University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

A diffusion generative model on atom distances for de novo molecular design

Supervisor
Prof. Sperduti Alessandro
Department of Mathematics

Co-supervisors
Rigoni Davide
Department of Pharmaceutical and Pharmacological Sciences

Cognolato Samuel
Department of Mathematics
Fondazione Bruno Kessler

Master Candidate
Marco Ballarini

Student ID
2096997

Academic Year
2023-2024

Abstract

The use of artificial intelligence is revolutionising almost every aspect of our lives; among the most impactful advances in AI, generative models are getting days by days more important thanks to their abilities in generating text, video, images and many other data. In recent years, following this trend, there has been a growing interest in development of generative models focused on de novo design of molecules for drug or material discovery. Many models available in literature nowadays consider 3D information, in particular the position in the three-dimensional space. This can pose a problem for learning architectures which do not incorporate rotational invariance. In this thesis an attempt is made to consider the information about the arrangement of the molecule in the space, overcoming problems related to coordinates in 3D space, by considering the distance matrix between between each atom. This allows to exploit information about the relationship of each atom in the space and overcome the problem related to the rotation of the molecule .This is done by considering a diffusion model that generates atoms, bonds and distance between each atom.

Contents

Listing of figures

Listing of tables

Listing of acronyms

SMILES . . . . . Simplified Molecular Input Line Entry System [2, 3]

DiGress . . . . . Discrete Denoising Diffusion For Graph Generation [4]

MiDI . . . . . Mixed Graph + 3D denoising diffusion [5]

EGNN . . . . . Equivariant Graph Neural Network [6]

rEGNN . . . . . relaxed Equivariant Graph Neural Network [5]

MLP . . . . . Multi Layer Perceptron

FiLM . . . . . Feature-wIse Linear Modulation [7]

QM9 . . . . . Quantum Mechanics 9 [8]

GDB13 . . . . . Generated Database 13 [9]

GDB17 . . . . . Generated Database 17 [10]

WLN . . . . . Wiswesser Line Notation [11]

MDS . . . . . Multidimensional Scaling [12]

PNA . . . . . Principal Neighbour Aggregator [13]

SE(3) . . . . . Special Euclidean in 3D group

C . . . . . Carbon

O . . . . . Oxygen

N . . . . . Nitrogen

F . . . . . Fluorine

S . . . . . Solfur

Cl . . . . . Chlorine

1 Introduction

1.1 De novo molecular design

De novo molecular design refers to the process of creating new chemical molecules with specific properties. This process is crucial for various applications, including pharmaceuticals and material science. In particular, considering the vastness of chemical space, there are potentially millions of molecules that could be created, but only a small subset of those are chemically possible and with favourable properties: so it's fundamental to have efficient methods that helps in this task. Traditional in vitro methods can be slow and costly, requiring extensive experimental validation with a try-and-error approach. The challenge is to develop an in silico process capable of generating a set of candidate molecules, guided by predefined rules, that can serve as a warm start for subsequent in vitro experiments.

1.2 Generative Models for molecular generation

In recent years there has been a growing interest in developing computer based methods focused on de novo design of molecules for drug or material discovery and, particularly, the application of generative models to this field have been grown over the last years thanks to technological advancements and the dedicated efforts of the scientific community. These models allow researchers to explore chemical space more efficiently and quickly compared to traditional

methods, which involve screening millions of compounds through a trial-and-error approach. By exploring better the chemical space it's possible to generate hundreds of molecules with specific pharmacological profiles that have not been synthetized before and also optimize some molecular properties that could enhance the molecule's performance. In particular pharmaceutical research could benefit a lot from these methodologies, firstly by reducing the time and costs to develop and produce new drugs, but also by considering future methods able to generate new medicine personalized for individual patients basing on their therapy or genetic profile. The use of Artificial Intelligence (AI) is transforming nearly every aspect of our lives. Among the most significant breakthroughs in AI, generative models are becoming increasingly important. These models are capable of creating a wide range of content, including text, images, videos, and more. This is allowed by technical advance along with theoretical developments, which permit to face complex problems with an huge computational load. In particular, neural networks have represented a game changer inside pattern recognition task, mainly due to their ability in exploiting complex pattern inside data.

Generative models are a class of machine learning algorithms that aims to learn the unknown underlying probability distribution that rules a phenomenon, with the main purpose of being able to sample from it. This is done using a learning architecture that is trained with examples collected from the analyzed phenomenon. The use of deep neural network is particularly suitable for these types of models, since, they can effectively approximate extremely complex distributions thanks to their generalization capabilities. Furthermore, they can be sampled from, allowing the generation of new data not present in the training set. Currently there exist several frameworks able to implement a generative model, ranging from variational autoencoder [14] and Generative Adversarial Networks [15] moving to Flow based methods [16] and diffusion models [17, 18].

1.3 Notable Works

Over the last years the literature about generative models has grown significantly, and on the same level also their application to molecular generation has expanded. In particular many different models have been designed and adapted to molecular generation task.

One of the first of this kind dates back dates 2018 and was done in [19]. This approach is based on variational autoencoder, a particular architecture based on an encoder-decoder structure, that aims to approximate the unkwnown data distribution by mapping a discrete representation of the molecule to a latent continuous space and then, from this, rebuild the original

one. Different points in the latent space encode molecules with different chemical properties, so exploring such a space and sampling from it, allows to generate new molecules with specific desired properties. This was the main focus of this work, and not that of that of generating a large number of chemically valid molecules. Moreover, the general architecture is based on a string representation, the SMILE, which doesn't contain all the structural information that are necessary to generate chemically valid molecules. Despite this criticality, this work represents a milestone in the field of generative models for chemistry and a starting point for many future works in this sector.

The work of [20] follows the same line of the previous one but with shifts the aim to improving chemical validity of generated molecules. In order to accomplish this, authors have studied a way of generating molecules by assembling smaller structure inside a larger one; this approach led to good results in terms of chemical validity of generated molecules, since it allows to expand each generated compound while keeping it chemical valid.

Another successful work is [21]. The authors represent molecules as graphs, increasing the expressiveness of the input, allowing the extraction of topological information inside the model. The generative approach is conditional: first the model generates bonds (edges) and then atoms (nodes), finally they are combined into the final molecule. The main architecture of this model is the normalizing flow generative model, which learns an invertible transformation from the data distribution to a simpler, tractable distribution, with the final aim of approximating the former by sampling from the latter.

Given the huge success of large language models in many fields, researchers are trying to apply these models also to the generation of new molecules. A notable example of this approach is [22], which uses large-scale molecular language models, trained on SMILES data, to generate molecules with specific chemical properties.

Recently there has been a huge clamour around diffusion model. This is motivated by the fact that, for image generation, these models achieve astonishing performances, so researchers worked to adapt them also to molecular generation task. In [23] the main idea is to apply directly the theory at the base of diffusion models for images by modelling each atom of the molecule as a particle in the three dimensional space, ensuring to create valid and realistic conformations. A different approach is taken by authors of [4], in which a discrete denoising diffusion models is trained on molecular graphs. Here atoms and bonds are considered as classes and the diffusion is performed in the discrete space. A mixed discete-continuous approach is considered in [5], in which a diffusion models is built to deal with both graph discrete features and continuous structural information.

