ALTEGRAD - 2024-25 Data Challenge

Guillaume Pradel
Institut Polytechnique de Paris

Maxence Lasbordes
PSL Research University

Abstract

The aim of this data challenge on Kaggle is to generate graphs from prompts using artificial intelligence methods. To achieve this, we explored various strategies to improve the baseline model provided for the challenge. This included experimenting with different approaches such as conditional Wassertein Generative Adversarial Networks with Gradient Penalty (cWGAN-GP), contrastive learning, and data augmentation techniques. We also investigated novel methods to encode the conditional vector derived from the textual description of the graph using a BERT model, aiming to better capture the semantic information in the prompts. Additionally, we planned to explore fine-tuning a large language model (LLM) for graph generation, although hardware limitations prevented us from fully implementing this approach. By refining these components, we aimed to enhance the model's ability to generate realistic and consistent graphs that align with the properties specified in the description.

1 Introduction

Graph generation is an increasingly studied method, particularly in biology and chemistry. Methods like MolGAN [1] or Junction Tree VAE [7] are being developed to create new molecules with desired properties. The main challenge of these methods is to create graphs consistent with the structural reality of the molecules, for example, the presence of conjugated rings in aromatic molecules. This data challenge is analogous to the previous methods in that we need to generate graphs structured by the prompt's "properties".

We're working on a labelled database, with graphs and their associated prompts. The baseline for this project is given by the output of the NGG model, presented in Evdaimon, et al. (2024) [2], with the same parameters shown in the section : Experimental setup. Like the methods described earlier, its aim is to produce realistic graphs that respect specific properties. The NGG structure is based on three main stages: a variational autoencoder (VAE), a latent space diffusion process, and a decoder. The VAE transforms a graph into a latent representation, projecting its features into a low-dimensional space. Next, a diffusion model gradually introduces noise into these latent vectors, while an inverse process learns to denoise them, conditioning this step on global properties of the graph, in this case the features contained in the prompt. Finally, the decoder uses the denoised latent representation to reconstruct a complete graph.

To address this challenge, we first aimed to improve the baseline model by implementing a contrastive learning approach, utilizing data augmentation, and encoding prompts using a pre-trained BERT model. Additionally, we introduced an innovative method—a Conditional Wasserstein GAN with Gradient Penalty (cWGAN-GP) [5]—which yielded our best results for this problem. We evaluated all approaches using our evaluation benchmark, which computes a normalized Mean Absolute Error (MAE) based on the feature vectors extracted from the generated adjacency matrices.

The dataset used consists of various graphs paired with their natural language descriptions. The training set contains 8,000 samples, the validation set includes 1,000 samples, and the test set also comprises 1,000 samples. Given that we work with medium-sized graphs of 50 nodes, the training set is relatively small. This limited size makes our models highly susceptible to overfitting, which has been a significant challenge in this project.

2 Proposed Approaches

2.1 NGG model (Baseline)

The NGG model is not the main focus of this report; however, we experimented with this baseline using various hyperparameter configurations to attempt to improve the results. Unfortunately, we did not achieve any significant improvements without incorporating additional techniques into this model.. The NGG model operates using two main components: a VGAE (Variational Graph Autoencoder) and a Diffusion model. The latent representation is conditioned on the feature vector constructed from the graph description.

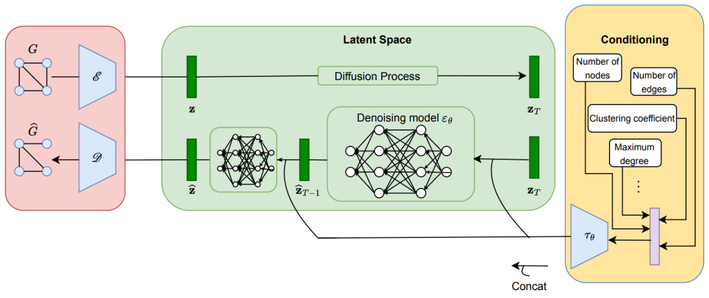

Figure 1: Overview of the NGG architecture [2]. The variational graph autoencoder generates a compressed latent representation $\mathbf{z}$ for each graph G. These representations are fed into the diffusion model, which operates in the latent space, adding noise to $\mathbf{z}$ to produce $\mathbf{z}_T$. The denoising process is conditioned on the encoding (output of τ_θ) of the vector containing the graph's properties. The output of the diffusion model is then passed to the decoder, which reconstructs the graph.

