note that due to normalization of embeddings, $\bar{x} = \mathbf{0}$.

Algorithm 1: Ordering nodes

Input: Geometric embeddings $(x_1^g, \ldots, x_n^g)$ where x_j^c and r_j denote the ball center and radius of the j-th node.
$\bar{x} \leftarrow \frac{1}{n}\sum_{k=1}^{n} x_k^c$
$\pi(1) \leftarrow \arg\min_{j\in\{1,\ldots,n\}} \|x_j^c - \bar{x}\|$
$y_1^g = x_{\pi(1)}^g$
$\mathbb{A} \leftarrow \{1, \ldots, n\} \setminus \{\pi(1)\}$
for $i = 2, \ldots, n$ **do**
  $\pi(i) \leftarrow \arg\min_{j\in\mathbb{A}} \left(\frac{\|x_j^c - x_{\pi(i-1)}^c\|}{0.5(f(r_j)+f(r_{\pi(i-1)}))} + \frac{\log_2 \|x_j^c - \bar{x}\|}{10} \right)$
  $y_i^g = x_{\pi(i)}^g$
  $\mathbb{A} \leftarrow \mathbb{A} \setminus \{\pi(i)\}$
end for
return $(y_1^g, \ldots, y_n^g)$

B.2 Formal Mechanics of Node Bundling

Given an ordered sequence of node embeddings $Y = (y_1^g, \ldots, y_n^g) \in \mathbb{R}^{n\times d_{\text{geom}}}$, the bundling procedure operates as follows:

1. **Padding:** If $n \pmod b \neq 0$, $n_{\text{pad}} = b - (n \pmod b)$ copies of the final node y_n^g are appended to Y, yielding a padded sequence of length $n' = \lceil n/b \rceil \cdot b$.
2. **Reshaping & Projection:** The padded matrix is reshaped into $T = n'/b$ bundle blocks of shape $[T, b \cdot d]$ and mapped via an MLP, $\phi : \mathbb{R}^{b\cdot d} \to \mathbb{R}^{d_{\text{tok}}}$, to form the encoder sequence tokens $[t_1, \ldots, t_T]$.
3. **Decoding:** In the decoder path, each generated token is mapped via an MLP, $\psi : \mathbb{R}^{d_{\text{tok}}} \to \mathbb{R}^{b\cdot d} \times \mathbb{R}^b$, into b raw geometric embeddings alongside b stopping logits used to accurately truncate any padded nodes in the final bundle.

B.3 Masking encoder tokens

Details of masking in the encoder attention are presented in Figure 4. We make sure class tokens do not attend to each other – this results in each of them is forced to learn a meaningful information about the rest of the tokens in a different way. Intuitively, the encoder may learn to transform different class tokens into orthogonal characteristics of the graph.

B.4 Optimization objective

The total objective function optimized during training is a combination of: a geometric reconstruction loss, hyperball reconstruction loss, an end of output sequence (EOS) loss, an optional Kullback-Leibler (KL) divergence regularization, and optional node and edge features' reconstruction losses. Crucially, to mitigate exposure bias (as described in Section 3.6), the losses are accumulated over all K iterative layers of the decoder.

Let $\hat{x}_i^{(k)} = [\hat{x}_i^{c,(k)}, \hat{r}_i^{(k)}]$ denote the geometric node embedding predicted at the k-th decoder layer, split into its center and raw radius as in Section 3.3, and let

$$m_{i,j}^{(k)} = 0.75\big(f(\hat{r}_i^{(k)}) + f(\hat{r}_j^{(k)})\big) - \|\hat{x}_i^{c,(k)} - \hat{x}_j^{c,(k)}\| \tag{5}$$