1.4 Goals and objectives

The main objective of this thesis is to study the impact of enriching the graph representation of a molecule with the distance between atoms, with the aim of improving chemical validity. From an intuitive point of view the injection of information about the relation in the space between atoms should be beneficial in terms of new molecules generation. Chemical compounds are complex object compose of atoms that are arranged in the three dimensional space; the spatial conformation indeed plays a crucial role in determining physical and chemical properties. In [5] and [23] these information are provided as atom's coordinates in 3D space, but this brings the problem to another level: positions are not uniquely determined. Intuitively, any rotation of 3D coordinates of a molecule should correspond to the same exact molecule. This problem is faced by developing learning architecture that are equivariant respect to rotation, making the output of the model as robust as possible to these changes.

In this thesis there is the attempt of using the distance matrix between each atom instead of coordinates in 3D space, circumventing the need of having a rotation equivariant architecture. This approach makes it possible to overcome issues related to rotation, as it allows the model to exploit information that depends solely on the relation between each atom in a molecule, rather than the spatial 3D arrangement of its atoms. In fact distance matrix doesn't change with rotation of coordinates. Moreover it would be possible to exploits the fact that distances between atoms depend on bond type (for example, bond length of triple bonds is smaller compared to single bonds) and use this information to produce more accurate conformer. In fact, also coordinate models deals implicitly with distance features, but exploiting this explicitly in the model could improve the flow of information inside the learning architecture and helps to produce better results. This thesis proposes a diffusion based generative model to create chemically valid molecules using also the information provided by atomic distance. Moreover this is the starting point for a future conditional diffusion model that generates distance features conditioned to bond information; In this case, the conditioning is applied during the diffusion process, unlike the proposed model, where no explicit conditioning is incorporated.

It's also important, before starting the proper theoretical part, fix the notation that will be used in this thesis: vector are indicated in lowercase, $x \in \mathbb{R}^n$, matrix in upper case $X \in \mathbb{R}^{n \times n}$ and tensors in bold upper case $\mathbf{X} \in \mathbb{R}^{n \times n \times n}$ This thesis will be structured as follows: chapter 2 is focused on the background concepts on which the proposed model will be based; chapter 3 contains the state of the art model that will be used as a benchmark inside this work; chapter 4 contains the explanation of the model proposed in this work. Finally in chapter 5 proposed

model's results are discussed and compared with state of the art ones. The appendix 7 includes the proof related to diffusion denoising probabilistic models loss

2

Background

This chapter will fix some prior concepts that are essential to fully understand the work of this thesis. The first Section 2.1 will treat how to represent molecules in an informative and complete way, considering two different approaches that will be used further in next chapters. The second Section 2.2 will be about the underlying theory of diffusion model: first considering the original formulation [18], in which, taking inspiration from non-equilibrium statistical physics, authors propose to learn data distribution by gradually adding noise to data samples in the dataset, and learn the backward denoising process through a neural network model. The concepts are in [18] then expanded with new implementation choices in [17], focusing mainly on continuous diffusion for image generation and exploring new implementation choices, fixing some parameter and considering an alternative loss function that is easily manageable.

The third 2.3 and the fourth Sections 2.4 of this chapter are dedicated to exploring the implementation of diffusion models in the discrete case, specifically focusing on graph-structured data. A thorough explanation of discrete diffusion models can be found in the seminal paper [24]. Since in this thesis all the learning architecture will be implemented to take molecular graphs as input, these will be considered inside a discrete diffusion framework, as done in [4] and [5].

Since all the learning models in this thesis are based on graph transformer, the Section 2.5 will treat the reasons behind it and the main implementation detail, the information used to write this section are mainly from [1].

2.1 Represent Chemical information

For any learning method, the ability of represent valuable information in a suitable way is crucial, since performance depends heavily on the ability of chosen data structure to retain and express valuable information, and on the capacity of models to discover fundamental pattern inside these.

Representing chemical molecules in a highly expressive manner is a complex challenge that has led to the development of many methods over the years. Not only must the representation be effective, but it also needs to be both mathematically and computationally efficient; data must be pre-processed in a way that's both informative and accessible for deep learning models, allowing them to extract meaningful patterns from the encoded molecular information.

A comprehensive overview of methods for representing chemical information in a computer-processable format can be found in [25]. This section will review the key historical developments that led to the creation of today's most widely used molecular representations, highlighting their main features.

IUPAC (International Union of Pure and Applied Chemistry) provided, between 1919 and 1930 [26], a systematic way of naming chemical compounds, that is still used and updated nowadays, and represents an essential tool for clarity in academic and technical reports, especially for complex molecules. IUPAC nomenclature is essential for human communication in chemistry, but it results extremely hard to process by computers due to its complexity and length. As a matter of fact, during the 1950s, with the rises of the first computer, there was the new necessity of produce or adapt the current notation for the usage in ASCII format. Back in that time the main necessity was just print and typewrite molecules and, eventually, being able to store them in databases. To solve these specific challenges, the Wiswesser Line Notation (WLN) was introduced [11], offering a compact and unambiguous way to represent molecules. As one of the earliest line notations for encoding chemical information, it became the standard for computational processing and data storage on early systems like punched cards. Despite this, with the times, WLN notation has become obsolete, in favour of other modern methods, that are more intuitive, easier to read and, especially, to be processed by machines. In the next subsections two of the main modern method to represent chemical information will be discussed.

2.1.1 SMILES

SMILES (Simplified Molecular Input Line Entry System) is a particular notation that aims to represent a molecular composite as a ASCII string. This notation was originally formulated in 1988 [2, 27, 3]; the main concept is to represent a molecular graph as a string, as a matter of fact atoms and bonds are written as letters, branches as parenthesis and ring closures with matching numbers. In particular, atoms are represented by their atomic symbol, using letters; generally each non-hydrogen atoms is written between square brackets, but for organic subset atoms (including B, C, N, O, P, S, F, Cl, Br and I) it's possible to omit the brackets notations. In the case in which is important to consider the formal charge, the brackets may include not only the letter representing the atom, but also a plus or minus symbol (+/-) and, if necessary, a digit to indicate the magnitude of the charge.

Cycles are represented with numbers, which indicate where the cycle starts and ends.

Regarding bonds, these are generally represented as ASCII symbols like $-$, # and \$ for respectively single, double and triple; generally for single bonds the symbol is omitted and considered for granted. Aromaticity can be represented in multiple ways: either by using the : symbol or by employing lowercase atom symbols to indicate aromaticity. Kekulé form refers to the conversion of aromatic bonds into alternating series of single and double bonds. To better understand this concept, consider benzene as an example: its SMILES representation before kekulization is "c1ccccc1", while after kekulization it becomes "C1=CC=CC=C1".

