# HAGGLE: Get a better deal using a Hierarchical Autoencoder for Graph Generation and Latent-space Expressivity

**Audun Myers**
Pacific Northwest National Laboratory
audun.myers@pnnl.gov

Stephen J. Young
Pacific Northwest National Laboratory

Tegan Emerson
Pacific Northwest National Laboratory

## Abstract

Generating realistic and diverse graph structures is a challenge with broad applications across various scientific and engineering disciplines. A common approach involves learning a compressed latent space where graphs are represented by a collection of node-level embeddings, often via methods such as a Graph Autoencoder (GAE). A fundamental challenge arises when we try to generate new graphs by sampling from this space. While many deep learning methods like Diffusion, Variational Autoencoders (VAEs), and Generative Adversarial Networks (GANs) can successfully generate new points in the latent space, they fail to capture the inherent relational dependencies between the node embeddings. This leads to decoded graphs that lack structural coherence and fail to replicate essential real-world properties. Alternatively, generating a single graph-level embedding and then decoding it to new node embeddings is also fundamentally limited, as pooling methods needed to create the graph level embedding are inherently lossy and discard crucial local structural information. We present a three-stage hierarchical framework called Hierarchical Autoencoder for Graph Generation and Latent-space Expressivity (HAGGLE) that addresses these limitations through systematic bridging of node-level representations with graph-level generation. The framework trains a Graph Autoencoder for node embeddings, employs a Pooling Autoencoder for graph-level compression, and utilizes a size-conditioned GAN for new graph generation. This approach generates structurally coherent graphs while providing useful graph-level embeddings for downstream tasks.

## 1 Introduction

The generation of realistic and diverse graph structures is a challenge in modern machine learning, with implications across fields from computational biology to social network analysis [4, 18, 20]. The ability to generate graphs that adhere to the statistical and structural properties of real-world networks is important for tasks like molecular design, dataset augmentation, and system simulation [2, 6, 11]. While early statistical models like Erdős-Rényi and Barabási-Albert fail to capture complex dependencies and community structures, more sophisticated statistical models such as Stochastic Block Models and Exponential Random Graph Models emerged to better address these limitations. However, more recently, deep learning has led to a paradigm shift, with generative models learning to map from compressed, continuous latent spaces to discrete graph spaces [18].

The dominant strategy for creating this latent representation is to use Graph Neural Networks (GNNs) as an encoder [17, 5]. Unlike traditional neural networks, GNNs are designed to operate on graph-structured data by a process of "message passing," where nodes aggregate information from their neighbors. This allows GNNs to effectively encode a graph's topology and node features into a continuous vector space, forming the foundation of a Graph Autoencoder (GAE) framework [12, 3]. The geometry of this latent space is a critical design choice, as it dictates how structural properties are preserved. For instance, the assumption that structurally similar nodes should be closer in Euclidean space is a common principle that influences the properties of the generated networks [21]. However, a limitation persists in many current methods, particularly those that adapt traditional generative models like Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) to the graph domain. While these models have been successfully applied to learn a continuous latent space for graphs, they often treat the node embeddings as independent points within this space [16, 13]. Crucially, node-level generative models represent essential building blocks for graph generation – they provide the fundamental machinery for creating realistic individual node representations that capture local structural properties and semantic information. The power of these models lies in their ability to learn rich, expressive distributions over node embeddings that encode both topological and feature-based similarities. However, the critical challenge lies not in the individual node-level generation itself, but in how to properly combine GAEs with these node-level generative models to sample coherent collections of node embeddings that maintain graph-level structure.

Models like GraphVAE [13] and its variants focus on learning a simple prior over the node embeddings, such as a Gaussian distribution, and then sample from this distribution to generate new graphs [13, 6]. While the individual node embeddings may be realistic, when sampled independently, the models fail to capture the crucial relational dependencies that define a coherent graph structure. Consequently, when new graphs are generated by sampling from this latent space, the decoded graphs often lack structural integrity, exhibiting disconnected components or illogical connections, and fail to replicate essential real-world properties, as the models focus on the embeddings themselves rather than the relationships between them [16, 21]. This problem has led to alternative approaches, such as those that work with discrete latent spaces to better preserve combinatorial properties [10] or use subtree-centric methods to avoid information loss from global compression [1].

An alternative approach, which attempts to summarize an entire graph into a single graph-level embedding, is also fundamentally flawed for generative tasks. While methods that rely on global pooling operations, such as summing or averaging node embeddings, are effective for graph classification and regression, they inevitably discard crucial local structural information [8]. The decoder lacks the necessary fine-grained detail to reconstruct a diverse range of graph topologies [9], which has led to efforts to enhance the latent space with subgraph information [14]. The challenge thus lies in finding a middle ground: a latent representation that is structured enough to capture relational dependencies but flexible enough to enable diverse graph generation.

To overcome these challenges, we present a three-stage hierarchical framework that systematically addresses the limitations in current graph generation methods which we term as the Hierarchical Autoencoder for Graph Generation and Latent-space Expressivity or HAGGLE.The framework consists of: (1) a Graph Autoencoder that learns rich node-level embeddings, (2) a Pooling Autoencoder that compresses node embeddings into graph-level representations while preserving structural information, and (3) a size-conditioned GAN that operates in the learned graph-level embedding space. This approach bridges node-level representations with graph-level generation, enabling the creation of structurally coherent graphs while maintaining useful graph-level embeddings for downstream tasks.

