DIISM *04 - 2024*

Graph Latent Diffusion

Tommaso Ancilli – Duccio Meconcelli

Overview

1.

Train the **GraphVAE** model to generate organic molecules

2.

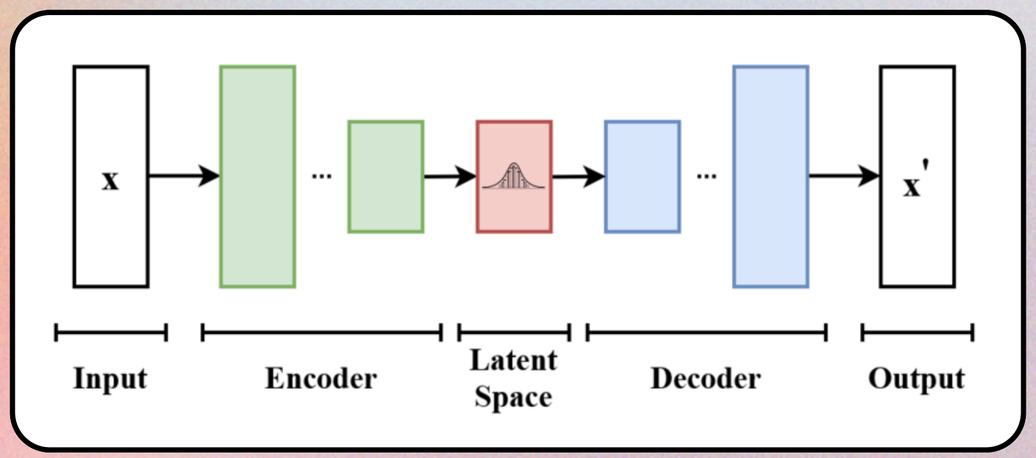
Implement a **Latent Diffusion Model** that takes advantage of the **GraphVAE** *Encoder* and *Decoder*

3.

Understand what advantages **Latent Diffusion** brings compared to standard **GraphVAE**

VAE

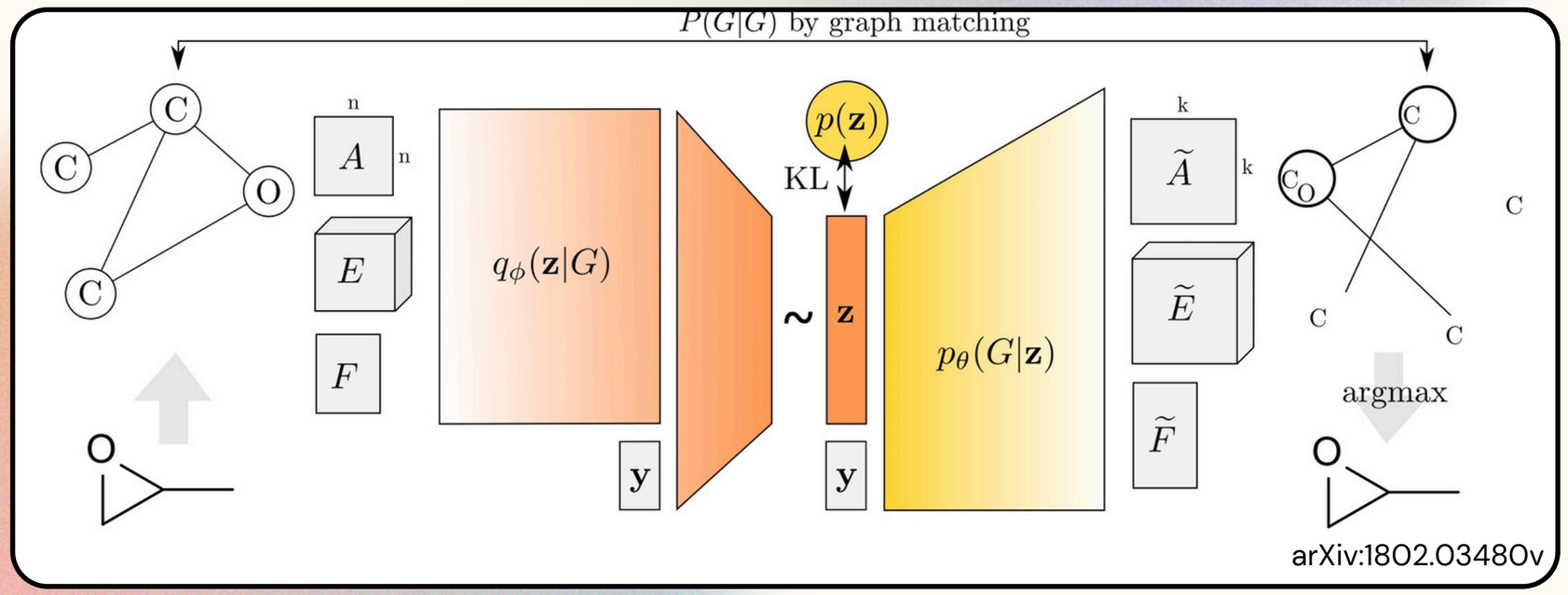
- *Variational Autoencoders* (**VAE**) are a class of generative models that use deep learning techniques to generate data.
- VAEs are based on the concept of autoencoder, a neural network used for data compression and decompression.
- Unlike autoencoders, VAEs introduce an element of statistical variability to improve example generation.
- The operation of VAEs is based on the combination of a probabilistic **encoder** and **decoder**.
  - The ***encoder*** maps the input data into a probability distribution in the latent space,
  - the ***decoder*** reconstructs the original data from random samples drawn from the latent space.
  - This process is driven by the goal of *maximizing the likelihood* of the training data, while ensuring that the latent distribution is similar to a known probability distribution, usually a normal distribution.