Another structural characteristic that needs to be take in consideration inside the SMILE notation is the presence of branches: these are descried with parentheses, in particular the first atom within the parentheses group is bounded with the first atom after the parentheses group.

Another important point to consider is that, in its original formulation, there is no one-to-one correspondence between a SMILES string and a molecule. This means that a single molecule could be represented by multiple different SMILES strings. To address this issue, authors of SMILES introduced a canonicalization process in 1989 [27], which ensures that each molecular structure is assigned to unique SMILES representation, eliminating ambiguities.

In the end it's possible to consider SMILES as a real language with its own vocabulary and syntax, but it has no mechanism to ensure that strings are valid from a chemical point of view: it's indeed possible to create a SMILE string that is syntactically correct but semantically incorrect, so ending with a molecule that is not physically possible. This problem impacts heavily on computer generated molecules relying on the SMILES representation, as the algorithm need to be robust against these edge cases.

In recent years some attempt has been done to develop methods that are robust both syntactically and semantically. Notably, in 2020, an article was published [25] proposing a new string-based notation, called SELFIES, that includes mechanisms for verifying the physical validity of the represented molecules. In this work SELFIES will not be used, but it still worth to mention it, since it will play an important role in future molecular generation models.

An important downside of the SMILES notation is that it is linear, so the representation is indeed not fully expressive, since molecules are three dimensional objects.

2.1.2 Molecules as graphs

From a chemical point of view the fundamental part of a molecule is constituted by a set of atoms that are placed in a 3D space and interconnected with different sorts of bond; this way of think a molecule could be immediately converted into a well defined mathematical structure: a graph, defined by nodes and edges. By proceeding on this line it's possible to consider atoms as nodes and bond as edges: in this way the entire molecular object could be treated as a graph.

From a mathematical point of view let $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ be a graph, where $\mathcal{X}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{X} \times \mathcal{X}$ a set of undirected edges. Let $n = |\mathcal{X}|$ be the number of nodes, $m = |\mathcal{E}|$ the number of edges. It's possible to write $X \in \mathbb{R}^{n \times d_v}$ the one hot encoding matrix of atom type and $\mathbf{E} \in \mathbb{R}^{n \times n \times d_e}$ the one hot encoding tensor of edge type for each atom couple, where d_v and d_e is the number of atom and bond classes; in particular each element $x_{i,j} = 1$ means that the i-th atom in the molecule is associated with type j-th, similarly for edge type tensor, $e_{i,j,k} = 1$ means that the bond between the i-th and the j-th atom in the molecule is of type k-th. Since the graph is undirected the edge type tensor E is symmetric with respect the first two dimensions. Moreover let $\mathcal{M} = \mathbb{R}^{n \times d_v} \times \mathbb{R}^{n \times n \times d_e}$ denote the a set molecules where a single molecule is represented as $M = (X, \mathbf{E})$.

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1d_n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nd_n} \end{bmatrix} \qquad x_{i,j} = \begin{cases} 1 & \text{if the i-th atom is of type j} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

$$\mathbf{E} = \begin{bmatrix} \begin{bmatrix} e_{111} & \cdots & e_{11d_e} \\ \cdots & \cdots & \cdots \\ e_{1n1} & \cdots & e_{1nd_e} \end{bmatrix} \\ \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix} \\ \begin{bmatrix} e_{n11} & \cdots & e_{n1d_e} \\ \cdots & \cdots & \cdots \\ e_{nn1} & \cdots & e_{nnd_e} \end{bmatrix} \end{bmatrix} \qquad e_{i,j,k} = \begin{cases} 1 & \text{if the atom i-th is bonded with j-th one} \\ & \text{with k-th bond type} \\ 0 & \text{otherwise} \end{cases}$$

It's possible to include also new features with different characteristic. Graph level features describe global properties and they are represented as $y \in \mathbb{R}$; node level features will be represented by $R \in \mathbb{R}^{n \times d_p}$, where the dimension d_p is the number of node features (in further section this will represents the coordinate in the 3-D space, so $d_p = 3$); the same is valid for edge level features, in this case the feature vectors is represented by a tensor $D \in \mathbb{R}^{n \times n \times d_d}$.

This last part is essential, indeed graph representation of a molecule allows to inject any type of information that is important for a specific task, from chemical information (like formal charges) to geometric information about the arrangement of the molecule in the space (like 3D coordinate or distance between atoms). This is a crucial advantage of molecular graph over SMILES: these, in fact, allow just a linear notation of the molecule information with just some additional feature: the arrangement of the molecule in the space get lost and it's necessary to use ad-hoc packages to recompute it through molecular optimization framework.

On the other hand molecular graph representations have an important issue: this is related to the fact that some molecular structures contain bonds that cannot be represented by simple relations between two atoms; among these types of structure there are polycentric bond, ionic bonds or metal-metal bond.

Molecular graphs are related also to computational problems: it's in fact extremely difficult to deal directly with matrices and tensors, mainly due to memory limitations; it's worth nothing that a molecule with 8 atoms chosen among 4 different types and 4 different bond types would lead to store in memory 32 elements for atom type matrix and 128 elements for bond type tensor. Since both atom type matrix and edge type tensor are built using one hot encoding vectors, to circumvent the issue in larger molecules, some work adopt a sparse representation

of a graph, given by its adjacency list.

2.2 Denoising Diffusion Models

Denoising Diffusion Probabilistic Models (DDPMs) [17] are a type of machine learning generative model inspired by non-equilibrium thermodynamics. Originally introduced in the field of statistical mechanics, diffusion models were developed to estimate complex probability distributions. The approach involves applying a diffusion process to the initial data, gradually transforming it into a known distribution from which it's easy to sample from. Then, starting from this simpler distribution, a denoising architecture is used to learn the reverse process, leading from a simple distribution to the original one.

Diffusion models are build upon three main blocks:

1. The forward process is dedicated to the corruption of original dataset until arriving to a known noise distribution.

2. The reverse process starts from the corrupted data and reverse the noise, arriving to the original denoised data

3. The sampling stage, once that the learning architecture is fully trained, it's possible to sample from the noise distribution and apply the reverse process until arriving to a proper data.