## 2 Background

Graph Autoencoders (GAEs) represent a fundamental approach to learning continuous representations of graph-structured data. A GAE consists of an encoder $E_\theta : \mathcal{G} \to \mathbb{R}^{n\times d}$ that maps a graph $G = (V, E)$ with $n$ nodes to a matrix of node embeddings $Z \in \mathbb{R}^{n\times d}$, and a decoder $D_\phi : \mathbb{R}^{n\times d} \to \mathcal{G}$ that reconstructs the graph structure. The encoder typically employs Graph Neural Networks (GNNs) that leverage message passing to aggregate neighborhood information:

$$z_i^{(l+1)} = \text{Update}^{(l)}\left(z_i^{(l)}, \text{Aggregate}^{(l)}\left(\{z_j^{(l)} : j \in \mathcal{N}(i)\}\right)\right) \tag{1}$$

where $\mathcal{N}(i)$ denotes the neighbors of node $i$, and $l$ indicates the layer index. The resulting latent space $\mathcal{Z} = \{z_1, z_2, \ldots, z_n\}$ is designed to preserve both local neighborhood structures and global graph properties.

Existing approaches to graph generation can be broadly categorized. Some methods, such as Variational Graph Autoencoders (VGAEs), are truly Graph Autoencoder (GAE) approaches that adapt standard generative models to operate on node embeddings. Similarly, Generative Adversarial Networks (GANs) and some Diffusion Models can also be applied to learn distributions over node embeddings. However, most of these methods treat node embeddings as independent samples, ignoring the relational structure that defines graph connectivity and focusing on marginal distributions rather than the joint distribution that encodes structural relationships. An alternative class of methods operates directly on the graph structure itself, including Sequential Graph Generation Models such as GraphRNN [19] and GNN-Based Diffusion Models [7, 15]. Another distinct approach involves learning a single graph-level embedding through pooling operations $h_G = \text{POOL}(\{z_1, z_2, \ldots, z_n\})$. While this enables the use of standard generative models directly, it suffers from significant information loss as the pooling operation necessarily discards fine-grained structural details, making it impossible to reconstruct diverse graph topologies accurately. The fundamental trade-off between compression and information preservation remains a critical challenge that our hierarchical framework addresses.

## 3 HAGGLE

Given a graph $G = (V, E)$ and its GAE-generated node embeddings $Z = \{z_1, z_2, \ldots, z_n\}$, traditional generative approaches sample new embeddings $\tilde{Z} = \{\tilde{z}_1, \tilde{z}_2, \ldots, \tilde{z}_m\}$ from a learned distribution $p(z)$ without considering the relational constraints that govern valid graph structures. We will use $\mathcal{R}(Z)$ as a short-hand for the intricate relationship between embeddings that encode the graphs's connectivity, community, structure, and other properties which are not local to nodes.[1] The fundamental problem is that independent sampling from $p(z)$ produces embeddings $\tilde{Z}$ where $\mathcal{R}(\tilde{Z}) \not\approx \mathcal{R}(Z)$. This leads to decoded graphs $\tilde{G}$ that lack structural coherence—meaning they fail to adhere to the graph's original and valid structural rules, resulting in connectivity loss, property violations, and local structure degradation.

Our Hierarchical Autoencoder for Graph Generation and Latent-space Expressivity (HAGGLE) framework addresses this through a novel three-stage hierarchical approach that systematically bridges node-level embeddings with graph-level generation. Rather than viewing graph generation as either single-step graph-level generation or independent node sampling, we propose a hierarchical compression-decompression paradigm that preserves GAE-generated structural information while enabling learnable graph-level representations through pooling autoencoders. Our framework integrates three sequential stages.

**Stage 1: Graph Autoencoder** We employ a standard GAE architecture with encoder $E_\theta$ and decoder $D_\phi$. The encoder uses Graph Convolutional Networks (GCNs) to generate node embeddings:

$$Z = E_\theta(X, A) = \text{GCN}(X, A) \tag{2}$$

where $X$ represents node features and $A$ is the adjacency matrix. The decoder reconstructs the adjacency matrix through:

$$\hat{A} = D_\phi(Z) = \sigma(ZZ^T) \tag{3}$$

This process is illustrated in Fig. 1. Components shown in color are reused during the generation stage (Stage 3), while grey components are used only during training to learn the node-level latent representation.

Node-level generative models form the foundation of this approach, providing the capability to generate realistic individual node embeddings that capture local structural patterns, semantic relationships, and distributional properties. However, individual node-level models require systematic integration to ensure that collections of generated embeddings maintain the relational structure necessary for coherent graph reconstruction.

**Stage 2: Pooling Autoencoder** This stage introduces a specialized Pooling Autoencoder that learns to compress collections of node embeddings into unified graph-level representations. The

[1] In practice we measure $\mathcal{R}(Z)$ using graph-level embeddings as well as some standard graph structure statistics which are described in Section 4.3.