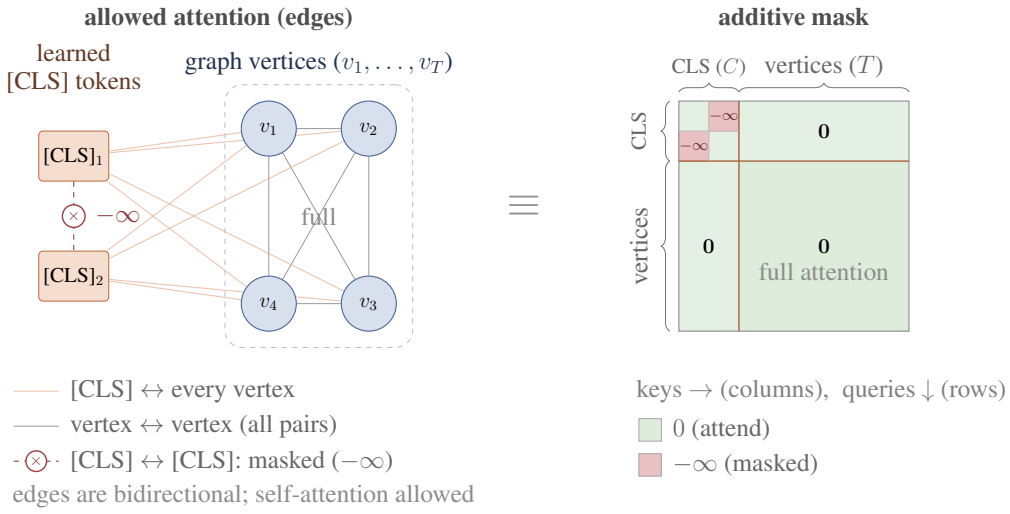

Figure 4: Encoder self-attention mask. The token sequence is $[\texttt{[CLS]}_1, \ldots, \texttt{[CLS]}_C, v_1, \ldots, v_T]$. Each `[CLS]` token attends to every graph vertex and every vertex attends back (global read/write), and vertices attend to one another over the complete set of pairs, so node–node attention is fully connected and permutation-invariant. The only forbidden interactions are between distinct `[CLS]` tokens, which are removed with an additive $-\infty$ entry; self-attention on the diagonal is kept. Left: the allowed attention edges. Right: the equivalent additive mask, whose `[CLS]` × `[CLS]` off-diagonal block is masked while all `[CLS]`–vertex and vertex–vertex blocks are zero.

be the predicted margin between nodes i and j. We treat edge existence as a binary classification problem on $p_{i,j}^{(k)} = \sigma(m_{i,j}^{(k)}/T)$ against $A_{i,j}$, $T > 0$ is constant, using the same focal-loss construction as in Eq. (1) ($\widehat{p}_{ij}^{(k)}$, α_t defined analogously, substituting $A_{i,j}$ for $D_{i,j} = 1$):

$$\mathcal{L}_{\text{geo}}^{(k)} = \frac{1}{\binom{n}{2}} \sum_{1 \le i < j \le n} -\alpha_t \, (1 - \widehat{p}_{ij}^{(k)})^{\gamma} \log(\widehat{p}_{ij}^{(k)}). \tag{6}$$

To improve the reconstruction of the original embeddings to preserve the structure of the generated embeddings, we also concurrently use the Huber loss:

$$\mathcal{L}_{\text{emb}}^{(k)} = \frac{1}{n} \sum_{i=1}^{n} H_\delta\big(\hat{x}_i^{(k)} - x_i\big), \qquad H_\delta(z) = \begin{cases} \frac{1}{2} z^2 & |z| \le \delta \\ \delta\left(|z| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{7}$$