Let's define the real generative data distribution as $x_0 \sim q(x_0)$ where x_0 is an example of the input dataset. The distribution is unknown and the final goal of generative models is to estimate this distribution in order to sample from it. During the forward phase a noise process, with the Markov property, will add noise progressively to the initial representation, creating a sequence of noisy data, that will be denoted as $(x_1, x_2, \cdots, x_T)$. Given the Markovian structure of the noise, the distribution of each sequence is:

$$q(x_1, \cdots, x_T|x_0) = q(x_1|x_0) \prod_{t=2}^{T} q(x_t|x_{t-1}); \tag{2.2}$$

$$q(x_0, \cdots, x_T) = q(x_0) \prod_{t=1}^{T} q(x_t|x_{t-1}). \tag{2.3}$$

The final distribution, when $t \to \infty$, is called steady state distribution, in practice, this will be approximated by using a large, fixed arrival time T. In particular the distribution of the last state should be known and simple since it's necessary to sample from it in order to generate new examples. To ensure this and other nice properties [17] gaussian transitions are chosen for the diffusion process. In particular let's define the noise process as:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I), \tag{2.4}$$

where $\{\beta_t \in (0,1)\}_{t=1}^{T}$ is called variance scheduling and defines the amount of error to inject in the data representation at each diffusion step t. As the sample becomes more noisy, so as t increase, the variance scheduling parameter becomes higher, until arriving near to one. Thanks to the definition of the variance scheduling and the noise trajectory, it's easy to see that $\lim_{t\to\infty} q(x_t|x_{t-1}) = \mathcal{N}(0, I)$: in practice, using the gaussian diffusion it's possible to inject progressively noise inside the initial data representation, until arriving to a standard normal, a distribution that is well known and from which it's easy to sample from during the sampling stage.

The forward process as defined previously, is meant to be run sequentially from $t = 0$ to T. This is quite unpractical during the training phase, in which at each iteration a different time t should be sampled. In fact in [17] authors propose to sample uniformly just one t, allowing to train the model in parallel and approximate the expected value of the loss. Using a simple property of normal distribution is possible to sample x_t at any arbitrary time step t in closed form. Let's define $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_t$, considering $x_t \sim \mathcal{N}(x_t|\sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I)$, it's possible to write:

$$\begin{aligned} x_t &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\varepsilon_{t-1}; && (2.5)\\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\varepsilon}_{t-2}; && (2.6)\\ &= \cdots \\ &= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon; && (2.7) \end{aligned}$$

where, $\bar{\varepsilon}_{t-2}$ merges two gaussian distribution and the final ε is distributed as $\mathcal{N}(0, I)$. Thanks to this notation is possible to jump directly from the initial representation to the noised data at time t.

The reverse process starts from the noised data produced by forward pass and the main idea behind this is to learn how to recover the denoised representation of the data. This is done by learning a parametrized reverse process defined as follows:

$$p_\theta(x_0, \cdots, x_T) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t), \tag{2.8}$$

where $p(x_T)$ is known in advance since, as explained before, it's equal to a standard normal distribution, $p_\theta(x_{t-1}|x_t)$ is the reverse parametrized process that needs to be learnt. A neural network model could parametrize this distribution . Using a Gaussian diffusion kernel in the forward phase, it is possible to also define the functional form of the reverse process, as demonstrated in [28]. Since both a Gaussian transition and its inverse share the same functional form, this allows for the derivation of the reverse dynamics in a mathematically tractable form and so obtain:

$$p_\theta(x_{t-1}|x_t) \sim \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t)). \tag{2.9}$$

The learning architecture will estimate the value of the mean function and variance function of the parametrized reverse function, since the functional form is known but not the parametrization. Indeed, the diffusion is a variational method, that tries to approximate $q(x_{t-1}|x_t, x_0)$ with $p_\theta(x_{t-1}|x_t)$, where $q(x_0)$ is unknown. This is the main reason for the parametrization of the reverse process.

In many applications the variance term is not estimated at all, and considered as constant over all the time steps, since this work is based on the theory of [17], the variance term will be considered fixed; other works, like [29], propose to learn the variance function, this lead to an increase computational burden.

In order to learn the parametrized mean $\mu_\theta(x_t, t)$ it's necessary to write the forward trajectory in reverse form with respect to x_0 (thanks to Markov property the conditioning doesn't have any effect on the results but helps to make the results tractable). Applying the Bayes rule together with Markov property it results:

$$q(x_{t-1}|x_t, x_0) = q(x_t|x_{t-1}, x_0)\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}; \tag{2.10}$$

$$= exp(-\frac{1}{2}(\frac{(x_t - \sqrt{\alpha_t}x_{t-1}^2)^2}{\beta_t}))exp(-\frac{1}{2}(\frac{(x_{t-1}^2 - \sqrt{\bar{\alpha}_{t-1}}x_0)^2}{1-\bar{\alpha}_{t-1}}))exp(-\frac{1}{2}(\frac{(x_t^2 - \sqrt{\bar{\alpha}_t}x_0)^2}{1-\bar{\alpha}_t})); \tag{2.11}$$

$$= exp(-\frac{1}{2}((\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}})x_{t-1}^2 - (\frac{2\sqrt{\alpha_t}}{\beta_t}x_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}x_0)x_{t-1} + C(x_t, x_0))). \tag{2.12}$$

So the reverse conditional probability is tractable since it's possible to write it as a gaussian with a specific mean and variance term. Particularly the mean function depends on the initial data:

$$q(x_{t-1}|x_t, x_0) \sim \mathcal{N}(x_{t-1}; \widetilde{\mu}(x_t, x_0), \sigma_t^2 I), \tag{2.13}$$

where $\sigma_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$ and $\widetilde{\mu}(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}\beta_t}}{1-\bar{\alpha}_t}x_0$.
This formulation is essential in order to train the diffusion model, since the initial data could be written in terms of error injected at any pass by simply rearranging the equation $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon$ in this way $x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\varepsilon_t)$. By plugging inside the riparametrization of input data, the mean formulation is rewritten with respect to the noisy data at time t and the error injected:

$$\widetilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_t). \tag{2.14}$$

In the end the learning architecture will try to predict the amount of error injected at each forward pass, this depends on the current time step t and on the current noised data representation. Since the functional form of the reverse parametrized distribution is gaussian, with variance fixed, it's necessary to learn a parametrized mean function that remove noise inside data at the current time step t, this is represented by

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}}\varepsilon_\theta(x_t, t)). \tag{2.15}$$

The noise predictor network $\varepsilon_\theta(x_t, t)$ is trained to estimate the noise added to each noisy representation, enabling it to denoise the current representation and recover the original one.

Once defined the forward and reverse process, it remains to defines the loss function that will be optimized during the learning phase. The final goal is to learn a set of parameter such that $\mu_\theta(x_t, t) \approx \widetilde{\mu}_t$. This could be done by maximizing the log likelihood criterion associated to the final parametrized distribution of initial data, and by applying Jensen inequality together with importance sampling criterion, the final formula to optimize is:

$$\underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{x_0 \sim q} - \log(p_\theta(x_0)) = \underset{\theta}{\operatorname{argmin}} \int -q(x_0) \log(p_\theta(x_0)) dx_0 \tag{2.16}$$

$$\leq \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{x_0}[D_{KL}(q(x_T|x_0)||p_\theta(x_T)) + \sum_{t=2}^{T} D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) - \log p_\theta(x_0|x_1)], \tag{2.17}$$

where D_{KL} is the Kullback Leibler divergence. The proof of this result could be found in the Appendix 7.