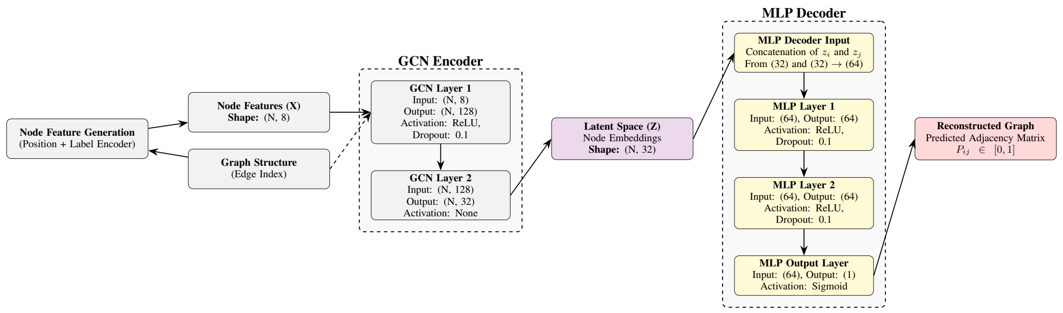


Figure 1: Stage 1: Graph Autoencoder (GAE) architecture. The Violet latent space Z is passed as input to Stage 2 (Fig. 2).

pooling encoder $P_\alpha$ aggregates node embeddings using sophisticated pooling mechanisms:

$$g = P_\alpha(Z) = \text{Pool}(f_\alpha(Z)) \in \mathbb{R}^{d_g} \tag{4}$$

where $f_\alpha$ represents learnable transformation layers and Pool$(\cdot)$ denotes the pooling operation. Our framework uses attention-based pooling ($\text{Pool}(Z) = \sum_{i=1}^{n} \alpha_i z_i$ where $\alpha_i = \text{softmax}(W_a \tanh(W_z z_i))$). The subscripts on the learnable weight matrices $W_a$ and $W_z$ denote their specific function in the attention mechanism, with $W_z$ transforming the input node embedding and $W_a$ projecting the result to a scalar attention score. The pooling decoder $Q_\beta$ reconstructs the original node embedding collection:

$$\hat{Z} = Q_\beta(g, n) \in \mathbb{R}^{n \times d_z} \tag{5}$$

This stage is trained with reconstruction loss $\mathcal{L}_{\text{pool}} = \|Z - \hat{Z}\|_F^2$, learning to preserve essential structural information in the compressed graph representation. The process is illustrated in Fig. 2, where colored components indicate elements used in generation while grey components are training-only.

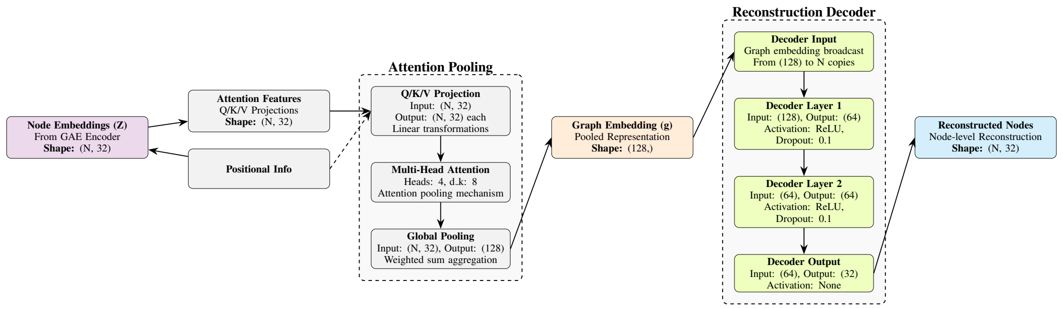


Figure 2: Pooling Autoencoder architecture for graph-level compression. The input Violet node embeddings Z are the latent space output from Fig. 1.

**Stage 3: Graph-Level Generation** This stage employs a size-conditioned Generative Adversarial Network (GAN) operating in the learned graph-level embedding space. We chose GAN over alternative generative models because VAE-based approaches enforce Gaussian priors through KL-divergence regularization that smooth over the discrete structural patterns necessary for preserving graph properties, while diffusion models require end-to-end training that prevents leveraging our pre-trained GAE and pooling representations. In contrast, the GAN operates on the learned embedding space without strict distributional constraints, allowing it to capture the complex, multimodal manifold of valid graph structures while building upon already-established encodings—as evidenced by substantially better distributional overlap in both node- and graph-level embedding spaces (see Appendix Figure 8). The generative model $G_\gamma$ learns the distribution of graph embeddings conditioned on graph size:

$$g_{\text{new}} \sim G_\gamma(\cdot \mid n) \in \mathbb{R}^{d_g} \tag{6}$$

The complete forward process for generating a new graph $\tilde{A}$ is:

$$\tilde{A} = D_\phi(Q_\beta(g_{new}, n)), \tag{7}$$

where $Q_\beta$ reconstructs node-level embeddings $\tilde{Z} = Q_\beta(g_{new}, n)$ from the generated graph embedding $g_{new}$, and $D_\phi$ produces the adjacency matrix $\tilde{A}$ from pairwise embedding interactions.