Concurrently, the model must accurately predict the graph size. The decoder predicts stop-token logits for each generated node slot. We treat this as a binary classification problem where the ground-truth target $y_i \in \{0, 1\}$ is 1 if the slot index meets or exceeds the actual graph size, and 0 otherwise. Given the significant class imbalance (mostly non-stop tokens), we evaluate the predictions using Focal Loss. Following the standard notation, let $q_i \in [0, 1]$ be the model's estimated probability for the i-th token being a stop token. We formally define $\widehat{q}_i$ and η_i as:

$$\widehat{q}_i = \begin{cases} q_i & \text{if } y_i = 1 \\ 1 - q_i & \text{otherwise} \end{cases}, \quad \eta_i = \begin{cases} \eta & \text{if } y_i = 1 \\ 1 - \eta & \text{otherwise} \end{cases}, \tag{8}$$

where $\eta \in [0, 1]$ is a weighting factor for the positive class. The stopping loss is then concisely formulated as:

$$\mathcal{L}_{\text{stop}}^{(k)} = \frac{1}{n} \sum_{i=1}^{n} -\eta_i (1 - \widehat{q}_i)^{\gamma} \log(\widehat{q}_i), \tag{9}$$

with $\gamma \ge 0$ serving as a focusing parameter that smoothly down-weights the loss for easily classified examples.

Optionally, when the model is configured to decode node, and edge features, we use the following feature reconstruction losses:

$$\mathcal{L}_{\text{nfeat}}^{(k)} = \frac{1}{n}\sum_{i=1}^{n} H_1(\hat{c}_i^{(k)} - c_i) + \text{CE}(\hat{\bar{c}}_i^{(k)},\ \bar{c}_i) \tag{10}$$

$$\mathcal{L}_{\text{efeat}} = \frac{1}{|E|}\sum_{(i,j)\in E} H_1(\hat{e}_{ij}^{(k)} - e_{ij}) + \text{CE}(\hat{\bar{e}}_{ij}^{(k)},\ \bar{e}_{ij}) \tag{11}$$

where c_i (e_{ij}) and $\hat{c}_i^{(k)}$ ($\hat{e}_{ij}^{(k)}$) are continuous node (edge) features and their reconstructions, $\bar{c}_i$ ($\bar{e}_{ij}$) and $\hat{\bar{c}}_i^{(k)}$ ($\hat{\bar{e}}_{ij}^{(k)}$) are 1-hot encodings of discrete node (edge) features and their reconstructions, H_1 is the Huber loss (7), and CE is the standard Cross-Entropy loss.

The total loss for a single graph is given by an averaging the geometric embedding reconstruction loss $\mathcal{L}_{\text{emb}}^{(k)}$ altogether with the geometric and stopping losses across all K instances of the decoder, and adding the KL divergence $\mathcal{L}_{\text{KL}}$ between the approximate posterior and the standard Gaussian prior, weighted by a linear warmup $\beta(t)$ from 0 to a target scale over the first training epochs:

$$\mathcal{L}_{\text{total}} = \beta(t)\,\mathcal{L}_{\text{KL}} + \frac{1}{K}\sum_{k=1}^{K}\left(\mathcal{L}_{\text{emb}}^{(k)} + \rho\mathcal{L}_{\text{geo}}^{(k)} + \mathcal{L}_{\text{stop}}^{(k)}\right) + \lambda_{\text{feat}}\left(\sum_{k=1}^{K}\mathcal{L}_{\text{nfeat}}^{(k)} + \mathcal{L}_{\text{efeat}}\right). \tag{12}$$

We divide the presented losses into: KL-div loss: $\mathcal{L}_{\text{KL}}$, topological losses: $\mathcal{L}_{\text{emb}}^{(k)}$, $\mathcal{L}_{\text{geo}}^{(k)}$, $\mathcal{L}_{\text{stop}}^{(k)}$, and feature reconstruction losses: $\mathcal{L}_{\text{nfeat}}, \mathcal{L}_{\text{efeat}}$. When VAE is disabled, the loss: $\mathcal{L}_{\text{KL}}$ is not used; When features decoding is disabled, feature losses: $\mathcal{L}_{\text{nfeat}}, \mathcal{L}_{\text{efeat}}$ are also not present in the $\mathcal{L}_{\text{total}}$. ρ is a hyperparameter.

