Generating Graphs with Specified Properties

Sacha BINDER, Sullivan CASTRO, Matthieu MÉRIGOT-LOMBARD
Master MVA - ENS Paris-Saclay

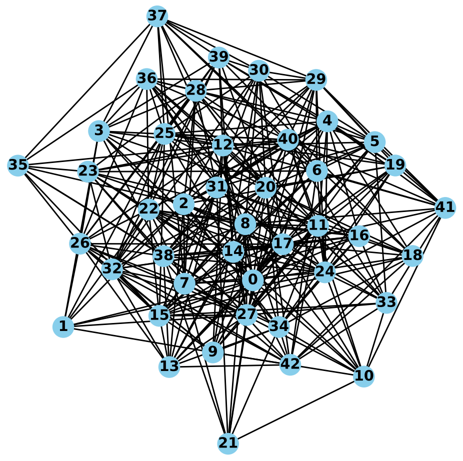
January 2025

1 Introduction

The challenge focuses on neural graph generation conditioned by textual descriptions. This task then combines both natural language processing and graph neural networks. The objective is to create a pipeline capable of generating graphs with specific structural properties described as an input. Features are: the number of nodes, number of edges, average degree, number of triangles, average clustering coefficient, maximum k-core, and the number of communities. This task has diverse applications including simulating social and biological networks, designing molecular structures for drug discovery, modeling urban and telecommunication systems and constructing knowledge graphs highlighting its broad practical utility.

2 Data

The data provided for this challenge is composed of three distinct datasets: a training dataset consisting of 8000 samples, a validation dataset containing 1000 samples and a test dataset comprising another 1000 samples. Both the training and validation datasets are structured as graphs stored in `.edgelist` files. Each graph is paired with a corresponding prompt, an example is illustrated in Figure 1. These prompts are text descriptions that provide context or additional information related to the graphs. However, the test dataset differs as it consists solely of text input prompts without any accompanying graph structures.



```
In this graph, there are 43 nodes connected by 321
    edges. On average, each node is connected to
    14.930232558139535 other nodes. Within the graph,
    there are 545 triangles, forming closed loops of
    nodes. The global clustering coefficient is
    0.35033211913434753. Additionally, the graph has
    a maximum k-core of 11 and a number of
    communities equal to 4.
```

Figure 1: **A dataset element** Graph (on the left) and its associate prompt (on the right).

2.1 Data preprocessing

The graph descriptions in the training and validation datasets include seven distinct features which are described explained Section 1. These features were deterministically extracted during preprocessing using a regex-like algorithm after ensuring that their specific order within the feature vectors was preserved. This consistent ordering is crucial for the conditioning of the denoiser during the training and inference processes.

Moreover, approximately 50.3% of the graph descriptions in the training dataset include an extra feature labeled "closed loops of nodes". This feature provides information about cyclic structures within the graphs and can be either true or flase. To ensure that this information is utilized effectively, we concatenate a boolean into the feature vectors for conditioning.

2.2 The use of LLM

In the previous sections, we explained that a deterministic function applied to text inputs is sufficient to guarantee near-perfect feature extraction. This method achieved a Mean Squared Error (MSE) of 0.089 on the training set and 0.085 on the validation set, along with a Mean Absolute Error (MAE) of 0.082 and 0.079, respectively. This approach is effective for structured or semi-structured text, thus, the challenge recommended the use of LLM.

While LLMs hold potential for applications involving more flexible or varied prompts, we opted to rely on the deterministic approach to ensure accuracy and consistency in feature extraction, as these were essential for the rest of the pipeline.

2.3 Data augmentation

To enhance the diversity of the training dataset and improve the robustness of our model, we implemented a data augmentation strategy by generating new graphs through permutations of the nodes in existing graphs. By applying random permutations to the nodes, we created augmented datasets that preserved the original graph topology while introducing variability in the node ordering. We used three new permutations on each graphs which created a new training dataset of 32000 samples.