It's easy to see that in gaussian case it's possible to compute each component of the loss function, in particular the first part could be ignored, since the term $p_\theta(x_T)$ is known to be a standard normal; the last component of the formula is considered differently in the literature, in particular in [17] authors consider it as a separate discrete decoder derived from a $\mathcal{N}(x_0; \mu_\theta(x_1, 1), \Sigma_\theta(x_1, 1))$, but in general the treatment of this part depends heavily on the data type in analysis. The fundamental part for the training phase is in the term $\sum_{t=2}^{T} D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$. As pointed out before the learning architecture should predict the noise injected at each time step, since both $q(x_{t-1}|x_t, x_0)$ and $p_\theta(x_{t-1}|x_t)$ are distributed as a gaussian, it's possible to use closed form for the Kullback-Leibler divergence and arrive to the following formulation:

$$D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) = \mathbb{E}_{x_0, \varepsilon}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\alpha_t)||\Sigma_\theta||_2^2}||\varepsilon_t - \varepsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, t)||\right]. \tag{2.18}$$

In [17] the extended formula is shrinked into a more compact one that doesn't consider weighting term and where the variance parameter is fixed to a constant $\sigma_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$. The final formula that will be optimized is :

$$\mathbb{E}_{x_0,\varepsilon}\left[||\varepsilon - \varepsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, t)||\right]. \tag{2.19}$$

All these formulas could be rewritten inside two different crucial algorithms that represents the core of diffusion models. The first algorithm is for train the noise predictor network to recognise and remove the error form a sampled time step.

Algorithm 2.1 Training stage

1: **repeat**
2: $x_0 \sim q(x_0)$
3: $t \sim U(0, T)$
4: $\varepsilon \sim \mathcal{N}(0, 1)$
5: $\nabla_\theta \left\|\varepsilon - \varepsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, t)\right\|$
6: **until** convergence

Algorithm 2.2 Sampling stage

1: $x_T \sim \mathcal{N}(0, 1)$
2: **for** $t = T$ to 0
3: $z \sim \mathcal{N}(0, 1)$
4: $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(x_t, t)) + \sigma_t z$
5: **end for**
6: **return** x_0

The second algorithm is dedicated to the sampling stage: by sampling from a standard normal it's possible to create new examples that comes from the same generative distribution of the input dataset. Particularly the formula $\frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(x_t, t)) + \sigma_t z$ is derived from the fact that given a standard normal distribution it's possible to rewrite a general normal distribution in terms of mean and standard deviation, in such form $X = \mu + \sigma Z$, where $X \sim \mathcal{N}(\mu, \sigma)$ and $Z \sim \mathcal{N}(0, 1)$.

It's important to consider also the parametrization of α_t over all the diffusion time steps: in [17] it is fixed to a constant equal to 10^{-4} when $t = 1$ and to 0.02 when $t = T$ so for the last steps of the diffusion. In [29] authors have proposed several improvements to the continuous diffusion model, including the use of a cosine-based variance schedule. In particular the choice of this scheduling function provide a non-linear drop during the training process, the formulation with respect to α is:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)} \qquad f(t) = \cos(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2})^2, \tag{2.20}$$

where s is a small offset to avoid values becomes too small near to $t = 0$. It's also possible to write this variance scheduling formulation with respect to β_t as seen previously, in this case it's appropriate to clip β value to be no larger than 0.999, in order to avoid any singularities when the diffusion is approaching the end.

In the end this is the mathematical background at the base of diffusion models, as treated in [18], and with some focuses on the implementation choices taken by authors of [17]; these two works have laid the basis for many works about diffusion models that are still being publishing in these times. Recently there is a florid literature that aims to solve diffusion models problem using stochastic differential equations (SDE) and score based methods; one of the most notable works about is [30]. This thesis will not treat these type of diffusion models, but they will certainly represent an interesting area for future development of molecular generative models, especially [31] proposes a graph diffusion process that models the joint distribution of nodes and edges with SDE

2.3 Discrete Denoising Diffusion Probabilistic Models

In Section 2.2, the fundamental theory and concepts underlying the diffusion model were reviewed, in this section there will be the focus on the application of diffusion models for treating discrete dataset.

The main idea behind discrete diffusion process is the same as the ones seen in previous section, what changes is the mathematical formulation used to add noise inside the graph representation, which is built on discrete classes [24].

Given the original discrete data input $x_0 \sim q(x_0)$ represented as a one hot encoding vector with respect to some discrete classes, let's call the sequence of noised input over time step as $(x_1, \cdots, x_T)$; the forward pass works similarly to the continuous case: as before the structure of the forward pass is Markovian and we are mainly interested in the distribution as $t \to \infty$, so $q(x_0)$ and $q(x_T)$, where T is the final time step. Since we are in a discrete environment, it's not practical to use the gaussian diffusion kernel, defined in the section before. A simpler approach would be to use transition matrices derived from Markov chain theory. This matrix will be called $Q \in \mathbb{R}^{n \times n}$ and the value $q_{i,j} \in Q$ will indicates the probability of move from i-th class to the j-th. Following this idea, it's possible to express the forward pass as follows:

$$q(x_t|x_{t-1}) = x_{t-1}Q_t, \tag{2.21}$$

where x_t represents a one-hot encoding vector for the discrete class, Q_t is the transition probability matrix at time t and the final distribution for $q(x_t|x_{t-1})$ is categorical.

This setting allows to move sequentially from a class to another with a certain probability; as before it's convenient to be able to make several forward passes with just one computation. This could be done by multiplying several transition matrices of multiple times together:

$$q(x_t|x_0) = x_0Q_1Q_2 \cdots Q_t = x_0\bar{Q}_t, \tag{2.22}$$

where $\bar{Q}_t = Q_1Q_2 \cdots Q_t$.

As explained in Chapter 2, the key objective of the forward process is to reach a limiting distribution that is simple and easy to treat; in continuous case this is done considering a gaussian noise, in discrete case the noising process is ruled by a transition matrix Q_t. To build this kind of matrix it's necessary to asses that each row sum to one, to conserve the probability distribution, and that $\bar{Q}_t$ converge to a known stationary distribution, as t increases. In literature many different methods has been proposed to design the transition matrix [24], since the structure is heavily task dependent a larger discussion will be considered in Section 2.4.

It's possible also to write the posterior distribution of the forward process. The reason for retrieving the reverse process is the same as before: it will be necessary during the learning phase to have a supervision. Also in this case the derivation is done by applying the Bayes theorem together with Markov property:

$$q(x_{t-1}|x_t, x_0) = q(x_t|x_{t-1}, x_0)\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)} = \frac{x_tQ_t^T \odot x_0\bar{Q}_{t-1}}{x_0\bar{Q}_tx_t^T}. \tag{2.23}$$

Using Markov property it's possible to write $q(x_{t-1}|x_t, x_0) = q(x_{t-1}|x_t)$. The parametrization of the reverse process is done according to what's written in Section 2.2, focusing on a learning architecture, called $\varphi_\theta(\cdot)$, that predicts the logits of the distribution $p_\theta(x_0|x_t)$. By combining with the previous formula, it's possible to write:

$$p_\theta(x_{t-1}|x_t) = \sum_{\tilde{x_0}} q(x_{t-1}|x_t, \tilde{x}_0) p_\theta(\tilde{x_0}|x_t), \tag{2.24}$$

where the term $q(x_t|x_0)$ is discarded since it doesn't depends on x_{t-1} and the summation is done over all the one hot representation of x_0.