In practice, each component of the topological loss is additionally rescaled every batch to match the mean loss magnitude across components (capped) before the static weights above are applied, so that no single term dominates the gradient purely by raw scale. During batch processing, $\mathcal{L}_{\text{total}}$ is averaged across all graphs in the batch.

C Benchmark datasets

The datasets used in the experimental study are MUTAG (Debnath et al., 1991), AIDS (Riesen & Bunke, 2008), IMDB-BINARY (Kriege et al., 2020), QM9 (Ramakrishnan et al., 2014), and SYNTHETIC NEW (Morris et al., 2020), COLLAB, REDDIT-BINARY (Yanardag & Vishwanathan, 2015). They are characterized in Table 6.

D Hardware and Computational Runtime

All experiments and hyperparameter optimizations were conducted across high-performance computing infrastructure, primarily leveraging two computational allocation grants on the supercomputer equipped with NVIDIA A100 GPUs (80GB). Due to job scheduling limits and queue constraints on the primary cluster, secondary computational environments (detailed in Table 7) were utilized in parallel to distribute the workload.

Across the entire research lifecycle—encompassing preliminary model explorations, hyperparameter sweeps, baseline evaluations, and ablation studies—the total compute expenditure accumulated to approximately 32,000 GPU-hours (20,000 and 12,000 GPU-hours across the two respective allocation grants). To ensure full transparency and reproducibility, the estimated end-to-end execution times required to complete a full benchmark evaluation (averaged over 5 independent random seeds per dataset) on a single NVIDIA A100 GPU are summarized in Table 8.

E Setup of GRALE, PIGVAE, ReGAE

Authors of all three of the methods presented as baseline for GeoGAE, have shared reference implementation of their algorithms, and these implementations were utilized in calculating their scores.

Dataset	Domain and graph construction	Task	Classes / targets	Provided node/edge information
MUTAG	Molecular compounds; mutagenicity classification	Graph classification	2 classes	Nodes: 7 discrete atom types; edges: 4 bond types
AIDS	Molecular compounds; anti-HIV activity classification	Graph classification	2 classes	Nodes: discrete labels + 4 continuous attributes; edges: 4 bond types
IMDB-BINARY	Movie collaboration ego-networks. Nodes are actors; an edge connects actors appearing in the same movie. Graphs come from Action and Romance movies.	Graph classification	2 classes	No native node labels, node attributes, edge labels, or edge attributes
QM9	Small organic molecules. Nodes are atoms and edges are chemical bonds; optimized 3-D coordinates and quantum-chemical properties are supplied. Molecules contain H, C, N, O and F, with at most nine non-hydrogen atoms.	Graph-level regression	Multiple continuous targets	Atom types/features, bond types and 3-D coordinates
SYNTHETIC NEW	Artificial graphs designed for controlled graph-classification experiments. Each graph has exactly 100 vertices and approximately 196 edges.	Graph classification	2 classes	Discrete node labels and one-dimensional continuous node attributes; no edge labels
COLLAB	Scientific collaboration ego-networks. Nodes are researchers and edges represent co-authorship. Graph labels correspond to three physics research fields.	Graph classification	3 classes	No native node or edge features/labels
REDDIT-BINARY	Reddit discussion graphs. Nodes represent users; edges represent interactions/replies in a discussion thread. The two classes group discussions from different types of subreddits.	Graph classification	2 classes	No native node or edge features/labels