2.2 Data augmentation

As the database is small, it would be interesting to implement a data augmentation algorithm. In other words, to add data to train the model. To do this, we created random graphs, first choosing a number of nodes between 1 and 50, then a number of edges between $n-1$ and $\binom{n}{2} = \frac{n\cdot(n-1)}{2}$. Table 2 presents the training information, highlighting the differences in training time and loss evolution with and without data augmentation for the same number of epochs.

Table 1: Training information for the NGG model when adding 50 000 new graphs with an Nvidia RTX 3060.

	NGG + Aug. Data	NGG (baseline)
VGAE - Train time (s)	803	446
VGAE - Train Loss	0.21796	0.22287
VGAE - Val Loss	0.21498	0.22287
Denoising - Train time (s)	360	176
Denoising - Train Loss	0.04591	0.02036
Denoising - Val Loss	0.05430	0.02452

2.3 Contrastive learning

To improve the performance of an existing model, it is possible to change the way the model is trained. This makes it possible to use a method known as contrastive learning. The beginnings of this method are examined in Hadsell et al. (2006) [3]. Later, contrastive learning was adapted to data representing graphs in Ju et al. (2006) [8].

Contrastive learning is a self-supervised learning method used to learn data representations by maximizing similarity between pairs of positive samples and minimizing similarity with negative samples. In our case, this method should allow the VGAE to produce close latent representations for similar pairs and distant latent representations for dissimilar pairs. In order to do this, we implement the same loss as used in [8], to compare the embeddings in the latente space : h_i, h'_i.

$$\mathcal{L}_{\text{con}}(h_i, h'_i) = -\log \frac{\exp\left(\text{sim}(h_i, h'_i)/\tau\right)}{\exp\left(\text{sim}(h_i, h'_i)/\tau\right) + \sum_{j\neq i} \exp\left(\text{sim}(h_i, h_j)/\tau\right)}$$

There are several methods for creating positive pairs, but we've decided to concentrate on just one: node droppping, removing between 1 and 2 nodes from each graph depending on its size.

Table 2: Training information for the NGG with constrastive learning.

	Constrastive Learning
VGAE Train Loss	0.92103
VGAE Val Loss	0.22357
VGAE Train time (s)	185

2.4 Encode the prompt with a Language Model

As explained previously, the baseline model reconstructs graphs based on a feature vector derived by extracting key numerical values from the graph description. To enhance this approach, we aimed to provide the diffusion model with a more meaningful representation of the graph's features by conditioning it on the full encoded prompt. To achieve this, we explored the use of a pre-trained BERT language model [6] to encode the entire textual prompt, replacing the traditional numerical feature vector with the CLS token from the BERT encoding. Leveraging BERT's contextual understanding, this method captures richer semantic information and nuanced relationships within the prompt text. The BERT embeddings, represented as 768-dimensional vectors for each token, are integrated into the diffusion model's training process, enabling graph generation conditioned on a comprehensive and semantically rich representation of the prompt. When encoding the prompt with BERT, the model has slightly more parameters to learn, as the linear layer's input dimension matches BERT's embedding size (768) instead of the baseline feature vector's size (7).

Table 3: Training information for the NGG model when encoding the prompt using a pre-trained BERT language model with an Nvidia RTX 3060. The training of the VGAE remains the same as in the NGG baseline model; only the training of the denoising model is modified.

	NGG + BERT enc.	NGG (baseline)
Preprocess time (s)	733	156
Denoising - Train Loss	**0.01361**	0.02036
Denoising - Val Loss	**0.01427**	0.02452
Denoising - Train time (s)	167	176

The use of the BERT model to encode the graph prompts resulted in a more than sixfold increase in preprocessing time for the training set (cf. Table 3). This is because each graph description must be encoded by the BERT model, which is a computationally intensive step. However, the training time remains roughly the same, and we observe a slight reduction in the training and validation loss for the denoising model. There is no change in the training of the VGAE, as its training process is independent of the prompt.