All the implementations details regarding variance scheduling parameter could be considered in the same way as done in Section 2.2, similarly for the loss specification: in this case the general theory explained in Section 2.2 is still valid and could be specified considering the Kullback Leibler divergence for discrete case.

2.4 Discrete Denoising Diffusion Probabilistic Model for graphs

In Section 2.3, we have worked with a single one-hot vector representing a discrete class, however, in the case of molecular graphs, the amount of input data increases. Let's establish the notation that will be used for discrete diffusion models applied to molecular graphs. Let's define a molecule, in the same way as expressed in 2.1, $M = (X, \mathbf{E})$ where the term X will be used to define a one hot encoding on the type of node (so in our case the type of atom), and $\mathbf{E}$ to define the one hot encoding tensor for the type of bond. The node type matrix is so defined as $X \in \mathbb{R}^{n \times d_x}$, where d_x are number of atom types and n the number of atoms in a specific molecule; the edge type tensor is defined as $\mathbf{E} \in \mathbb{R}^{n \times n \times d_e}$, where d_e is the number of bond class. The general elements x_i of X and $e_{i,j}$ of $\mathbf{E}$ represent respectively a one hot vector for a single atom and bond class.

Given the preceding notation for molecular graph it's possible to define two transition matrices, as done in [4]: one for bond and the other for atom types:

$$[Q_X^t]_{ij} = q(x^t = j | x^{t-1} = i); \tag{2.25}$$

$$[Q_E^t]_{ij} = q(e^t = j | e^{t-1} = i); \tag{2.26}$$

where the single elements of $[Q_X^t]_{ij}$ indicate the probability of passing at time step t from the class i to the class j, and similarly for $[Q_E^t]_{ij}$.

It's also possible to think the forward process as constituted of different categorical distribution such that $q(x_t|x_{t-1}) \sim Cat(x_{t-1}Q_t)$.

The diffusion step in this case is done on the entire graph, so keeping in consideration the pair node and edge; the diffusion is defined by categorical distribution over edge and node types:

$$q(G_t|G_{t-1}) = (X^{t-1}Q_X^t, \mathbf{E}^{t-1}Q_E^t). \tag{2.27}$$

Considering the fact that molecular graphs are undirected, $\mathbf{E}$ is symmetric with respect the first two dimensions.

Once that the diffusion process is defined, it misses to consider how to design the transition matrix in order to produce the proper diffusion. The key point to keep in consideration is the fact that the limit distribution need to be known and easy to treat in sampling stage. The first ideas of authors in [4] is to design the transition matrix in order to arrive to a uniform distribution over all the classes as time steps increases. In this case he formulation for the transition matrix is the following:

$$Q_X^t = \alpha_t I + (1 - \alpha_t)\frac{1_X 1_X^T}{d_x}; \tag{2.28}$$

$$Q_E^t = \alpha_t I + (1 - \alpha_t)\frac{1_E 1_E^T}{d_e}; \tag{2.29}$$

where 1_X is a column vector of ones with the length of atom classes, the same for 1_E and $\{\alpha_t\}_{t=1}^T$ value is the variance scheduling parameter (in this case it's fixed for both edge and nodes values, but it can also be different), which increases with the time until arriving near to 0 when $t = T$; all the parametrization of variance scheduling are the same as the ones already discussed. This kind of transition matrix lead to a limit distribution $q(X_T)$ and $q(\mathbf{E}_T)$ that is uniform over all the classes. This approach is simple to understand and to implement practically. Other methods are reported in [24], like the use of absorbing states or a discretized version of gaussian distribution, and they are applied to molecular generation inside [32].

As in the previous section, once that the forward pass is defined it remains only to work on reverse process. All this part will treat directly the case in which the reverse process is done on a molecular graph. Following directly what written in section about continuous diffusion models, it's necessary to define a parametrized reverse process able to denoise the input repre-

sentation at a specific time step, to do so with graph it's possible to write:

$$p_\theta(G^{t-1}|G^t) = \prod_{i=1,\dots,n} p_\theta(x_i^{t-1}|G^t) \prod_{i,j=1,\dots,n} p_\theta(e_{i,j}^{t-1}|G^t). \tag{2.30}$$

The final parametrized reverse process is defined by the product of the node and edge reverse process over all the elements. The basic assumption of this formulation is that the reverse process is independent for both nodes and edges. This assumption is strong, but it motivated by looking at diffusion models for images, in which the noise is applied independently on each pixel [17, 24]

It's possible to marginalize the reverse parametrized transition with respect the distribution of all the classes:

$$\begin{aligned} p_\theta(x_i^{t-1}|G^t) = p_\theta(x_i^{t-1}|x_i^t) &= \sum_{x\in \mathrm{X}} p_\theta(x_i^{t-1}|x_i = x, G^t)\hat{p}_\theta^X(x_i|x_t) \qquad (2.31)\\ &= \sum_{x\in \mathrm{X}} q(x_i^{t-1}|x_i = x, x_i^t)\hat{p}_\theta^X(x_i|x_t);\\ p_\theta(e_{i,j}^{t-1}|G^t) = p_\theta(e_{i,j}^{t-1}|e_{i,j}^t) &= \sum_{x\in \mathrm{X}} p_\theta(e_{i,j}^{t-1}|e_{i,j} = e, G^t)\hat{p}_\theta^E(e_{i,j}|e_t) \qquad (2.32)\\ &= \sum_{e\in \mathrm{E}} q(e_{i,j}^{t-1}|e_{i,j} = e, e_{i,j}^t)\hat{p}_\theta^E(e_{i,j}|e_t). \end{aligned}$$

Since it's possible to assume that $p_\theta(x_i^{t-1}|x_i = x, G^t) = q(x_i^{t-1}|x_i = x, x_i^t)$ if $q(x_i^t|x_i = x) > 0$, motivated by the independence between node and edge forward process. In particular, $q(x_i^{t-1}|x_i = x, x_i^t)$ represents the probability of moving from an atom class to another in passing at t step, considering the fact that the original class is fixed, x_t represents the noised class at time t and $\hat{p}_\theta^X(x_i|x_t)$ represents the learning architecture estimate for the probability that atom x_i in the clean graph G_t is of type x. All this holds also for edge tensor case.

This last formulation is extremely important and highlights the main difference with continuous diffusion models: in this case, the learning architecture does not predict the error injected at each time step to subtract it from the sample. Instead, it directly learns the node and edge distribution over the support. Using a neural network, it predicts the clean graph for each noised graph. The neural network architecture $\varphi_\theta(x_t, e_t) = \hat{G}_0$ takes as input the nodes and edges representation in a specific time step, and return the denoised representation of the graph, so original representation given the noised one at time t.