Table 6: Datasets

Cluster name	GPU device	GPU driver version	CPU device	Operating system	Total RAM
Cluster 1	NVIDIA A100 PCIe 80GB	565.57.01	AMD EPYC 7713 64-Core Processor	Linux-6.8.0-64-generic-x86_64-with-glibc2.39	2048 GB
Cluster 2	DGXA100 920-23687-2531-001	580.173.02	AMD EPYC 7742 64-Core Processor	Linux-5.15.0-1107-nvidia-with-glibc2.34	1024 GB
Cluster 3	AMD Radeon RX 9070 XT (Navi 48)	Mesa 26.2.2 / ROCm 7.8.0	AMD Ryzen 9 9950X3D 16-Core Processor	Linux-6.18.49-1-MANJARO-x86_64-with-glibc2.44	64 GB
Cluster 4	NVIDIA RTX PRO 500 Blackwell Generation	580.173.02	Intel(R) Core(TM) Ultra 5 235H 14-Core Processor	Linux-7.0.0-28-generic-x86_64-with-glibc2.39	30 GB

Table 7: Specification of the computational clusters used in the experiments.

Dataset	No features	With feature decoding
MUTAG	$\approx$ 24 min	$\approx$ 25 min
AIDS	$\approx$ 2h 55 min	$\approx$ 18h 4 min
IMDB-BIN	$\approx$ 1h 50 min	N/A
SYNTHETIC NEW	$\approx$ 44 min	N/A
COLLAB	$\approx$ 23h 37 min	N/A
QM9	$\approx$ 29h 16 min	$\approx$ 38h 16 min
REDDIT-BIN	$\approx$ 21h 39 min	N/A

Table 8: Runtime comparison for GeoGAE across datasets with and without feature decoding, calculated on **Cluster 1**. We report results averaged over 5 seeds. N/A denotes where computations were not applicable (datasets with no node or edge features).

Dataset	emb	hid	ppf-h	heads	layers	k-ls	p-ls	lr	w-decay
MUTAG	64	128	512	4	4	1e-3	1e-1	1e-4	0.0
AIDS	64	128	512	4	4	1e-3	1e-1	1e-4	0.0
IMBD-BIN	16	256	512	4	4	1e-2	1e-1	1e-4	1e-4
QM9	32	128	256	2	1	1e-3	5e-1	1e-4	0.0
SYNTHETIC	16	256	256	8	4	1e-2	1.0	1e-4	0.0
COLLAB	32	128	256	2	1	1e-3	5e-1	1e-4	0.0
REDDIT-BIN	32	128	256	2	1	1e-3	5e-1	1e-4	0.0

Table 9: Hyperparameters of PIGVAE: emb – graph embedding size, hid – hidden layer size, ppf-h – transformer layer size, heads – number of transformer heads, layers – number of transformer encoder/decoder layers, k-ls – kld loss scale, p-ls – permutation loss scale, lr – learning rate, w-decay – weight decay.

Dataset	emb	encod	decod	block	l-r	w-decay
MUTAG	160	2048	2048	4	3e-4	1e-3
AIDS	160	2048	2048	4	3e-4	1e-3
IMBD-BIN	160	2048	4096	4	5e-4	1e-4
QM9	160	1024	1024	32	3e-4	1e-3
SYNTHETIC	160	2048	4096	8	1e-3	1e-3
COLLAB	604	2048, 1536	4096	16	3e-4	1e-3
REDDIT-BIN	1720	4096	6144	64	1e-4	1e-4

Table 10: Hyperparameters of ReGAE: emb – embedding size, encod – sizes of encoder hidden layers, decod – sizes of decoder hidden layers, block – block size, l-r – learning rate, w-decay – weight decay of optimizer. For all experiments we set 0.5 as the mask weight and 0.2 as the embedding norm weight for the loss calculation. We use ELU (Clevert et al., 2016) as the activation function.

For all four of the evaluated algorithms, the dataset reading, splitting, training loop, metrics calculation and reporting code was common to ensure fair reporting. The hyperparameters reported in Tables 9, 10 and 11 were determined using automatic parameter hypertuning algorithm, separately for each algorithm and dataset, only exceptions being REDDIT-BIN and COLLAB for which, due to limited computational capacity, parameters were taken from the original papers or tuned manually. The best results for PIGVAE and GRALE are obtained for relatively small graph embedding sizes. Surprisingly, for larger embedding sizes, the scores for these architectures are worse.

