# M2 - ALTEGRAD course
# "Neural Graph Generation with conditioning"
# 2024-25 Data Challenge

DaSciM group
LIX, ´Ecole Polytechnique
November 17, 2024

## 1. Challenge Description

The goal of this project is to study and apply machine learning/artificial intelligence techniques to generate graphs of specific properties. One of the most challenging tasks of machine learning on graphs is that of graph generation. Graph generation has attracted a lot of attention recently, and its main objective is to create novel and realistic graphs. For instance, in chemo-informatics, graph generative models are employed to generate novel, realistic molecular graphs that also exhibit desired properties (e.g., high drug-likeness). Recently, there has been a surge of interest in developing new graph generative models, and most of the proposed models typically fall into one of the following five families of models: (1) Auto-Regressive models; (2) Variational Autoencoders; (3) Generative Adversarial Networks; (4) Normalizing Flows; and (5) Diffusion models [Zhu et al., 2022, Guo and Zhao, 2023]. These models can capture the complex structural and semantic information of graphs. In this challenge, given a text query that describes some properties of the structure of the graph, you need to generate the graph corresponding to the description. The pipeline to deal with this task can be achieved by training a latent diffusion for conditional graph generation.

The challenge is hosted on Kaggle, a platform for predictive modeling on which companies, organizations and researchers post their data, and statisticians and data miners from all over the world compete to produce the best models. The challenge is available at the following link:
https://www.kaggle.com/competitions/generating-graphs-with-specified-properties/
To participate in the challenge, use the following link:
https://www.kaggle.com/t/e6fa16aac0a84124b5e8b63780b2452a

## 2. Baseline Description

In this section, we introduce the used baseline that is based on NGG [Evdaimon et al., 2024]. We present the two main components of the proposed NGG model: (1) the variational graph autoencoder which produces a compressed latent representation for each graph; and (2) the diffusion model which performs diffusion in the latent space and which can be conditioned on various inputs (vector of graph properties in our case). An overview of the proposed model is given in Figure 1. For the the variational graph autoencoder: Given a graph G, the encoder E encodes the graph into a latent representation z = E(G), and the decoder D reconstructs the graph from the latent representation, giving ˆG = D(z) = D(E(G)), where $z \in R^d$.

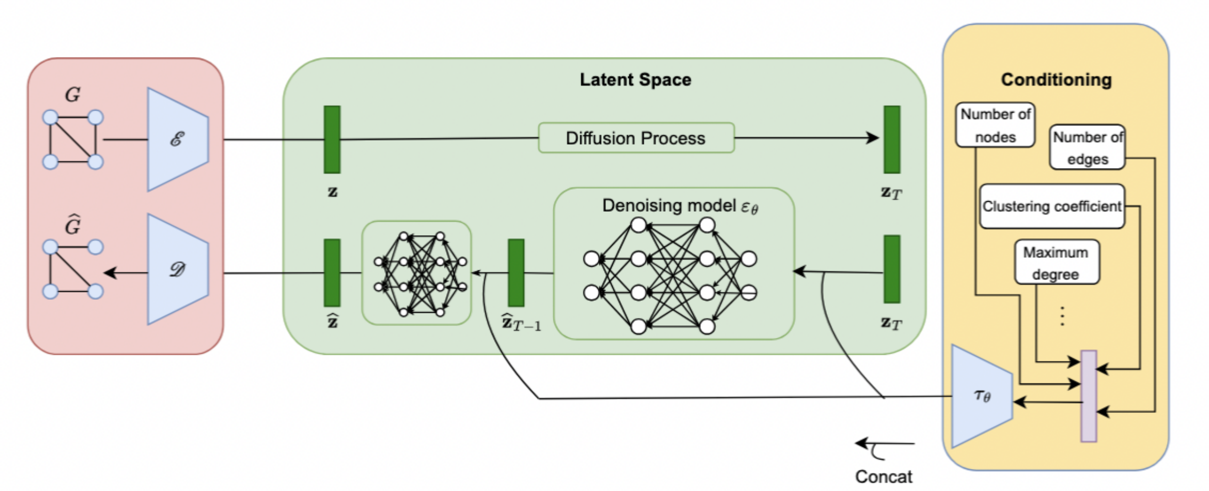

Figure 1: Overview of the proposed architecture. The variational graph autoencoder is responsible for generating a compressed latent representation z for each graph G. Those representations are fed to the diffusion model which operates in that latent space adding noise to z resulting to $z_T$. The denoising process is conditioned on the encoding (output of $\tau_\theta$) of the vector that contains the graph's properties. The output of the diffusion model is passed on to the decoder which generates a graph.

**Latent Diffusion Model**. Once the autoencoder is trained, we can use the encoder E to embed graphs into a low-dimensional latent space. Latent diffusion models consist of two main components: (1) a noise model; and (2) a denoising neural network. The noise model q progressively corrupts the latent representation of the input graph z to create a sequence of increasingly noisy vectors ($z_1$, . . . , $z_T$ ). Specifically, the forward process q samples some noise from a Gaussian distribution at each time step t, which is added to the representation of the previous time step. The second component of the latent diffusion model (i.e., the denoising neural network) is responsible for predicting the added noise for a given time step t. We implement the denoising neural network $\epsilon_\theta$ as an MLP. Once the denoising neural network $\epsilon_\theta$ is trained, to generate new data, we sample a vector of pure noise $z_T$ from a Gaussian distribution and then use the neural network to gradually denoise it (using the conditional probability it has learned). The denoised vector can then be transformed into a graph with a single pass through the decoder D.