The loss function used in this case is derived from [4]. Authors have specialized the loss explained in Section 2.2, to graph discrete case and obtain the following formulation:

$$l(G, G_t) = \lambda_x CE(X, p_\theta(X_t)) + \lambda_e CE(\mathbf{E}, p_\theta(\mathbf{E}_t)), \tag{2.33}$$

where $CE(\cdot, \cdot)$ is the cross-entropy.

In the case of graph diffusion implemented in [4] and [5] the objective function is indeed a cross entropy over edge and node support. The main goal of the learning architecture is to predict the right class of nodes and edges for each molecule in order to denoise it.

2.5 Graph Transformer

Once that the general theoretical background of diffusion models is established it's necessary to explain the structure of Graph transformer as explained inside [1]. So far, only the fundamental theory behind Denoising Diffusion Probabilistic Models (DDPMs), discrete DDPMs, and graph-based DDPMs has been reviewed, in this chapter there will the explanation of the learning architecture used to learn the parameterized reverse process. To accomplish this, the model proposed in this work uses a graph-based transformer architecture that takes noisy data representations as input and learns to denoise them.

Authors of reference article [1]consider two different architecture: one with just node information and the other with both node and edge information. Since this latter is the one that will be used in the proposed model and in the baseline, the analysis will be done according to this setting.

The main idea behind the graph transformer is to bridge the gap between original Transformer architecture [33] and Graph Neural Network [34] in dealing with graph representation, allowing the self attention mechanism to exploit information from both edge and node. In fact original Transformer deals with relationship inside a sequence by representing all tokens of the input as a fully connected graph, this in order to learn relations between each component. From one hand it could be possible to use a traditional transformer architecture also inside graph structured dataset, but it's worth to notice that for large graphs the computational load would be unfeasible and all the structure information provided by edge connectivity would be lost. To overcome this problem, graph transformers were developed to handle irregular and non-sequential input. The self-attention mechanism is implemented in order to study node-

to-node relationship just between connected ones, making the attention sparse and so less computational demanding. Moreover thanks to these features is possible to exploit both local and global structure information.

The general outline of the graph transformer could be divided in three main blocks: the input block, the transformer block and then the output block. It could be also visualized in Figure 2.1. The input phase acts as a preprocessing stage in which node and edge features are passed through subsequent fully connected layer, in order to produce an initial informative embedding. In this stage there is also a procedure that mimics the positional encoding of original transformer: in fact in order to inject information about structural relationship of nodes inside the graph, one way is to compute the eigenvectors of the Laplacian matrix associated to the adjacency matrix of the molecular graph:

$$L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}, \tag{2.34}$$

where I is the identity matrix, D is the the diagonal matrix filled with zero values outside the diagonal and the elements $d_{i,i}$ represent the number of bond of the i-th atom and A is the adjacency matrix.

In particular, given a graph G with n nodes, let x_i represent the one-hot encoding vector that denotes the class of each atom, and similarly, let $e_{i,j}$ represent the one-hot encoding vector for each edge. It's possible to write the input embedding for each node and edge element through a linear projection:

$$\hat{h}_i^0 = A^0 x_i + a_0; \tag{2.35}$$

$$\hat{e}_{ij}^0 = B^0 e_{ij} + b_0; \tag{2.36}$$

where $A^0 \in \mathbb{R}^{k \times d_x}$ and $B^0 \in \mathbb{R}^{k \times d_e}$ are the parameters of linear projection, a_0 and b_0 are two bias terms, k is the dimension of hidden dimension. These values could be then passed through a non linear activation function, in order to increase their expressiveness. Eventually all the procedure is done multiple times to increase the generalization. Given a laplacian matrix $L \in \mathbb{R}^{n \times n}$, eigenvectors are defined with the factorization $L = \Gamma^T \Lambda \Gamma$, where γ_i is the eigenvector associated to the i-th atom and Λ is the diagonal matrix with eigenvalues. It's possible to compute a linear projection to obtain an embedding of each eigenvector:

$$\gamma_i^0 = C^0\gamma_i + c_0; \tag{2.37}$$

$$h_i^0 = \hat{h}_i^0 + \gamma_i^0; \tag{2.38}$$

where $C^0 \in \mathbb{R}^{k\times n}$ is the coefficient matrix associated with the bias term c_0 that maps each lapalcain vector into the same hidden space with dimension k as the node embedding, γ_i^0 is the linear embedding associated to the laplacian eigenvector and h_i is the augmented embedding with both node and positional information.

Once that the input stage is ended, both the embedding $\hat{x}_i$ and $\hat{e}_{ij}$ are used as input of transformer stage. The graph transformer layer works similarly to the original transformer, in particular the node update equation for a general layer l is:

$$\hat{h}_i^{l+1} = O_h^l \Big\|_{k=1}^{H} \Big(\sum_{j\in\mathcal{N}_i} w_{ij}^{k,l} V^{k,l} h_j^l\Big); \tag{2.39}$$

$$\hat{e}_{ij}^{l+1} = O_e^l \Big\|_{k=1}^{H} (\hat{w}_{ij}^{k,l}); \tag{2.40}$$

$$w_{ij}^{k,l} = \text{softmax}_j(\hat{w}_{i,j}^{k,l}); \tag{2.41}$$

$$\hat{w}_{i,j}^{k,l} = \Big(\frac{Q^{k,l}h_i^l \cdot K^{k,l}h_j^l}{\sqrt{d_k}}\Big) \cdot E^{k,l}\hat{e}_{ij}^l; \tag{2.42}$$

where $Q^{k,l}$, $K^{k,l}$ and $V^{k,l}$ are the three query, keys and values matrix necessary for attention mechanism, the number $h = 1, \ldots, H$ denotes the number of attention heads, the operator $\Big\|$ indicates the concatenation, $O_h^l \in \mathbb{R}^{d\times d}$ and $O_e^l \in \mathbb{R}^{d\times d}$ are the learnable parameters associated to the concatenated embedding.

Then each attention output is passed to a feed forward neural network, followed by a residual connection layer and a normalization layer. The output of transformer layer is then processed with a feed forward network with skip connection and normalization layer:

$$\tilde{h}_i^{l+1} = \text{Norm}(h_i^l + \hat{h}_i^{l+1}); \tag{2.43}$$

$$\tilde{\tilde{h}}_i^{l+1} = W_{h,2}^l \text{ReLu}(W_{h,1}^l \tilde{h}_i^{l+1}); \tag{2.44}$$

$$h_i^{l+1} = \text{Norm}(\tilde{\tilde{h}}_i^{l+1} + \tilde{h}_i^{l+1}); \tag{2.45}$$

where $W^l_{h,1} \in \mathbb{R}^{2d\times d}$ and $W^l_{h,2} \in \mathbb{R}^{d\times 2d}$, regarding $\tilde{\tilde{h}}^{l+1}_i$ and $\tilde{h}^{l+1}_i$ these are the intermediate representation of the node embedding. Regarding edge attention outputs, these are computed in the same way as node ones so:

$$\tilde{e}^{l+1}_{i,j} = \text{Norm}\left(e^l_{i,j} + \hat{e}^{l+1}_{i,j}\right); \tag{2.46}$$

$$\tilde{\tilde{e}}^{l+1}_{i,j} = W^l_{e,2}\,\text{ReLU}\left(W^l_{e,1}\tilde{e}^{l+1}_{i,j}\right); \tag{2.47}$$

$$e^{l+1}_{i,j} = \text{Norm}\left(\tilde{\tilde{e}}^{l+1}_{i,j} + \tilde{e}^{l+1}_{i,j}\right); \tag{2.48}$$

where $W^l_{e,1} \in \mathbb{R}^{2d\times d}$ and $W^l_{e,2} \in \mathbb{R}^{d\times 2d}$, regarding $\tilde{\tilde{e}}^{l+1}_{i,j}$ and $\tilde{e}^{l+1}_{i,j}$ these are the intermediate representation of the edge embeddings.