There are additional strategies that we did not have the opportunity to explore such as removing random nodes from the graph or generating new graphs by extracting and combining subgraphs from the dataset. These strategies are particularly useful for what described in Section 3.3.2

3 Model

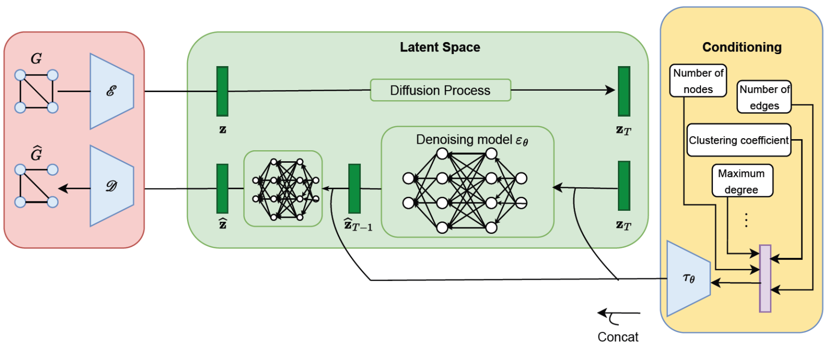


Figure 2: **Pipeline schema.** The VAE ($\mathcal{E}$ and $\mathcal{D}$) is represented by the red part. It encode the graph in a latent spacre represented by the green part. In it, we there is the denoiser. Finally, the conditioning space is represented by the yellow part.

3.1 Architecture

We used the architecture model of the main article given [1] composed of a VAE and a denoiser.

3.1.1 Encoders

We tested different encoders in our architecture: Graph Convolutional Network (GCN), Graph Attention Network (GAT), and Graph Isomorphism Network (GIN).

The GAT introduces an attention mechanism to assign different weights to neighbors during feature aggregation. It learns attention coefficients that emphasize the most relevant neighbors for each node, making it highly adaptable to diverse graph structures. This flexibility provides improved performance for heterogeneous graphs. But, the computational cost of calculating attention weights can make GAT less efficient than simpler alternatives like GCN.

The GIN is designed to capture the isomorphism properties of graphs, enabling it to distinguish between different graph structures with high fidelity. It employs a learnable aggregation function and injects node identity information to enhance its representation power. This architecture achieves strong performance for capturing graph structures accurately. However, GIN requires careful tuning of its parameters to avoid overfitting.

We also test a multi-layer GIN-based GCN to extract both node-level and graph-level embeddings [4, 5]. Each layer consists of a GIN convolution, ReLU activation, and batch normalization, capturing hierarchical features across multiple scales. Outputs from all layers are concatenated and pooled to produce comprehensive graph-level embeddings, which are further refined through a projection head. While highly expressive and capable of preserving detailed structural information, this design can be computationally intensive and prone to over-smoothing in deep networks, limiting its scalability for large or dense graphs.

After conducting experiments using the same hyperparameters, we observed that the GIN model outperformed the GCN model, as expected, given its design for capturing graph structure more effectively. However, when testing the GAT model, we found it to be approximately five times slower than the GIN model, making it less practical for our use case. Consequently, we decided to proceed with the GIN model due to its superior performance and efficiency.

3.1.2 Decoders

For decoding the latent representations obtained after passing through the decoder, we tested two approaches: a Multi-Layer Perceptrons (MLPs) and a GNN-based decoder using convolutions operations.

The MLP-based decoder is a fully connected neural network that maps the latent representations to the desired output format. This approach is versatile and straightforward but could struggle for tasks that require preserving complex structural properties of graph data.

In combinaison with GConv we explore the product decoder [2] to reconstructing graph structures by leveraging node embeddings. It computes the reconstructed adjacency matrix $\hat{A}$ as $\sigma(ZZ^\top)$, where Z is the matrix of node embeddings, $ZZ^\top$ computes pairwise dot products between node embeddings, and σ (sigmoid) ensures the outputs represent probabilities between 0 and 1.

3.1.3 Denoiser

For the denoising component, we employed diffusion models implemented using both MLP and U-Net architectures.

The MLP-based diffusion model is a simpler and computationally efficient approach to denoising, particularly for tasks where the noise is predominantly feature-level and structural dependencies are less critical.