The complete generation pipeline is illustrated in Fig. 3, which shows how the three trained stages work together during inference. Given a target graph size $n$, the GAN generates a new graph-level embedding $g_{\text{new}}$ in the learned latent space. The pooling decoder from Stage 2 then expands this compressed representation into a collection of node embeddings $\tilde{Z}$, maintaining the relational structure encoded during training. Finally, the GAE decoder from Stage 1 reconstructs the adjacency matrix $\tilde{A}$ from these node embeddings through pairwise interactions. This hierarchical approach ensures that generated graphs preserve both local node-level patterns and global structural properties by leveraging the learned representations from both autoencoders.

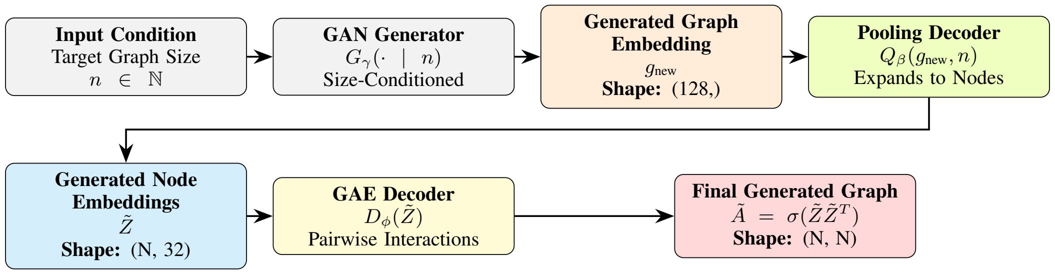


Figure 3: High-level overview of the HAGGLE generation process (Stage 3). **Color coding:** Colored components correspond to trained models from previous stages—Orange graph embedding and Lime pooling decoder from Fig. 2, Yellow GAE decoder and Red output from Fig. 1, and Cyan intermediate node embeddings. Grey components are unique to the generation stage.

## 4 Experimental Design and Results

### 4.1 Datasets

We evaluate our framework on three classes of synthetic graphs: Stochastic Block Models (SBMs), Random Trees, and Disjoint Unions of Cycles (DUCs). Each dataset contains 10,000 graphs with variable node counts to assess structural coherence across scales. SBMs test community structure preservation, Random Trees challenge hierarchical structure maintenance, and DUCs test local cyclical patterns and global disconnectedness. Detailed generation parameters are provided in the Appendix. Results focus primarily on SBMs with summary statistics across all datasets.

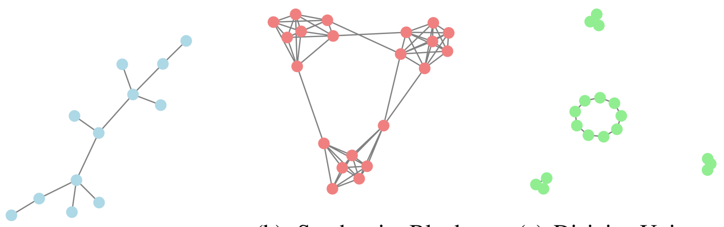

(a) Random Trees (b) Stochastic Block Model (c) Disjoint Union of Cycles

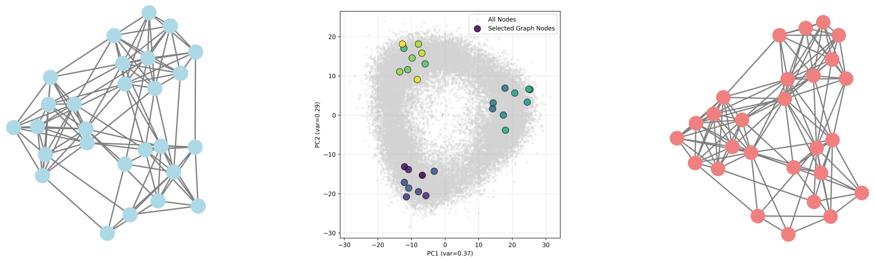
Figure 4: Example graphs from each synthetic dataset used for evaluation. (a) Random trees exhibit hierarchical structure without cycles. (b) Stochastic block models contain community structure with dense intra-community and sparse inter-community connections. (c) Disjoint union of cycles consist of multiple disconnected cyclic components.

### 4.2 Benchmarking

#### 4.2.1 Graph Autoencoder

The Graph Autoencoder serves as the foundation of our hierarchical framework, establishing node-level representations for all subsequent generation methods. Our GAE baseline achieves high-fidelity reconstruction with greater than 95% edge prediction accuracy across all datasets, successfully capturing local neighborhood patterns and global structural properties as evidenced by clear clustering in PCA projections.

Figure 5 demonstrates the GAE's structured representations. The PCA visualization shows that structurally similar nodes cluster together while maintaining sufficient separation for accurate reconstruction. The embedding space exhibits structural preservation, smooth interpolation capabilities, and dimensionality efficiency with 32-dimensional embeddings that remain computationally tractable. We also see in Figure 5 that the foundational GAE's reconstruction capabilities and embedding space structure. The visualization shows how the GAE successfully encodes graph topology into a continuous latent space while maintaining high-fidelity reconstruction, establishing the quality of node-level representations used by subsequent methods.



(a) Original Graph (b) PCA Embeddings (c) Reconstructed Graph

Figure 5: Graph Autoencoder performance demonstration showing: (a) an original graph from the testing dataset, (b) the learned node embeddings visualized via PCA projection into 2D space, and (c) the decoded graph.