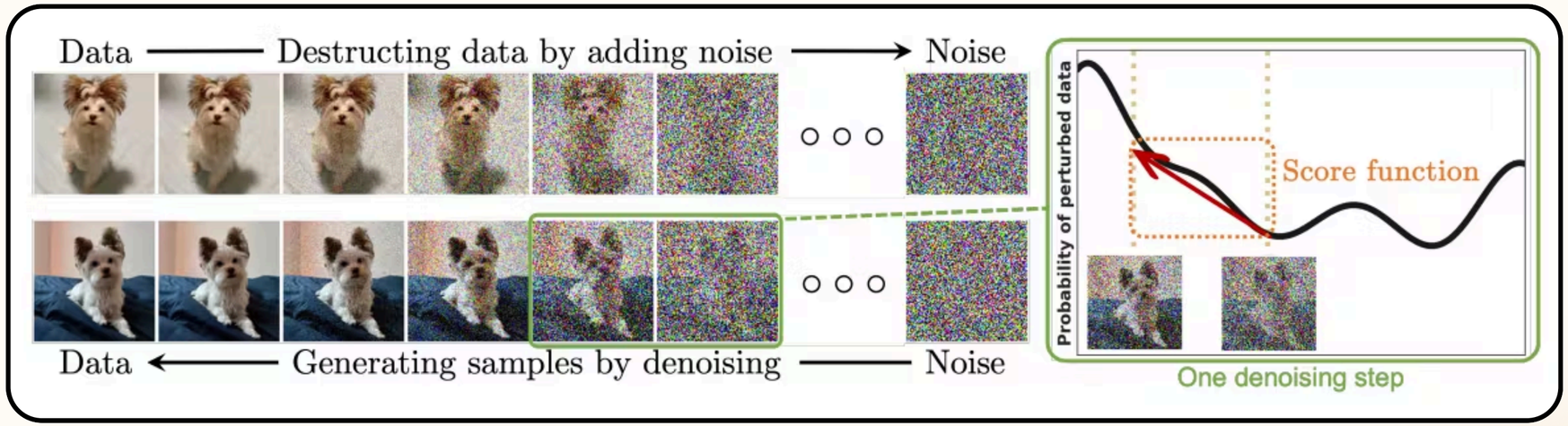
Graph VAE

- **GraphVAE**s, or Variational Autoencoders for Graphs, extend the concept of VAE to the *graph domain*. These models are capable of generating new graphs that mirror the properties of an input graph dataset.

- In a GraphVAE, the encoder takes a graph as input and encodes it into a continuous *latent space*. The decoder then takes as input a point in the latent space and generates a new graph. This process allows the generation of graphs that maintain the structural properties of the input graphs, making GraphVAEs useful in a variety of applications, such as chemistry and social network analysis.



Diffusion Models



- Classical diffusion models operate on the fundamental principle of diffusion, a process that describes the spread of information or particles across a system over time. In the context of generative modeling, diffusion models simulate the evolution of data by iteratively adding noise to a starting point, gradually transforming it into a sample from the target distribution

- At the heart of classical diffusion models lies the diffusion process itself, governed by stochastic differential equations or discrete-time dynamics. These equations dictate how the data evolves over time, incorporating factors such as noise level and step size to control the rate and extent of diffusion