In contrast, the U-Net-based diffusion model utilizes skip connections between its layers enabling it to retain both spatial and hierarchical information throughout the denoising process. This architecture is particularly effective for tasks involving high-dimensional data or data with significant spatial dependencies such as dense graphs or image-like graph representations. However, since the majority of our errors originated from the autoencoder component, we opted not to implement the U-Net model and instead concentrated our efforts on improving the autoencoder.

3.2 Performance metric

The metric used to evaluate the performance of the model is the Mean Absolute Error (MAE) which is computed between the features from the predicted graphs $\hat{f}$ and the real features f from the graph descriptions such that MAE $= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{7}\sum_{j=1}^{7}\left|\hat{f}_{i,j} - f_{i,j}\right|$.

3.3 Loss

3.3.1 Trade-off Reconstruction Loss/KLD

In the Variational Autoencoder (VAE), the loss function is composed of two terms: the reconstruction loss and the Kullback-Leibler Divergence (KLD) such that:

$$\mathcal{L}_{\text{Reconstruction}}(\hat{A}, A) = \mathbb{E}[||\hat{A} - A||]$$

$$\mathcal{L}_{\text{Kullback-Leibler Divergence}}(q|p) = \frac{-1}{2}\sum_{i=1}^{d}(1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2)$$

The reconstruction loss measures how well the model reconstructs input data from the latent space, encouraging accurate reconstructions. A high reconstruction loss can indicate poor reconstruction quality or an ineffective decoder due to an overly complex latent space or elements being too close.

On the other hand, the Kullback-Leibler Divergence (KLD) regularizes the latent space by penalizing the divergence between the learned latent distribution and the prior (usually a standard Gaussian). A high KLD suggests that the latent space deviates significantly from the prior, resulting in a less structured or more scattered representation. Thus, the goal is to minimize KLD without driving it to zero, to maintain a structured space without overfitting or over-constraining.

The optimization objective of a VAE balances these two terms: minimizing reconstruction error while ensuring the latent space follows the desired distribution, typically by adjusting a weight factor β between the two losses.

One major improvement in our approach was reducing β to downscale the importance of the KLD loss, thereby focusing the training more on the reconstruction loss (leading to a reduction from 0.89 MAE on the test set to 0.44). We also implemented adaptive beta scheduling to progressively shift the model focus, initially prioritizing reconstruction loss and later emphasizing KLD loss.

3.3.2 Contrastive Loss

The contrastive loss is a commonly used in tasks involving similarity learning. The goal of this loss is to learn a representation where similar pairs of data points are close together in the feature space and dissimilar pairs are pushed farther apart. It typically operates on pairs of samples where positive pairs (similar data points) are encouraged to have a small distance, and negative pairs (dissimilar data points) are pushed away. A temperature τ controls the scaling of the similarity values, helping to adjust the sharpness of the probability distribution. A higher temperature makes the model more flexible, allowing more pairs to be considered similar, while a lower temperature sharpens the distinction. This can be written:

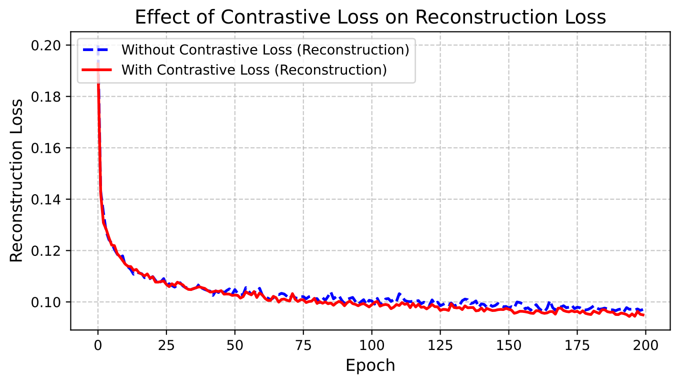
$$[\mathcal{L}_{\text{contrastive}}]_{i,j} = -\log \frac{\exp \frac{\text{sim}(z_i, z_j)}{\tau}}{\sum_{k \neq i} \exp \frac{\text{sim}(z_i, z_k)}{\tau}}$$

For our implementation we use the similiarity cosinus as sim defined as $\text{sim}(u, v) = \frac{u \cdot v}{\|u\|_2 \times \|v\|_2}$.