### 4.2.2 Independent Latent Space Sampling

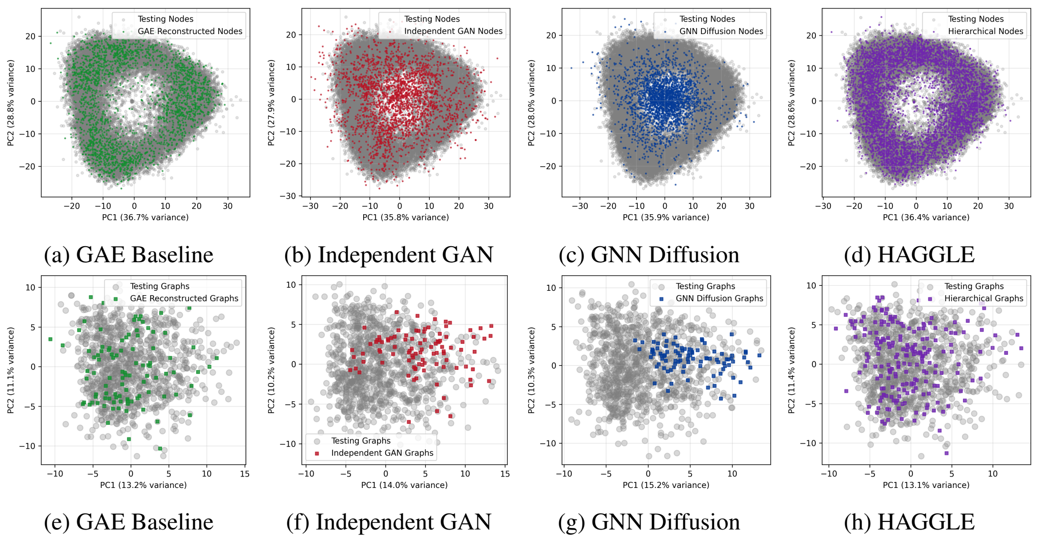
Independent sampling approaches represent traditional GAE-based methods that sample node embeddings independently from learned distributions (Gaussian VAEs, GAN-based generators, diffusion models). These methods serve as our primary comparison to highlight structural coherence improvements achieved by our hierarchical framework.

Independent sampling methods exhibit systematic failures in maintaining structural coherence, frequently producing disconnected graphs and deviating significantly from training distributions. This indicates fundamental failure to capture relational dependencies, as independent sampling cannot maintain correlations between adjacent nodes' embeddings. As shown in Figure 6, the Independent GAN method demonstrates poor distributional preservation at both node and graph levels.

Additional comparisons with other generative methods showing direct embedding generation are provided in Appendix Figure 8. Note that all embeddings shown in Figure 6 represent the complete forward process: generated graphs are first decoded to adjacency matrices, then re-encoded through the GAE to obtain node embeddings, and finally processed through the pooling autoencoder to obtain graph-level embeddings. This provides a more realistic assessment of end-to-end generation quality compared to direct embedding generation.

### 4.2.3 Graph Diffusion Neural Network

GNN-based diffusion models represent a state-of-the-art approach for generating graphs with strong structural coherence. These methods apply diffusion processes directly to graph structures and use Graph Neural Networks (GNNs) to preserve structural dependencies. In our experiments, we use a discrete denoising diffusion model for graph generation, a specific implementation known as **DiGress** [15]. While these models have shown impressive performance on molecular datasets, their need for end-to-end training limits their ability to leverage the pre-trained GAE representations foundational to our framework. As shown in Table 1 and Figure 6, this end-to-end training paradigm resulted in a significant loss of structural fidelity on our synthetic datasets, with poor embedding distribution preservation and a high average JS Divergence of 0.280. This suggests that while DiGress is highly effective at learning to generate complex, real-world graph distributions, it struggled to maintain the fine-grained structural properties of our specific synthetic datasets when compared to our hierarchical approach.



(a) GAE Baseline (b) Independent GAN (c) GNN Diffusion (d) HAGGLE

(e) GAE Baseline (f) Independent GAN (g) GNN Diffusion (h) HAGGLE

Figure 6: Comprehensive PCA comparison across all methods showing both node-level and graph-level embedding distributions obtained through the complete forward process. **Top row:** Node-level PCA projections comparing original (grey) vs. generated node embeddings, where generated embeddings are obtained by re-encoding the generated graphs through the GAE. **Bottom row:** Graph-level PCA projections comparing original (grey) vs. generated graph-level embeddings, where generated embeddings are obtained by processing the GAE node embeddings through the pooling autoencoder.

## 4.3 Comparative Performance Metrics

We employ a comprehensive evaluation framework that assesses graph generation quality across two complementary dimensions: embedding space fidelity and structural property preservation. This dual approach enables both direct evaluation of our hierarchical representation learning and assessment of the final generated graph quality.

**Graph-Level Embedding Metrics** These metrics evaluate how well our framework preserves the distributional properties of the learned graph-level embeddings. Wasserstein Distance measures the optimal transport cost between true and generated graph embedding distributions, providing a measure of distributional similarity. Maximum Mean Discrepancy (MMD) computes distributional discrepancy using kernel methods, capturing differences in statistical moments of the distributions.