2.5 Conditional Wasserstein Generative Adversarial Networks with Gradient Penalty (cWGAN-GP)

To enhance graph generation based on specific properties, we explored Conditional Generative Adversarial Networks (cGANs) [9]. Unlike regular GANs [4], cGANs condition the generation process on additional input, such as a feature vector. In our case, we condition the generation of graphs using the feature vector from the graph's prompt, ensuring the generated graphs align with the desired properties. The generator learns to produce realistic graphs, while the discriminator evaluates their authenticity and tries to predict whether a given graph, in the form of an adjacency matrix, has been generated or is real. This approach enables the generation of graphs that match the specified features while maintaining realistic structures as they will be inspired from the dataset.

The conditioning is achieved by concatenating the feature vector with the randomly generated noise vector, which is then fed as input to the generator. Similarly, the feature vector is concatenated with the adjacency matrix and provided as input to the discriminator. During inference (e.g., for the test set), the discriminator is no longer used, and the graph is generated solely by the generator using a random noise vector concatenated with the feature vector from the graph description.

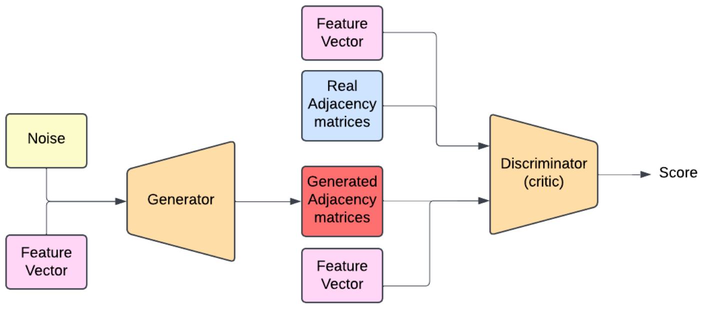

Figure 2: Overview of the conditional WGAN architecture (cWGAN-GP), the conditioning is done by concatenation with the input for the generator and the discriminator.

The standard conditional GAN was very unstable and we were not able to train the model correctly so we decided to use the Wasserstein GAN with Gradient Penalty (WGAN-GP) [5] conditioned on the feature vector from the graph description. This is why we have negative losses, as the loss changes and the discriminator becomes a critic which outputs a raw score instead of a positive probability.

The Wasserstein GAN with Gradient Penalty, is different from the vanilla GAN as it introduces a gradient penalty term to enforce the Lipschitz constraint. The loss functions for the generator G and the critic D are defined as follows:

$$L_D = \underbrace{\mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)]}_{\text{Wasserstein loss for } D} + \underbrace{\lambda\,\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Gradient penalty}}, \tag{1}$$

$$L_G = -\mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})], \tag{2}$$

where G is the generator, D is the critic (discriminator), $\mathbb{P}_r$ is the real data distribution, $\mathbb{P}_g$ is the generated data distribution (output of G), $\hat{x}$ is sampled uniformly along straight lines between points in $\mathbb{P}_r$ and $\mathbb{P}_g$, and λ is the gradient penalty coefficient.

Finally, to improve model performance, we performed one-hot encoding on 2 features in particular: nodes and edges. For the other features, we left them as is. We also used data augmentation to create a larger training set, thus diversifying it and enabling the model to better capture the information contained in the features. So, feature engineering was very basic, and we didn't perform any complex transformations on our data.