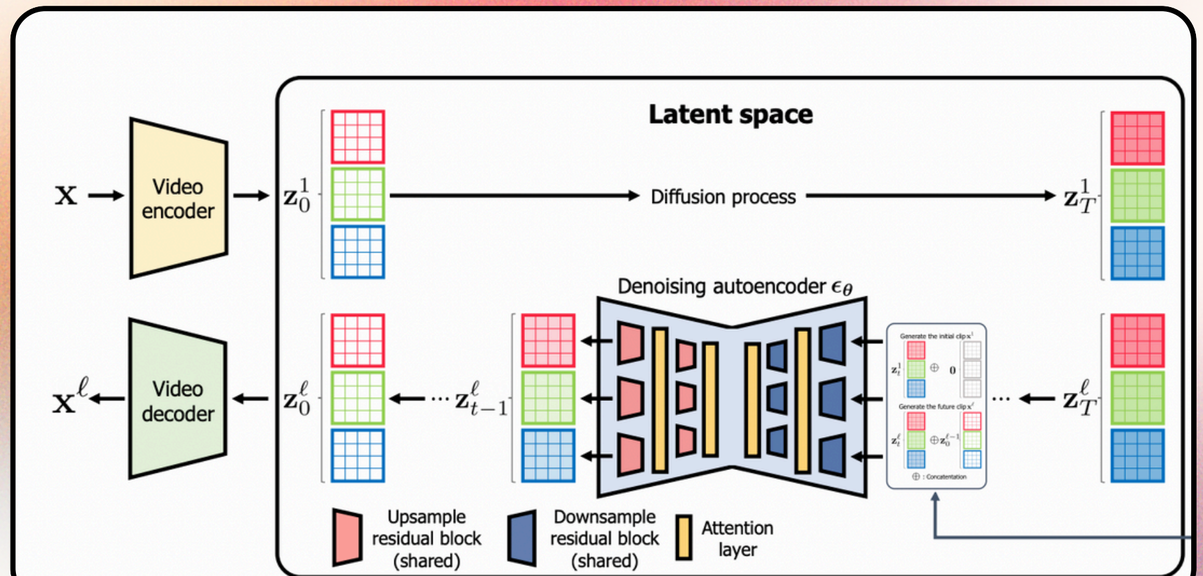
- The goal of diffusion models is to learn a diffusion process that generates a probability distribution for a given dataset from which we can then sample new data

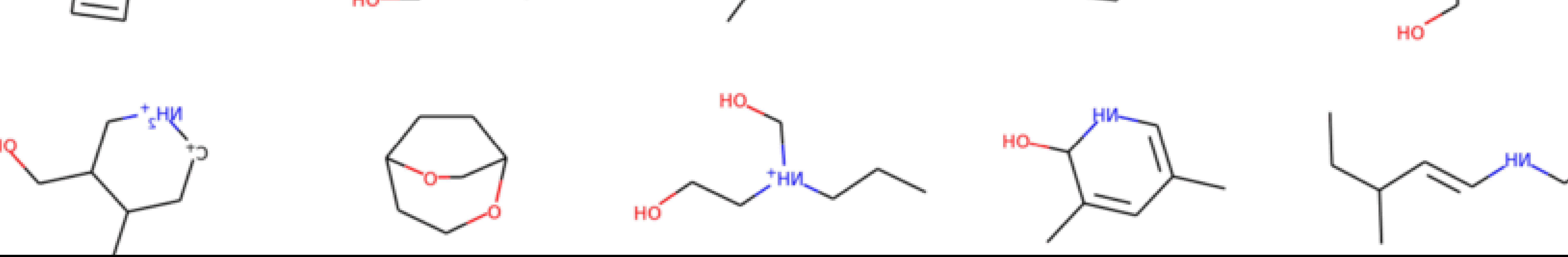
Latent Diffusion

- Latent Diffusion represents an evolution of classic diffusion models, which is based on the idea of diffusing the data not directly, but rather a latent representation of the data.

- They can be used in combination with pre-existing generative models, such as Variational Autoencoders (VAE), leveraging already trained encoders and decoders to generate a latent representation of the data. This combination allows you to benefit from the learning capabilities of VAEs, while exploiting the diffusion process to further improve data generation.

- Using an already trained VAE allows to create shallower models and thus they are computational efficient
- The inclusion of latent variables allows the latent diffusion model to better capture the underlying structure of the diffusion process.





QM9 Dataset

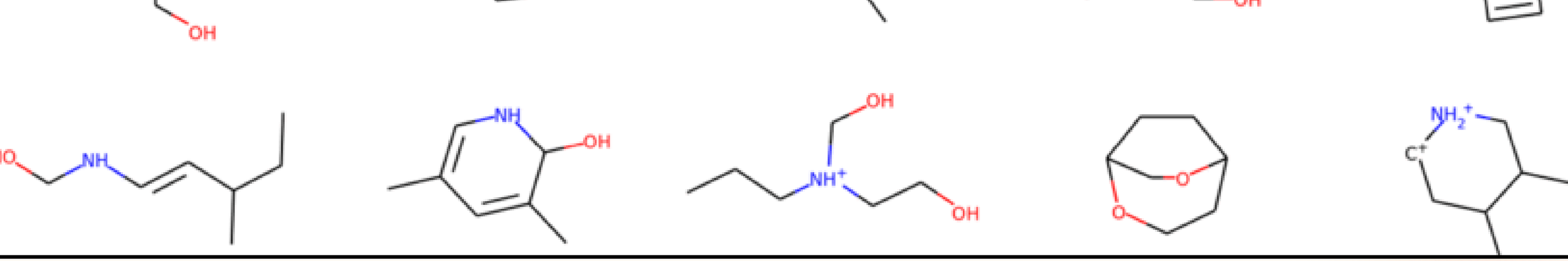
Source: *quantum-machine.org*

Composed of more than 130k organic molecules

Molecules composed of atoms of:
Carbon (**C**), Oxygen (**O**), nitrogen (**N**), hydrogen (**H**), Fluorine (**F**)