**Graph Structure Metrics** These metrics evaluate the structural properties of the final generated graphs using Jensen-Shannon (JS) divergence to measure distributional similarity between generated and training graphs. All structural metrics range from 0 (identical distributions) to 1 (completely different distributions), with lower values indicating better preservation of structural properties. Degree Distribution JS Divergence measures the similarity between degree distributions of generated and training graphs, assessing preservation of connectivity patterns and network topology. Clustering Coefficient JS Divergence evaluates the preservation of local clustering patterns by comparing the distributions of node clustering coefficients, measuring how well the generated graphs maintain local community structure and transitivity properties. Path Length JS Divergence compares shortest path length distributions to assess global connectivity patterns and the preservation of graph diameter and efficiency properties, computed on a subset of graphs for computational efficiency.

## 4.4 Results and Findings

Our hierarchical framework, HAGGLE, performs effective generation through the three-stage pipeline: graph-level embedding generation, node-level embedding reconstruction, and final graph structure recovery. A detailed end-to-end generation example is provided in Appendix Figure 7. The comprehensive comparison in Figure 6 shows that our pooling autoencoder effectively preserves embedding distributions at both hierarchical levels with substantial overlap between original and generated distributions.

HAGGLE demonstrates superior performance across both embedding and structural metrics compared to baseline approaches, as clearly illustrated in Figure 6. Independent Sampling consistently performs poorly across all datasets, confirming fundamental limitations of independent node embedding generation. GNN Diffusion shows mixed results with significant failures on clustering coefficient preservation. The visual comparison shows that our method achieves the best distributional overlap between original and generated embeddings at both node and graph levels. The results show that by explicitly modeling the relationships between node embeddings through our pooling autoencoder, we achieve significantly better preservation of both local and global graph properties while maintaining high-quality embedding space representations.

| **Dataset** | **Metric** | **GAE Baseline** | **Independent Sampling** | **GNN Diffusion** | **HAGGLE** |
|---|---|---|---|---|---|
| **SBMs** | *Embedding Space Metrics* | | | | |
| | Wasserstein Distance | 0.7828 | 7.7295 | 3.3175 | **1.3826** |
| | Maximum Mean Discrepancy | 0.0186 | 0.0298 | 0.0172 | **0.0077** |
| | *Graph Structure Metrics (JS Divergence)* | | | | |
| | Degree Distribution | 0.1181 | 0.2103 | 0.07426 | **0.06863** |
| | Clustering Coefficient | 0.2992 | 0.2304 | 0.64067 | **0.06151** |
| | Path Length Distribution | 0.0573 | 0.0917 | 0.13165 | **0.01523** |
| | *Computation Time* | | | | |
| | Training (seconds) | 1118.2 | 1406.7 | 3474.4 | 1974.9 |
| | Running (seconds/graph) | 0.0018 | 0.0021 | 2.8911 | 0.0243 |
| **Trees** | *Embedding Space Metrics* | | | | |
| | Wasserstein Distance | 0.012681 | 0.132516 | **0.06416** | 0.258494 |
| | Maximum Mean Discrepancy | 0.003438 | 0.047027 | **0.031715** | 0.081483 |
| | *Graph Structure Metrics (JS Divergence)* | | | | |
| | Degree Distribution | 0.0444 | 0.3717 | 0.3305 | **0.1795** |
| | Clustering Coefficient | 0.1318 | 0.7900 | 0.1505 | **0.0000** |
| | Path Length Distribution | 0.0761 | 0.3064 | 0.3161 | **0.1828** |
| | *Computation Time* | | | | |
| | Training (seconds) | 488.1 | 562.0 | 1510.0 | 701.9 |
| | Running (seconds/graph) | 0.0011 | 0.0016 | 1.92 | 0.0194 |
| **Cycles** | *Embedding Space Metrics* | | | | |
| | Wasserstein Distance | 0.011934 | 0.030425 | 0.098906 | **0.021444** |
| | Maximum Mean Discrepancy | 0.045811 | 0.141704 | 1.176883 | **0.085173** |
| | *Graph Structure Metrics (JS Divergence)* | | | | |
| | Degree Distribution | 0.0502 | 0.3813 | 0.3665 | **0.1734** |
| | Clustering Coefficient | 0.0449 | 0.3216 | 0.2115 | **0.0631** |
| | Path Length Distribution | 0.0975 | 0.2006 | **0.1034** | 0.1663 |
| | *Computation Time* | | | | |
| | Training (seconds) | 465.6 | 558.9 | 1423.3 | 684.4 |
| | Running (seconds/graph) | 0.0010 | 0.0016 | 2.14 | 0.0199 |

Table 1: Comprehensive comparison of graph generation methods across all three datasets. Results show embedding fidelity and structural preservation metrics. Lower values indicate better performance for all metrics. In the are most performative generative model not including the GAE baseline as is is not generative.

The experimental results across all three datasets reveal several important insights about HAGGLE, as shown in Table 1. We note that the GAE baseline is not considered in the comparison as it isn't itself a generative model, but we provide its performance in reconstructing the testing graphs as a point of comparison. HAGGLE demonstrates strong performance across different graph types. On SBMs, it achieves the best performance across all structural and embedding metrics, indicating excellent community structure preservation. On Trees, it achieves perfect clustering coefficient preservation, effectively capturing hierarchical structure. On Cycles, it achieves the best degree distribution preservation, demonstrating good cyclical pattern handling.