### 3. Dataset Description

As mentioned above, you will evaluate your methods on a dataset consisting of graphs and their descriptions. You are given the following files (available at: https://drive.google.com/file/d/1Ey54FhVnIUlryhV_AwUFykp4mdjUvcuI/view?usp=sharing). In the folder *data*, you will find three subfolders:

1. train: This folder includes two subfolders named graphs and descriptions. Each graph in the graphs folder has a corresponding textual description with the same filename in the description

folder. The graphs folder contains files in both edgelist and graphml formats, while the description folder contains txt files. For example, the graph file *data/train/graphs/graph_1.edgelist* has a corresponding description in *data/train/description/graph_1.txt* Overall, the training set consists of 8,000 samples.

2. valid: Same as train. The validation set contains 1,000 samples

3. test: Which contains test.txt file. A text file contains 1,000 textual descriptions from the test dataset. **This dataset should not be shuffled**.

### 4. Evaluation

For each generated graph ˆGi, we compute a vector where each component is the value of each one of the 7 considered properties. Then, this vector is compared against the vector of properties of Gi. We use as metrics, the mean absolute error (MAE), while lower values indicate better performance, as they represent a smaller difference between the predicted and actual values. In order to prepare your submission file, you need to create a csv file with an ID column representing the ID of each generated graph (from 0 to 999 in the same order as the given test text file).

*The evaluation of your submission in Kaggle takes several minutes (~20). Thus, be patient!*

### 5. Provided Source Code

You are given Python scripts that will help you get started with the challenge.

• main.py: This script is used to set the model, and its hyperparameters, and to train it. The model is trained for 200 epochs in the case of VGAE and for 100 epochs in the case of the diffusion model. The model with the best performance on the validation dataset is used for the generation of the graphs.

• autoencoder.py: This script contains the class that implements the VGAE model
• denoiser model.py: This script contains the class that implements the diffusion model
• utils.py: contains many useful methods to preprocess the data or to set model parameters like the scheduler of the diffusion process.
• extract feats.py: This script extracts the graph properties from the description of the graph.

### 6. Suggestions

We encourage you to go beyond the baselines with novel ideas. So, you can consider experimenting with:
1. Various Contrastive Learning Techniques.
2. Try Different Encoders.
3. Try Different Decoders.
4. Use the language model as encoder of the prompt and not just to extract the required features from the prompt.
5. Try different language models that encode prompts.

6. Other generative models: (i) VAE, (ii) diffusion models, (iii) GANs, (iv) Normalizing flows, (v) autoregressive models.
7. Use of LLMs

## 7. Useful Python Libraries

In this section, we briefly discuss some tools that can be useful in the challenge and you are encouraged to use:

- A very popular deep learning library in Python is PyTorch. The library provides a simple and user-friendly interface to build and train deep learning models.
- PyG (PyTorch Geometric) is a library built upon PyTorch to easily write and train Graph Neural Networks (GNNs) for a wide range of applications related to structured data.
- Networkx, a library for creating, analyzing, and visualizing complex networks or graphs.

It provides tools to build and manipulate both undirected and directed graphs.

## 8. Rules and Details about the Submission of the Project

**Rules**. The following rules apply to this challenge: (i) one account is allowed per participant; (ii) there is a limit on the size of each team (at most 3 members); (iii) privately sharing code outside of teams is not permitted; (iv) there is a limit on the number of submissions per day (at most 4 entries per day); (v) the use of external data and code is not allowed (except word embeddings, e. g., BERT, GPT, BART, WordVec, etc.). For instance, you are not allowed to use external data to determine if a summary is generated by a machine or written by a human. (vi) your code must be reproducible.

**Evaluation and Submission**. Each team must fill this form before 20/11/2024 Your final evaluation for the project will be based on (1) the group oral presentation (50%), (2) your total approach to the problem, and the quality of the report (30%), and (3) your score on the private leader-board based on the accuracy score that will be achieved (20%).

**Deliverable.** As part of the project, you have to submit the following:

- *A 4-5 pages report* (excluding references), in which you should describe the approach and the methods that you used in the project. Since this is a real generative task, we are interested to know how you dealt with each part of the pipeline, e.g., how you created your representation, which features did you use, which contrastive learning techniques did you use and why, the performance of your methods (loss, accuracy and training time), approaches that finally didn't work but are interesting, and in general, whatever you think that is interesting to report.
- *A folder with the code of your implementation* (not the data, just the code).

**Submission**: **Create a .zip file containing the code and the report and submit it here. *Make sure that the name of the file is as follows: team name.zip.***

**Deadline** (for both competition and submitting code and report): 15/01/2025 23:59.

**Presentation**: As mentioned above, you will be asked to present orally as a group the approach you followed, and the results achieved. The presentation duration (including questions will be 20 mins).

**Date of presentation**: *The dates of the presentation will be towards the 3rd week of January and will be announced in due time.*

**References**

[Evdaimon et al., 2024] Evdaimon, I., Nikolentzos, G., Xypolopoulos, C., Kammoun, A., Chatzianastasis, M., Abdine, H., and Vazirgiannis, M. (2024). Neural graph generator:Feature-conditioned graph generation using latent diffusion models.

[Guo and Zhao, 2023] Guo, X. and Zhao, L. (2023). A systematic survey on deep generative models for graph generation. IEEE Trans. Pattern Anal. Mach. Intell., 45(5):5370–5390.

[Zhu et al., 2022] Zhu, Y., Du, Y., Wang, Y., Xu, Y., Zhang, J., Liu, Q., and Wu, S. (2022). A survey on deep graph generation: Methods and applications. In Learning on Graphs Conference, pages 47–1. PMLR.