Each molecule in the dataset is coded as follows:

- Adjacency matrix
- A Features Vector for each atom (dimension: 11)
  - including: atomic number, position within the molecule (X,Y,Z), energy, etc...
- A Features Vector indicates the type of each bond *(single, double, triple, aromatic)*
- **SMILES** representation of the molecule
- 19 labels (used for classification/regression)



Preprocessing Data

Processing was applied to the dataset to make it manageable by the network model

- **One-hot encoding** of node and edge features
- Removal of molecules made up of a single C, N, O, F atom
- *Removal of H atoms in each molecule*
- Padding of adjacency matrices of molecules with less than 9 atoms

Training methodology

Decoder Quality Metrics. The quality of a conditional decoder can be evaluated by the validity and variety of generated graphs. For a given label $\mathbf{y}^{(l)}$, we draw $n_s = 10^4$ samples $\mathbf{z}^{(l,s)} \sim p(\mathbf{z})$ and compute the discrete point estimate of their decodings $\hat{G}^{(l,s)} = \arg\max p_\theta(G|\mathbf{z}^{(l,s)}, \mathbf{y}^{(l)})$.

Let $V^{(l)}$ be the list of chemically valid molecules from $\hat{G}^{(l,s)}$ and $C^{(l)}$ be the list of chemically valid molecules with atom histograms equal to $\mathbf{y}^{(l)}$. We are interested in ratios $\text{Valid}^{(l)} = |V^{(l)}|/n_s$ and $\text{Accurate}^{(l)} = |C^{(l)}|/n_s$. Furthermore, let $\text{Unique}^{(l)} = |\text{set}(C^{(l)})|/|C^{(l)}|$ be the fraction of unique correct graphs and $\text{Novel}^{(l)} = 1 - |\text{set}(C^{(l)}) \cap \text{QM9}|/|\text{set}(C^{(l)})|$ the fraction of novel out-of-dataset graphs; we define $\text{Unique}^{(l)} = 0$ and $\text{Novel}^{(l)} = 0$

Source: arXiv:1802.03480v

Procedure

- 4 different training sessions based on the number of examples (*70% for training, 30% for validation*) :
  - 100
  - 1'000
  - 5'000
  - 10'000

- For each **training**:
  - Training GraphVae for **8 epochs**
  - Save the GraphVae weights (for encoder and decoder)
  - Training Latent Diffusion (*the number of examples was an order of magnitude larger than for GraphVae*) for **150 epochs**

- For each model trained make a **test** on the same dataset used for training and calculate **Uniqueness**, **Validity** and **Novelty**
- **Grid search** to find the ideal thresholds to maximize **Validity** for each model

*Keep in mind that the calculation of the metrics is done by transforming the graph into **smiles**, so possible simplifications have been made*

Graph VAE architecture

Encoder Input Dimension	• **9X17** (Matrix Nodes features)
Latent Space Dimension	• **2 Vector with : 9 elements** (*mean*) , **9 elements** (*variance*)
Decoder Output Dimension	• **9X9** (Adjacent Matrix) , **9X17** (Matrix Nodes features) , **9X4** (Matrix Edges features)
Architecture Latent Space	• MLP

Training VAE - 1

To train the GraphVAE, 4 different losses were used :

- ELBO:
  - *Adjacent Reconstruction Loss* (using **BCE** between the true Adjacent matrix and the predicted one)
  - *KL divergence* (to consider the mean and variance of the *encoder*)

- Edge Loss (using **MSE** between the Edges Features Matrices)

- Node Loss (using **MSE** between the Nodes Features Matrices)

Training VAE - 2

- ELBO (Adjacent Reconstruction & KL divergence)

In the probabilistic setting of a VAE, the encoder is defined by a variational posterior $q_\phi(\mathbf{z}|G)$ and the decoder by a generative distribution $p_\theta(G|\mathbf{z})$, where ϕ and θ are learned parameters. Furthermore, there is a prior distribution $p(\mathbf{z})$ imposed on the latent code representation as a regularization; we use a simplistic isotropic Gaussian prior $p(\mathbf{z}) = N(0, I)$.

ELBO (Evidence Lower Bond)

$$\mathcal{L}(\phi, \theta; G) = \\ = \mathbb{E}_{q_\phi(\mathbf{z}|G)}[-\log p_\theta(G|\mathbf{z})] + \mathrm{KL}[q_\phi(\mathbf{z}|G)||p(\mathbf{z})]$$

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

Adjacent Reconstruction (BCE)

Enforces high similarity of sampled generated graphs to the input graph G.