Our hierarchical approach maintains competitive embedding space quality while achieving superior structural preservation. The framework offers practical computational efficiency [2] with training time (1974.9 seconds) significantly lower than GNN Diffusion (3474.4 seconds). The consistent strong performance across three structurally distinct datasets demonstrates the framework's generalizability and ability to adapt to different graph topologies while maintaining structural coherence.

## 5 Conclusion and Future Directions

This work addresses a fundamental limitation in graph generation: traditional generative models fail to maintain relational dependencies between node embeddings, resulting in structurally incoherent graphs despite realistic individual embeddings. The HAGGLE framework provides a systematic solution through a specialized Pooling Autoencoder that bridges node-level and graph-level representations. This enables generative models to operate in compressed graph embedding spaces while preserving structural relationships necessary for coherent reconstruction. Experimental evaluation demonstrates consistent superior performance across structural metrics and embedding fidelity measures. HAGGLE achieves significant improvements over baseline approaches; particularly for JS divergence scores, degree distribution, and path length distribution. The modular architecture allows flexible integration of different generative models while maintaining computational efficiency.

Extensions to real-world datasets, attributed graphs, and dynamic networks represent important directions for investigation. Alternative pooling mechanisms and theoretical analysis of the learned embedding spaces could provide deeper insights. Our work establishes hierarchical representation learning as a promising direction for graph generation, demonstrating that explicit modeling of relational dependencies significantly improves structural coherence while maintaining efficiency.

## Acknowledgments and Disclosure of Funding

This work was conducted under the Laboratory Directed Research and Development Program at PNNL, a multi-program national laboratory operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

## References

[1] Samer Al-Sarayji and Islem Rekik. Graphtreegen: Subtree-centric approach to efficient and supervised graph generation. *arXiv*, abs/2304.09311, 2023.

[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[3] Aleksandar Bojchevski, Petar Bojchevski, S Gunnemann, and S Günter. Netgan: Generating graphs via random walks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[4] Jianliang Cai, Xiao Wu, Lingxiao Huang, Chuan Shi, and Xiaoming Li. Deep generative models for graph generation. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):1–24, 2022.

[5] Jianan Gao, Yanan He, and Yanjun Li. Gnns for link prediction: A survey. *arXiv preprint arXiv:2102.04944*, 2021.

[6] Rafael Gomez-Bombarelli, David Duvenaud, Wei Du, Jessica Lamy, Jesús Gómez-Duque, José Muro, and Alán Aspuru-Guzik. Molecular graph generation with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[7] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Graph generation with destination-driven diffusion mixture. In *International Conference on Learning Representations (ICLR)*, 2022.

[2] This work was performed using the CPU of an Apple M1 Max Chip and 32 GB of memory.

[8] Runxue Li, Guanyao Wang, Ming Wang, Jianmin Wang, and Qiang Li. Information-preserving graph neural networks. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021.

[9] Hongxu Ma, Zejun Xu, Chengyan Deng, Chao Zhang, Peilin Chen, Yanhua Wang, and Chao Lu. A survey on graph pooling methods. *arXiv preprint arXiv:2102.10097*, 2021.

[10] Van Khoa Nguyen, Yoann Boget, Frantzeska Lavda, and Alexandros Kalousis. Glad: Improving latent graph generative modeling with simple quantization. *arXiv*, abs/2403.16883, 2024.

[11] Kevin Preuer, Markus Mautner, Günther Klambauer, Sepp Hochreiter, and Jürgen Schmidhuber. Generative models for molecules: A survey. *Frontiers in Artificial Intelligence*, 4:68, 2021.

[12] Chuan Shi, Xiaoming Li, Yifei Liu, Guilin Zhang, and Lingxiao Huang. Molecular graph generation with attention-based graph neural networks. In *Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN)*, 2021.

[13] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of realistic graphs with deep autoencoders. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[14] Yixin Su, Shuo Wang, Ruifeng Xu, and Hongbo Wang. Enhanced graph autoencoder for graph anomaly detection using subgraph information. *Applied Sciences*, 12(15):7523, 2022.

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

[16] Hongwei Wang, Bingzhe Cui, Wenbin Chen, Jiaxing Yu, Shiliang Shi, and Cheng Yang. Graphgan: Graph representation learning with generative adversarial nets. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

[17] Z Wu, S Pan, F Chen, G Long, C Zhang, and PS Yu. Graph neural networks: A survey. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(2):542–576, 2020.

[18] Weihua Yin, Guilin Zhang, Yifei Li, Haotian Zhang, Zhi Li, and Ruo Wang. Generative models for graph-structured data. *ACM Computing Surveys (CSUR)*, 54(1):1–36, 2021.

[19] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 5708–5717, 2018.

[20] Jie Zhou, G Cui, S Hu, Zhen Zhang, Cheng Yang, Z Liu, L Wang, M Li, and Maosong Sun. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.00030*, 2018.

[21] Yan Zhu, Yuyu Liu, Wenbo Xu, Yizhou Sun, and Hongbin Zha. Constrained graph generation with latent-space regularization. In *Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN)*, 2019.