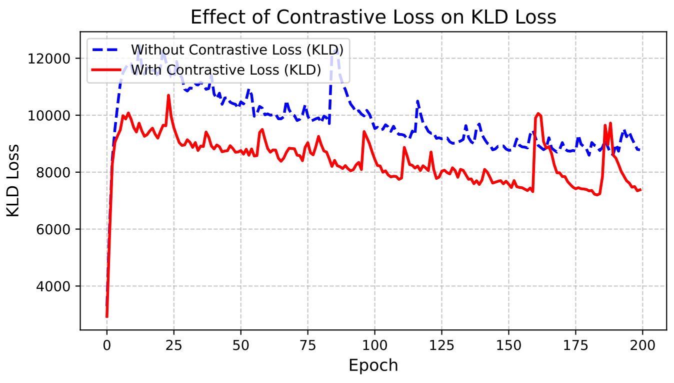
Cosine similarity is a metric used to measure the similarity between two vectors, focusing on their directionality rather than magnitude. In contrastive loss, it encourages latent vectors to explore the entire vector space by promoting spherical separation—similar vectors are pulled together, while dissimilar ones are pushed apart. This leads to more meaningful and balanced separations in high-dimensional spaces.

To create positive pairs, we employed a k-Nearest Neighbors (KNN) approach based on the property features of the training set graphs. By adjusting the number of neighbors, we fine-tuned the model's ability to distinguish between similar and dissimilar pairs, enhancing the quality of the latent space.

Combined with data augmentation, this method improved Kullback-Leibler Divergence (KLD) in Figure 3(b) for a given reconstruction loss in Figure 3(a), reshaping the latent space. This synergy enhanced the model's capacity to capture meaningful relationships, improving the quality of the learned representation.



(a)



(b)

Figure 3: **Contrastive Loss influence** (a) In this figure, we observe that the contrastive loss has minimal impact on the reconstruction loss. (b) In this figure, we see that the contrastive loss helps reduce the KLD loss.

3.4 Tuning hyperparameters

To tune the hyperparameters of the model: hidden dimension encoder/decoders, latent dimension, number of layers encoder/decoder, β... We used the framework *Optuna*. It is an open-source framework for hyperparameter optimization that uses Bayesian optimization to efficiently search for the best hyperparameters in machine learning models. Unlike traditional grid search which explores every possible combination, Optuna builds a probabilistic model of the objective function and selects hyperparameters that are likely to improve performance. This method significantly reduces the number of trials needed by focusing the search on the most promising areas of the hyperparameter space. [3]

Initially, we selected the best model by minimizing the $\text{MAE}_{\text{mean}} + \frac{\text{MAE}_{\text{std}}}{2}$ over the validation set. However, this metric turned out to be unsuitable due to its lack of reproducibility. We should have employed a K-Fold cross-validation, but given the already time-consuming nature of the training process, this was not feasible. Reducing the number of epochs posed a risk of premature convergence. Consequently, we decided to select the best model based on the total validation loss, defined as:

$$\text{Total Loss} = \mathcal{L}_{\text{reconstruction}} + \beta\mathcal{L}_{\text{Kullback-Leibler}} + \gamma\mathcal{L}_{\text{contrastive}},$$

where β and γ are weighting factors that balance the contributions of the individual loss components.

4 Results

The results presented here were obtained using an RTX 3080Ti GPU running on Linux 6.6.69-1-lts with Python 3.12.8. We used batch of size 256. The complete codebase is available on GitHub and is fully reproducible after installing the required dependencies.