F Ablation study

In this section, we conduct a comprehensive ablation study to evaluate the impact of key architectural choices and training strategies on GeoGAE's ability to reconstruct graph topologies. We evaluate these variations on three datasets containing both node and edge features: MUTAG, AIDS, and QM9. The quantitative results of these experiments are summarized in Table 12 and visually supported by Figure 5, and Figure 6.

Multi-layer Teacher Forcing and Exposure Bias. Autoregressive Transformer decoders are notoriously susceptible to exposure bias during inference, as they must rely on their own potentially flawed predictions instead of ground-truth tokens. We hypothesized that our multi-layer teacher forcing strategy (stacking the decoder K times) mitigates this issue. To isolate its effect, we evaluated the model using only a single decoder iteration ($K = 1$) and compared it against our default setting ($K = 3$). As shown in Table 12, reducing the iterations to one causes a drastic collapse in performance. For instance, the topological F1 score on the AIDS dataset drops from 84.12% to 72.59%. More importantly, the model completely loses its ability to correctly predict the target graph size, with the average size difference skyrocketing from 0.23 to 4.67. Figure 5 further illustrates this phenomenon across different values of K on the AIDS dataset. The reconstruction performance and size prediction stabilize strictly at $K = 2$. This proves that a small number of iterative refinement steps is both crucial and sufficient to eliminate cascading errors during graph generation.

Dataset	Layers	Heads	Node Dim	Edge Dim	Latent	Matcher	LR	Dropout	Batch
MUTAG	3	8	64×64	null×64	256	Soft	1e-4	0.0	64
AIDS	7	8	128×128	128×128	128	Soft	2e-5	0.1	8
IMDB-BIN	3	4	64×64	32×32	64	Sink	2e-5	0.0	4
QM9	6	8	64×128	64×64	128	Sink	1e-4	0.0	256
SYNTH. NEW	5	8	64×64	32×32	64	Sink	1e-4	0.0	8
COLLAB	3	4	64×128	64×64	64	Sink	1e-4	0.0	1
REDDIT-BIN	2	2	64×128	64×64	128	Sink	1e-4	0.0	1

Table 11: Hyperparameters of GRALE: Layers – number of Evoformer layers, Heads – attention heads, Node/Edge Dim – hidden × model dimension, Latent – graph embedding dimension (d_g), Matcher – node matching operator (Sink: Sinkhorn, Soft: SoftSort), LR – learning rate, Dropout – attention/MLP dropout, Batch – batch size.

Impact of Variational Regularization (VAE). Next, we investigated the effect of enabling the full Variational Autoencoder (VAE) formulation. Adding the Kullback-Leibler (KL) divergence loss forces the latent space to conform to a standard Gaussian prior, which is typically desired for generative sampling.However, our results indicate that this regularization slightly degrades the exact topology reconstruction capability. Across all three tested datasets, the purely deterministic model outperforms the VAE variant in reconstruction accuracy. For example, on the QM9 dataset, the topological F1 score decreases from 99.38% to 95.88% when VAE is enabled. This highlights a natural trade-off in graph autoencoders between achieving exact deterministic reconstruction (autoencoding) and maintaining a smooth, sampleable latent space (generation).

Influence of Node and Edge Features on Topology. Also, we examined whether encoding node and edge attributes helps the model comprehend the underlying graph structure better, even when the decoder is not explicitly tasked with reconstructing these features. In the "with features, w/o feature decoding" configuration, the encoder processes the graph's attributes, but the feature reconstruction losses ($\mathcal{L}_{nfeat}$, $\mathcal{L}_{cfeat}$) are disabled. Interestingly, we observe a consistent improvement in topological F1 scores across the datasets when side information is provided (e.g., MUTAG improves from 61.29% to 63.61%, and AIDS from 84.12% to 86.00%). This suggests that chemical or structural attributes act as strong auxiliary signals. They implicitly regularize the topological representation, helping the encoder to construct a more robust geometric embedding even for purely structural tasks.