Table 4: Training information for the conditional Generative Adversarial Networks (cWGAN-GP) for 700 Epochs with one-hot encoding and 13,000 additional samples with data augmentation.

	cWGAN-GP + one-hot enc. + Data Aug.
Train time (s)	830

2.6 Fine-tune a LLM for graph generation

We want to mention that in our exploration of graph generation methods, we investigated the use of large language models (LLMs) as an alternative approach, despite experiments conducted in the paper by Evdaimon, et al. (2024) [2] demonstrating that LLMs underperformed compared to the baseline in terms of results while requiring less memory. We initially planned to fine-tune a pre-trained 0.5 billion parameter Qwen model for this task, with the goal of generating adjacency matrices directly from graph descriptions. However, due to hardware limitations, particularly insufficient VRAM (even on google collab), we were unable to proceed with these experiments. The challenges included the need to transform the dataset and adjacency matrices into suitable prompts for the LLM, which proved computationally infeasible given our resources. As a result, we were unable to fully explore this direction, highlighting the significant hardware constraints associated with leveraging LLMs for graph generation tasks.

3 Results and Analysis

3.1 Benchmark: Normalized MAE

To compare the graphs generated by our models with the ground truth, we replicated (with our interpretation) the evaluation process described in the project report. The process involves generating adjacency matrices for 1,000 graph descriptions using the trained models on the provided test samples. We then compute their corresponding feature vectors (using the same 7 features as in the baseline) and calculate the average Mean Absolute Error (MAE) after normalizing each feature in the vectors. This normalization ensures that each feature contributes equally to the final result. The outcome is a Normalized MAE value between 0 and 1 for all configurations, where a lower value indicates better performance.

3.2 Results

We will now compare and analyze the performance of the previously proposed methods and comparing them against the baseline on our benchmark.

Table 5: MAE results on the test set for various configurations. Performance is measured by averaging the MAE across 1,000 test samples. For each sample, the MAE is computed by comparing the vector of 7 graph properties normalized for each feature, for the generated graph G against the ground truth graph G'. Lower MAE values indicate better performance.

	MAE Normalized	Epochs
NGG (baseline)	0.4355	200
NGG + BERT enc.	**0.2926**	200
NGG + Contrastive learning	0.3089	200
NGG + Aug. Data	0.2962	200
cWGAN-GP	0.08819	500
cWGAN-GP + One-hot enc. + Aug. Data	**0.0409**	700
cWGAN-GP + One-hot enc.	0.07272	500
cWGAN-GP + BERT enc.	0.2928	500

Almost all the proposed methods lead to better performance with a lower Normalized MAE. In particular, cWGAN-GP with one-hot encoding and data augmentation achieves the best performance, with a Normalized MAE of 0.0409, significantly outperforming other methods, including the NGG baseline.

4 Conclusion

In this project, we explored various strategies to improve graph generation from textual prompts, focusing on enhancing the baseline NGG model. The conditional Wasserstein GAN with Gradient Penalty (cWGAN-GP), especially when enhanced with one-hot encoding and data augmentation, delivered particularly promising results. Those results could probably be further increased with an even larger data augmentation et more training epochs, but we lacked time and compute power. We also investigated other techniques, such as encoding textual prompts using a pre-trained BERT model and contrastive learning. While these methods were not as effective as cWGAN-GP, they still improved performance on our MAE benchmark compared to the baseline. Additionally, we briefly explored VAE-GMM, which showed potential but was outperformed by cWGAN-GP. Although cWGAN-GP performed well, future work could explore reinforcement learning (RL) in conjunction with GANs, as seen in methods like MolGAN. Overall, our efforts to generate realistic and consistent graphs aligned with prompt properties through diverse approaches yielded conclusive results.

References

[1] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs, 2022.

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

[3] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, 2006.

[4] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J. Goodfellow, Jean Pouget-Abadie. Generative adversarial networks, 2014.

[5] Martin Arjovsky Vincent Dumoulin Aaron Courville Ishaan Gulrajani, Faruk Ahmed. Improved training of wasserstein gans, 2017.

[6] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[7] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation, 2019.

[8] Wei Ju, Yifan Wang, Yifang Qin, Zhengyang Mao, Zhiping Xiao, Junyu Luo, Junwei Yang, Yiyang Gu, Dongjie Wang, Qingqing Long, Siyu Yi, Xiao Luo, and Ming Zhang. Towards graph contrastive learning: A survey and beyond, 2024.

[9] Simon Osindero Mehdi Mirza. Conditional generative adversarial nets, 2014.