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i) \log_2 \left(\frac{P(i)}{Q(i)}\right).$$

KL Divergence

Regularizes the code space to allow for sampling of z directly from p(z) instead from qφ(z|G) later.

Training VAE – 3

- Edge Loss & Node Loss

Reconstruction Losses

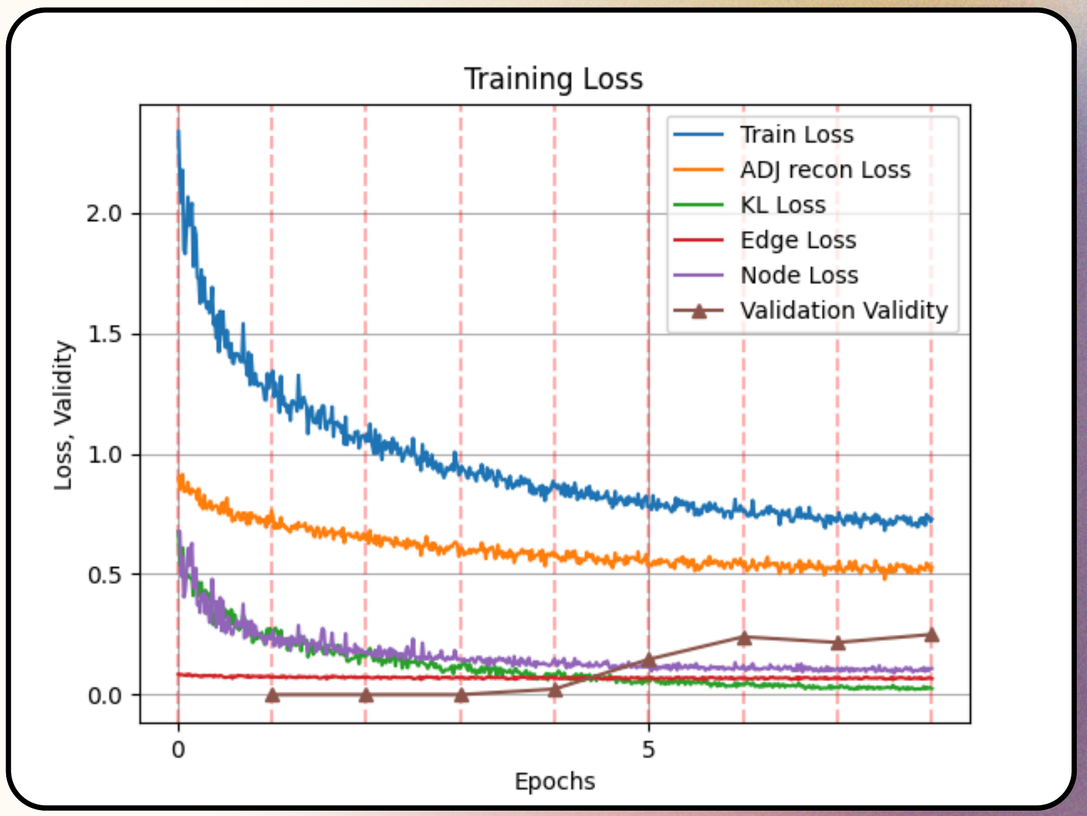
$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Edges Features Matrices