4.1 Quantitative Results

	VAE Only (Validation Set)		Global Pipeline (Validation Set)		Global Pipeline (Test Set)			Epochs	Training Time (s)
	Mean ↓	Std ↓	Mean ↓	Std ↓	Mean ↓	Std ↓	Kaggle Score ↓	VAE/Denoiser	
5	0.0675	0.0986	0.0791	0.1293	0.0757	0.1225	0.1421	1000/1000	15min27s
4	0.0811	0.1303	0.0828	0.1290	0.0853	0.1399	0.1689	500/400	7min19s
3	0.0978	0.2033	0.2275	0.5064	0.2303	0.5228	0.2689	200/100	2min27s
2	0.1257	0.2345	0.5513	1.3532	0.5535	1.3695	0.4485	50/25	50s
1	0.8409	1.5876	0.8379	1.5896	0.8485	1.6171	0.8917	200/300	3min24s

Table 1: **Quantitative results.** Performance comparison of the VAE-only model and the global pipeline on both validation and test sets. The table also includes training times in seconds. The different configurations evaluated are as follows: (1) the base model with a score of 0.89, (2) the base model with a downsized beta, achieving a score of 0.45, (3) the base model with a downsized beta and hyperparameters optimized jointly, along with an adaptive beta, resulting in a score of 0.27, (4) the base model with a downsized beta and hyperparameters optimized one by one, in addition to an adaptive beta, achieving a score of 0.17, and (5) the base model with a downsized beta, hyperparameters optimized one by one, an adaptive beta, contrastive loss, and data augmentation, yielding the best score of 0.14.

4.2 Discussion

Upon analyzing the MAE of the reconstructed graphs across multiple samples, we observed that the VAE struggled to accurately reconstruct the data. This directed us to focus our efforts on improving the autoencoder. The most significant improvement came from reducing the β parameter and tuning the hyperparameters. The primary issue was the autoencoder's inability to accurately model the latent space distribution which was deduced to be from an excessively high KLD. By reducing β, the model was better able to capture the correct latent space distribution, leading to improved denoiser performance during training. These optimizations, combined with hyperparameter tuning, resulted in a fourfold reduction in the benchmark error.

We changed our approach to hyperparameter optimization from a joint optimization to a sequential one. Initially, we used grid search to explore all the main parameters at once (e.g., number of layers, hidden dimensions, latent space dimensions). However, this approach didn't allow us to capture the specific nuances of the best hyperparameters (e.g., smaller latent spaces being more effective). By reducing the number of epochs and optimizing one parameter at a time while keeping others fixed, we were able to identify more accurate hyperparameters.

Using data augmentation, we quadrupled our dataset, which resulted in multiple permuted graphs. The contrastive loss, combined with the KNN feature, became more effective by grouping all permuted graphs together while giving us control over whether to include graphs outside the permuted group. This led to a slight improvement in the results, suggesting that further refinement is needed. We reviewed a method [4] for a novel approach to data augmentation with contrastive loss. Rather than permuting, this method involves removing edges, nodes, subgraphs, or applying masks, with augmentation applied within each batch. Our plan is to pretrain the autoencoder using contrastive loss with this augmented data as positive pairs, then train the encoder and decoder jointly. While we were in the process of implementing this data augmentation approach, time constraints led us to pursue a different strategy for the time being.

One challenge encountered during training was the lack of clarity in the loss metrics: despite similar reconstruction loss and KLD values across epochs, the final MAE could vary significantly. Additionally, discrepancies were observed between the computed MAE on the seven features and the results reported on the Kaggle platform which added another layer of complexity to the evaluation process. We also observed unusual fluctuations in the KLD: it decreased as the batch size increased, leading to misinterpretations early in the challenge.

References

[1] Christos Xypolopoulos Ahmed Kammoun Michail Chatzianastasis Hadi Abdine Iakovos Evdaimon, Giannis Nikolentzos and Michalis Vazirgiannis. Neural graph generator: Feature-conditioned graph generation using latent diffusion models, 18 Sept 2024. 2

[2] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016. 3

[3] Toshihiko Yanase Takeru Ohta Masanori Koyama Takuya Akiba, Shotaro Sano. Optuna: A next-generation hyperparameter optimization framework. 2019. 4

[4] Yongduo Sui Ting Chen Zhangyang Wang Yang Shen Yuning You, Tianlong Chen. Graph contrastive learning with augmentations. 2020. 2, 5

[5] Yanqiao Zhu, Yichen Xu, Qiang Liu, and Shu Wu. An Empirical Study of Graph Contrastive Learning. *arXiv.org*, September 2021. 2