## A Appendix

### A.1 Dataset Generation Parameters

**Stochastic Block Models (SBMs)** Each graph contains 3 blocks with 5-14 nodes per block, resulting in total node counts of 15-42. We sample distinct probability matrices for each graph: intra-block edge probabilities are drawn from [0.75, 0.95] while inter-block probabilities are sampled from [0.05, 0.15]. This parameterization ensures clear community separation with varying density patterns. Edges are formed using Bernoulli processes based on these block-specific probabilities.

**Random Trees** We generate trees with 5-19 nodes using Prüfer sequences, which provide a bijective mapping between labeled trees and integer sequences. This approach ensures uniform sampling over all possible labeled trees of a given size, creating a diverse set of hierarchical structures for evaluation.

**Disjoint Union of Cycles (DUCs)** Each graph contains 5-19 total nodes distributed across multiple disjoint cycles, with each cycle containing at least 3 nodes. The number of cycles and their sizes are randomly determined while maintaining the target total node count, creating graphs that challenge models to maintain both local structure and global topology.

### A.2 End-to-End Generation Pipeline Example

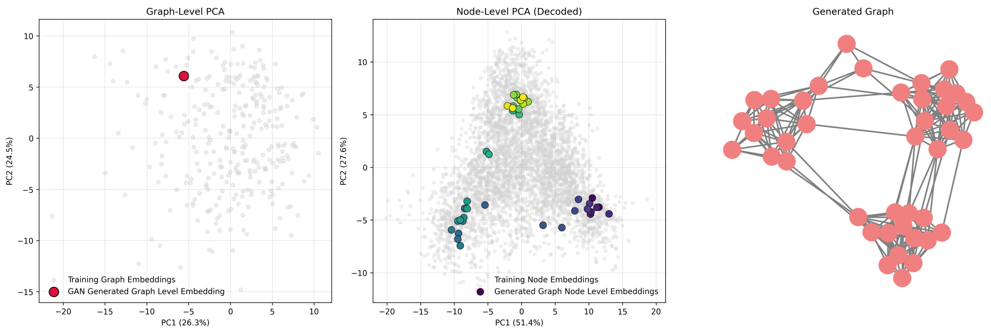
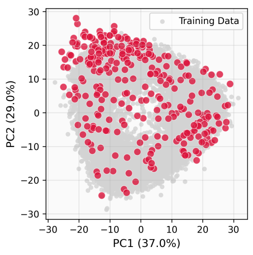
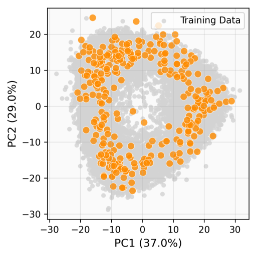


Figure 7: End-to-end generation pipeline example for a single graph showing (left) the generated graph-level embedding highlighted within the PCA projection of all training data graph embeddings, (center) the decoded node-level embeddings for the same graph highlighted within the PCA projection of node embeddings across a sampled set of graphs, and (right) the final reconstructed graph obtained by decoding the node embeddings into an adjacency matrix. This illustrates how a single generated vector in the graph-level latent space expands into a coherent set of node embeddings and ultimately a structured graph.

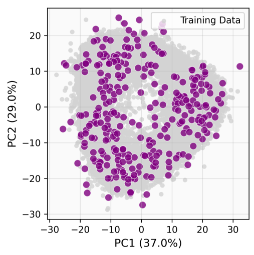
### A.3 Complete Generative Methods Comparison



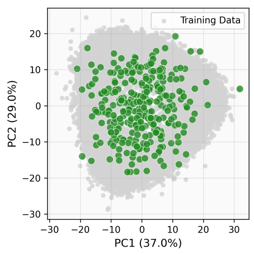
(a) Diffusion Model



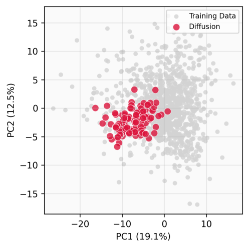
(b) GAN Model



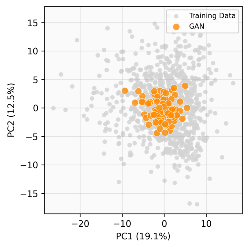
(c) Mixture Density Network



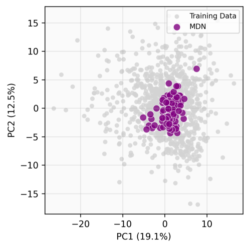
(d) Variational Autoencoder



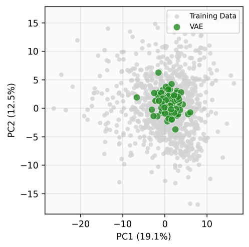
(e) Diffusion (Simulated)



(f) GAN (Simulated)



(g) MDN (Simulated)



(h) VAE (Simulated)

Figure 8: Complete comparison of embedding space characteristics across all generative methods. **Top Row:** PCA comparison of original vs. generated node embeddings for each method. Each subplot shows the 2D PCA projection, with original embeddings in blue and generated embeddings in red. **Bottom Row:** Simulated decoded graph embeddings (after GAE decode and pooling encoder re-embedding) compared to the original pooled embedding background (light gray). The GAN model (shown in main text Figure 6) demonstrates the best performance with substantial overlap in both node-level and graph-level embedding spaces.