Edges Features Matrices

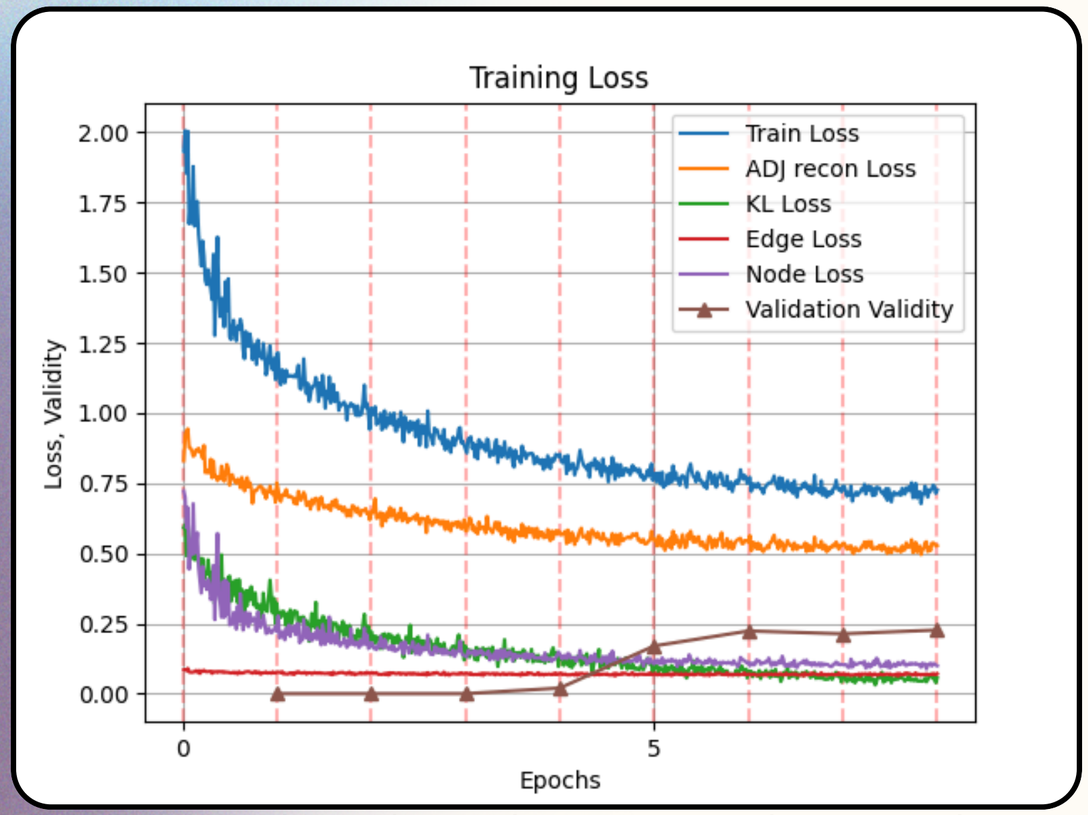
Training VAE - 4

Regularization term : Penalizing the KL divergence regulates the complexity of the latent space, preventing overfittig for the Encoder, by forcing the model to create a smoother latent space



1'000 samples

(v1 on Github)



1'000 samples with Penalization on KL divergence loss

(v2 on Github)

*Adding penalization term on KL term **decrease (slightly) the Train Loss***

Latent Diffusion architecture

Latent Space Input Dimension	• Vector with **9 elements**

UNET Dimension	• **3 Down Channel & 3 Up Channel**
	• From dimension 9:
	  ◦ **7 then 5 then 3** for Down channel (Mirror for the Up Channel)
	  ◦ using **Linear Layer** with **Time embeddig with Dimension 6**

Encoder & Decoder from GraphVAE	• Encoder Input:
	  ◦ **9X17** (Matrix Nodes features)
	• Decoder Output:
	  ◦ **9X9** (Adjacent Matrix) , **9X17** (Matrix Nodes features) , **9X4** (Matrix Edges features)

Training Latent Diffusion – 1

1. The pre-trained GraphVAE architecture is loaded
2. The embedding for each molecule is created
3. The noise is added to the embedding proportionally to a time t, sampled uniformely from a distribution
4. Force the model, with the exclusion of the GraphVAE to discern the noise from the original matrix
5. Calculate the loss (MSE Loss) between the true noise and the predicted one

Repeat

- $\boldsymbol{x}_0 \sim q(\boldsymbol{x_0})$
- $t \sim uniform(\{1, \dots, T\})$
- $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \boldsymbol{I})$
- $\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\nabla_{\boldsymbol{\theta}}\left|\left|\epsilon - \epsilon_\theta(\boldsymbol{x}_t, t)\right|\right|_2^2$