Impact of Bundle Size. We evaluate the impact of bundle size $b \in \{1, 2, 4, 8\}$ on graph reconstruction quality (Figure 6). Conceptually, increasing b reduces sequence length $T = \lceil n/b \rceil$, which mitigates exposure bias during autoregressive decoding. However, packing multiple node embeddings into a single token creates an information bottleneck. Empirically, the topological F1 score degrades monotonically as b increases, dropping from 84.1% at $b = 1$ to 79.5% at $b = 8$. In contrast, the graph size prediction error reaches its minimum at $b = 2$ (0.09 vs. 0.23 at $b = 1$). Because the primary objective of GeoGAE is high-fidelity topological reconstruction, and an average size error of 0.23 nodes is practically negligible, we select $b = 1$ as our default configuration. Larger bundle sizes ($b > 1$) are reserved as a trade-off mechanism when scaling to very large graphs.

Configuration	Dataset	F1 [%]	Avg. size diff
Baseline results	MUTAG	61.29 ± 5.39	0.12 ± 0.08
	AIDS	84.12 ± 1.67	0.23 ± 0.09
	QM9	99.38 ± 0.18	0.00 ± 0.00
Using VAE	MUTAG	60.78 ± 2.01	0.24 ± 0.12
	AIDS	82.17 ± 1.98	0.61 ± 0.16
	QM9	95.88 ± 4.76	0.00 ± 0.00
Using only one stacked decoder	MUTAG	51.72 ± 2.91	4.76 ± 1.69
	AIDS	72.59 ± 2.75	4.67 ± 0.59
	QM9	98.93 ± 0.17	0.00 ± 0.00
With encoded features, without decoding them	MUTAG	63.61 ± 2.85	0.09 ± 0.09
	AIDS	86.00 ± 3.63	0.31 ± 0.12
	QM9	98.89 ± 0.32	0.00 ± 0.00

Table 12: The table presents results of ablation study experiments in comparison with the base GeoGAE results.