Until convergence

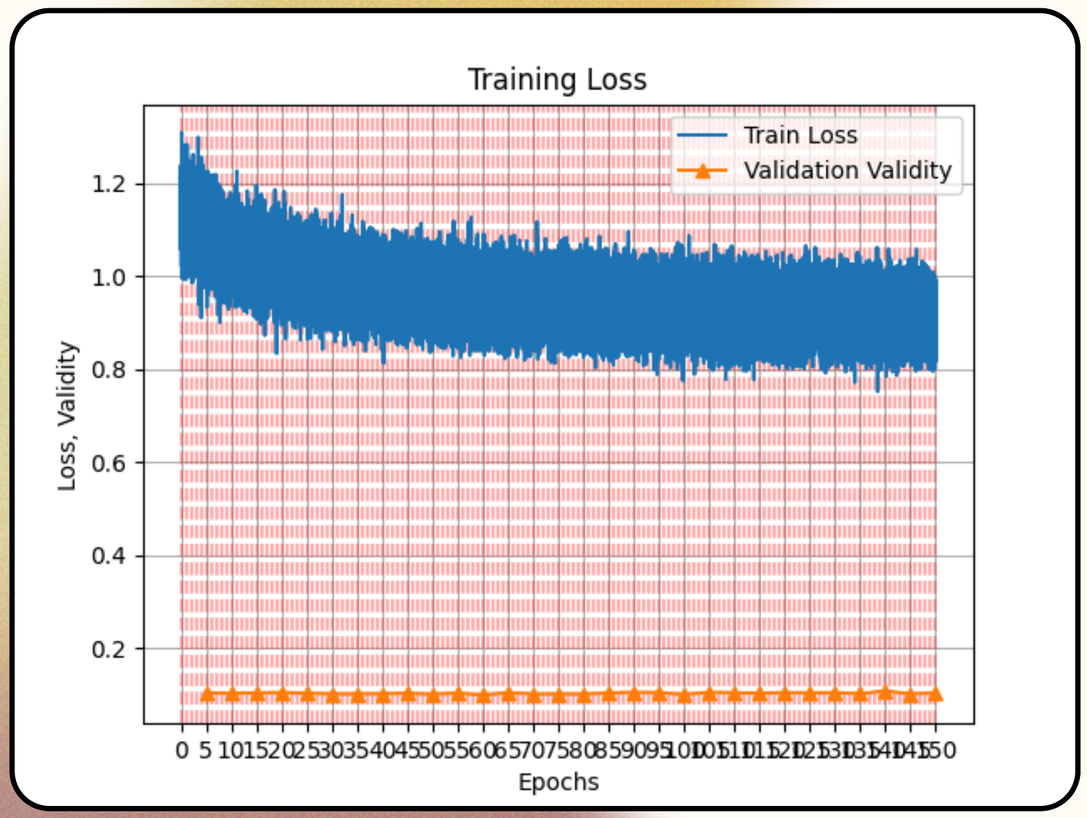
```python
def get_loss(model, x_0, t):
    x_noisy, noise = forward_diffusion_sample(x_0, t, device)
    noise_pred = model(x_noisy, t)
    return F.mse_loss(noise_pred, noise)
```

```python
def forward_diffusion_sample(x_0, t, device="cpu"):
    """
    Takes an image and a timestep as input and
    returns the noisy version of it
    """
    noise = torch.randn_like(x_0)
    sqrt_alphas_cumprod_t = get_index_from_list(sqrt_alphas_cumprod, t, x_0.shape)
    sqrt_one_minus_alphas_cumprod_t = get_index_from_list(sqrt_one_minus_alphas_cumprod, t, x_0.shape)


    return sqrt_alphas_cumprod_t.to(device) * x_0.to(device) + sqrt_one_minus_alphas_cumprod_t.to(
        device
    ) * noise.to(device), noise.to(device)
```

Training Latent Diffusion – 2

10'000 samples



100'000 samples

By increasing the number of samples, **loss gradually decreases**
And the **Validation Validity also decreases**

Results

	Valid	Accurate	Unique	Novel
Ours $c = 20$	0.485	0.485	0.457	0.575
Ours $c = 40$	0.542	0.542	0.618	0.617
Ours $c = 60$	0.517	0.517	0.695	0.570
Ours $c = 80$	0.557	0.557	0.760	0.616

results from the original paper (25 epochs)
(arXiv:1802.03480v)

dataset size	Valid	Unique	Novel
100	0.3	0.144	1.0
1000	0.41	0.0289	1.0
5000	0.421	0.0292	0.98
10000	0.681	0.011	1.0

Best Results GraphVAE (8 epochs)

dataset size	Valid	Unique	Novel
1000 from 100	0.272	0.024	1.0
1000 from 1000	0.431	0.03	1.0
10000 from 1000	0.409	0.0067	1.0
50000 from 5000	0.412	0.0063	0.945
100000 from 10000	0.684	0.0018	0.8968

Best Results Latent Diffusion (8 epochs)

Conclusion – 1

Paper Results VS Our Results

- Increasing the number of examples in our experiment tends to **improve the validity** metric but concurrently **diminishes uniqueness**. This trend may be indicative of a scenario where the model tends to generate a limited set of molecules repeatedly, **leading to a decrease in both uniqueness and novelty**. *Additionally, the likelihood of generating molecules already present in the QM9 dataset increases with the augmentation of training examples*

- In comparison to the models described in the original paper, our models exhibit **significantly lower uniqueness** while managing to achieve **higher validity** with a larger number of training examples

These observations raise concerns regarding the potential **overfitting** of the models, wherein they tend to produce a narrow range of molecules repeatedly. This phenomenon may be particularly pronounced in the case of latent diffusion, where the uniqueness metric experiences a considerable decline.

Conclusion – 2

GraphVAE vs Latent Diffusion

The experiments conducted highlight a significant difference in training between Latent Diffusion and GraphVAE:

- It has been observed that **Latent Diffusion training is significantly faster than GraphVAE:**
  - This can be attributed to the fact that Latent Diffusion operates on small dimensions of the latent space, allowing the number of examples and epochs for training to be increased without running into computational problems.
- During the analysis of the results, it emerged that **the loss stability in Latent Diffusion is significantly lower than that observed in GraphVAE**. This instability persists even after decreasing the learning rate, indicating a challenge in model training convergence.
- However, it is interesting to note that **by increasing the number of examples in the training set, a slight improvement in decreasing the loss was observed**. Furthermore, the observed effect of increasing the number of examples in the training set on Latent Diffusion suggests that greater diversity in the training data could aid model convergence.

It is important to note that despite the lower loss stability in Latent Diffusion, there may be other advantages to using this technique over GraphVAE. For example, *the higher training speed of Latent Diffusion could make this model more suitable for applications where rapid sample generation is required, even at the expense of accuracy compared to GraphVAE*.

Conclusion – 3

GraphVAE vs Latent Diffusion

Despite the speed in training, in terms of metrics (validity and uniqueness mainly) *the results show that* ***Latent Diffusion*** *performs slightly* ***worse*** *than* ***GraphVAE***.

This phenomenon can be caused by several factors:

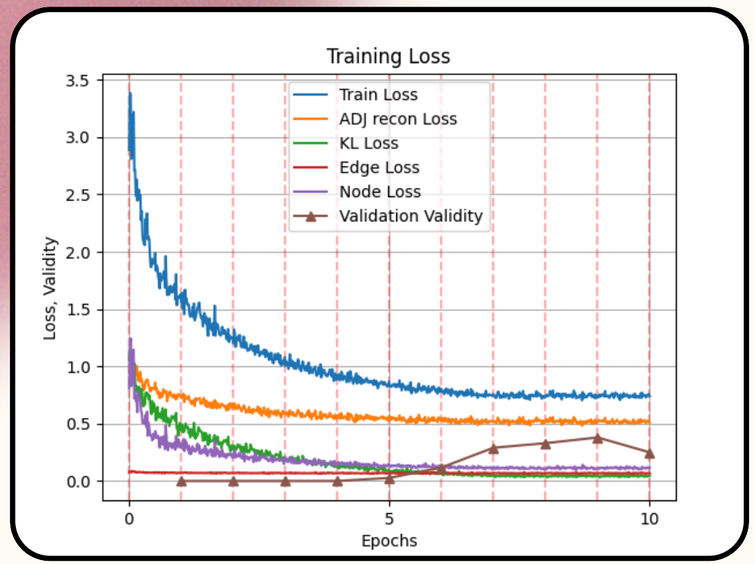
- **Size of the latent space**: A smaller latent space (*9 dimensions*) was used compared to those commonly used in GraphVAE (*20 to 80 dimensions*). This choice may limit Latent Diffusion's ability to capture the complexity of the data, as a smaller latent space may not be expressive enough to accurately represent organic molecules.
- **Amount of data in Diffusion training**: We have increased the size of the Latent Diffusion training dataset compared to GraphVAE by a factor of 10. However, it could be that this increased size is not optimal and may negatively affect the performance of the model (***possible overfitting?***).
- **Incorrect hyperparameters**: It may be that some hyperparameters used in training Latent Diffusion have not been optimized properly, affecting its performance compared to GraphVAE.

Further Development

- The fine-tuning of the hyperparameter for the KL loss is crucial for the correct generalisation capabilities, so it has to be fine-tuned based on the case study

- **Increase computer power**:
  - Consider a wider training set
  - Construct more complicate neural network, (using spatial information: CNN, GNN, Transofmer, etc...)
  - Combining the aforementioned points we can exploit the pros coming from the union of a latent diffusion with VAE. With our limited resources the biggest VAE is still capable to create unique molecule albeit a low degree of validity.

- Find a better strategy for search tresholds to apply to the adjacency matrix to generate molecules then setting a fixed threshold, although it is efficient, in future it might be useful to use other strategies (*employing a Neural Network for example)*



Training Loss on a model with a latent dimension = 20

Latent Dim	dataset size	Valid	Unique	Novel
9	1000	0.41	0.0289	1.0
20	1000	0.5928	0.014	1.0

Clarifications

- **The results obtained may be underestimated compared to the actual performance.** This discrepancy arises because, during the testing phase, certain molecules were not considered valid due to issues in sanitizing the generated molecules, which is not entirely reliable in the RDKIT library.
- **The comparison between the generated molecules and the real ones is not entirely accurate**. This limitation stems from the fact that the comparison is based solely on the SMILES string representation of molecules. In reality, a single molecule can be represented by multiple valid SMILES strings, leading to potential discrepancies in the evaluation process.
- **The selection of thresholds for each model using grid search was carried out in a somewhat arbitrary manner**. The choice of threshold values was based on maximizing the evaluation metrics without necessarily considering the broader context or implications.
- The evaluation of metrics was based on the original SMILES of the QM9 dataset used for training. Consequently, **the evaluation only reflects the performance of the model on a subset of the dataset used for training**, rather than the entire dataset. This approach may not fully capture the model's ability to generalize to unseen data or account for potential biases in the training set (*for example if I trained the model with 100 examples of the dataset then the test was done by comparing the 100 SMILES taken for training and not the entire dataset*)

Fine

Grazie per l'attenzione!

The code, models and all results are available on **Github**