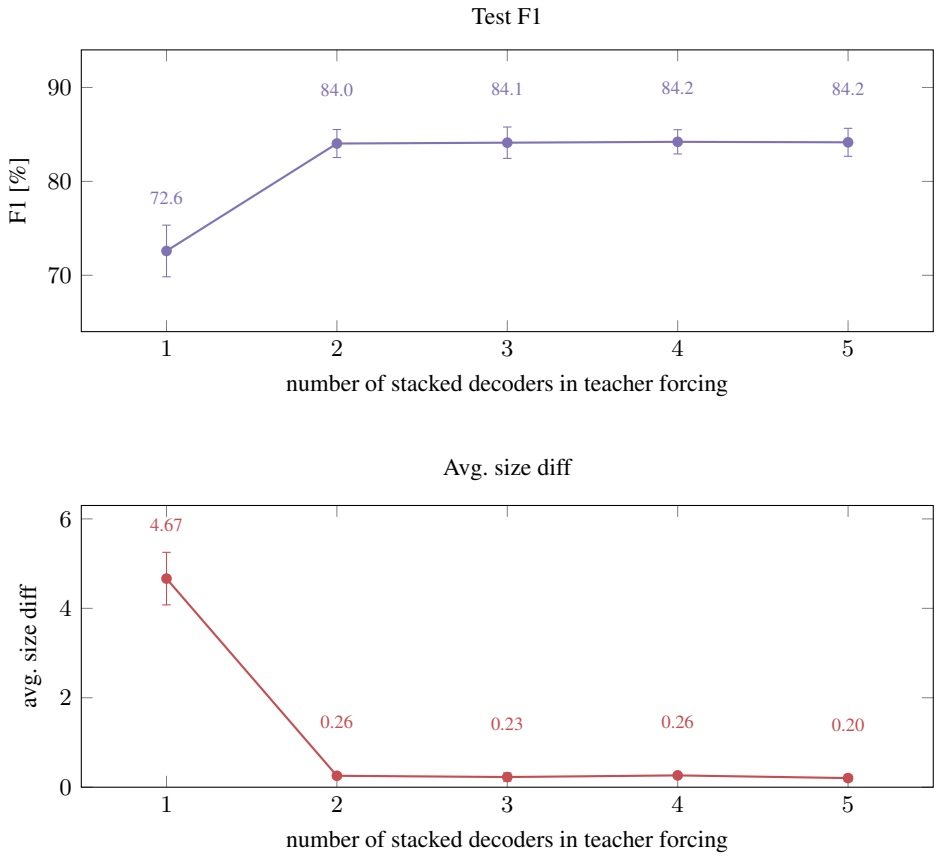

Figure 5: Effect of the number of stacked decoders in teacher forcing on AIDS test performance: F1 score (left) and average node-count (size) difference (right). Error bars show $\pm$ 1 standard deviation over 5 seeds.

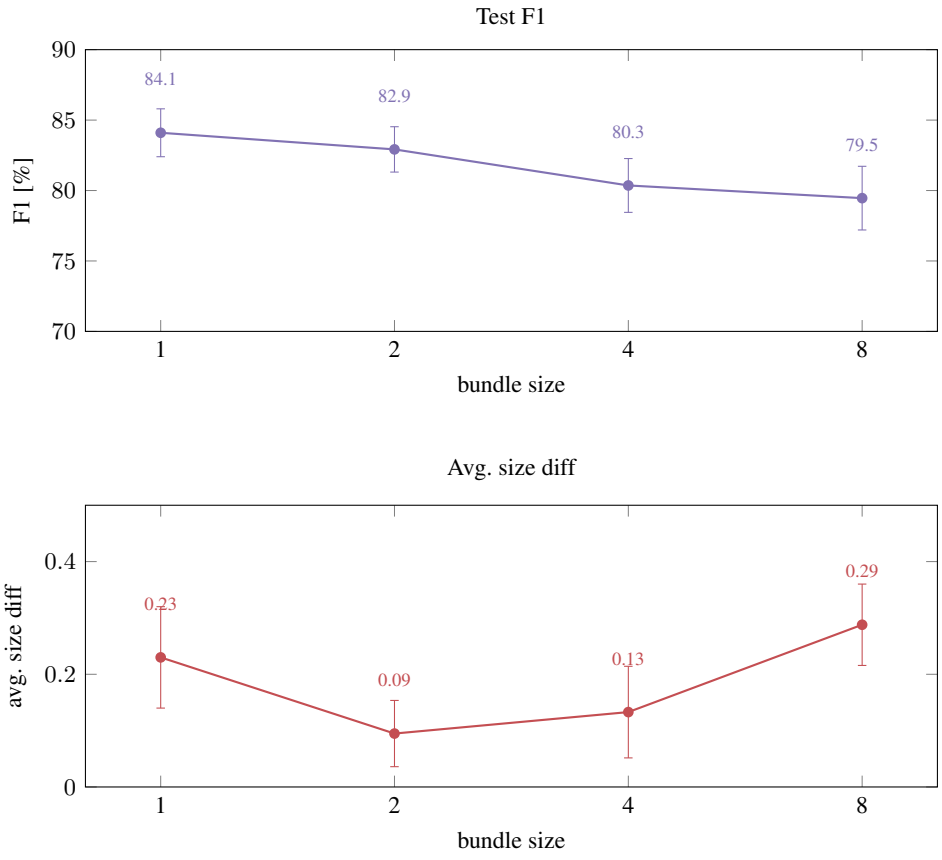

Figure 6: Effect of bundle size on test performance on AIDS dataset: F1 score (left) and average node-count (size) difference (right). Error bars show $\pm$ 1 standard deviation over 5 seeds.