After all the graph transformer layers, all the attention representation are then passed through a Multi Layer Perceptron (MLP) to obtain a task dependent output for the model.

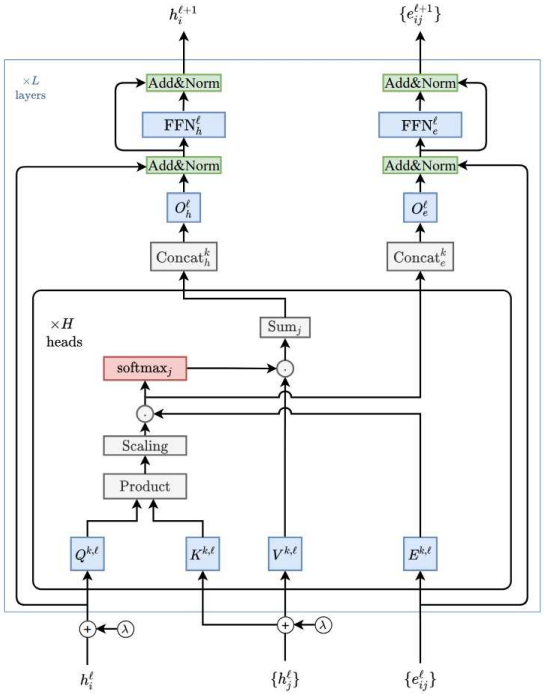


Figure 2.1: Representation of Graph Transformer as in [1]

The explanation of this architecture has been done following the original work [1], which is not totally suitable for generative models, in first stage since it doesn't contain the diffusion time step variable. In the following chapters this architecture will be revised and modified twice:

in Chapter 3, when explaining the benchmark model, and in Chapter 4. when explainin the original model of this work. Despite the modification, the general structure of the architecture remains the same.

3

State of the art

This chapter will introduce and explain the main features behind MiDi model[5], a discrete diffusion model that generates jointly 2D and 3D structure of each molecule; before explaining the main details at the base of MiDi, it's worth to mention some other methods that deals with atomic coordinates in the space. In [6], authors uses a continuous denoising diffusion probabilistic model, explained in Section 2.2 of Chapter 2, to generate both atom type and coordinate. In this approach atom type are treated as continuous and there is no bonding information inside the learning architecture: this is predicted from results using distance between atoms and atom class.

A different approach is used by authors of [35], here they propose a new model called JODO (Joint 2D and 3D Diffusion Model) that simultaneously models both the 2D bonding graphs and 3D geometries of molecules, this is done using a diffusion graph transformer, where nodes, edges and geometric features interact using a relational attention mechanism.

The general theory of MiDI derives directly from what is written in Chapter 2, there are only some changes in order to consider a mixed approach between discrete and continuous features. Let's define a molecule $M = (X, \mathbf{E})$ with the same notation used Chapter 2. In addition to atom and bond information it's possible to consider $C \subseteq \mathbb{R}^{n\times 3}$ the one hot encoding matrix of formal charge class as a node level feature. With this specification the term c_i is the one hot encoding vector associated to the formal charge of each atom. The use of formal charge is crucial since it provides valuable information about the chemical structure of the generated compound. Regarding coordinates, $R \subseteq \mathbb{R}^{n\times 3}$ is the matrix of coordinates, where the single

$r_i \in \mathbb{R}^3$ is the position vector of one atom.

Let's consider a discrete noise process that corrupts edges, atoms and charges independently; for what concern coordinate data, it is used a gaussian noise with the zero center of mass subspace such that $\varepsilon \sim \mathcal{N}_{CoM}(\alpha_t R_{t-1}, \sigma_t^2 I)$. This type of noise is essential in order to obtain a roto-translational equivariant architecture [23] and works by centering the matrix of coordinate with respect the center of mass each time the noise is sampled; this step is crucial because it ensures that the system remains stable and does not deviate significantly from its central position.

In the end the forward process that controls the noise injection inside molecular representations is given by the product of all the noise together:

$$q(G_t|G_{t-1}) \sim \mathcal{N}_{CoM}(\alpha_t R_{t-1}, \sigma_t^2 I) \times \text{Cat}(X^{t-1}Q_E^t) \times \text{Cat}(\mathbf{E}^{t-1}Q_E^t) \times \text{Cat}(C^{t-1}Q_C^t), \quad (3.1)$$

where Q is the transition matrix defined at each diffusion time step for charges, atoms and bonds, the parameter $\{\alpha\}_{t=1}^T$ is the variance scheduling parameter that works as explained in previous chapter; similarly σ_t^2 is the variance term that will be considered fixed as done in Chapter 2.
The reverse process is defined as the product of all the marginal parametrized posteriors:

$$p_\theta(G_{t-1}|G_t) = \prod_i^n p_\theta(r_i^{t-1}|G_t) p_\theta(x_i^{t-1}|G_t) p_\theta(c_i^{t-1}|G_t) \prod_{i,j}^n p_\theta(e_{i,j}^{t-1}|G_t); \quad (3.2)$$

It's possible to calculate each discrete term by marginalizing over the network prediction as explained in the previous chapter:

$$\begin{aligned} p_\theta(x_i^{t-1}|G^t) = p_\theta(x_i^{t-1}|x_i^t, G_t) &= \sum_{x \in \mathrm{X}} p_\theta(x_i^{t-1}|x_i = x, G^t) \hat{p}_\theta^X(x_i|x_t, G_t) \quad (3.3) \\ &= \sum_{x \in \mathrm{X}} q(x_i^{t-1}|x_i = x, x_i^t) \hat{p}_\theta^X(x_i|x_t, G_t); \end{aligned}$$

Regarding positional feature, the diffusion is done in continuous space, so the parametrized reverse process is defined as $p_\theta(R^{t-1}|G^t) = p_\theta(R^{t-1}|R^t, G_t) \sim \mathcal{N}(\mu_\theta(x_t, t, G_t), \Sigma_\theta(x_t, t, G_t))$, as done in Chapter 2. By considering the variance term as fixed to a constant σ_t^2, it's possible to

write the whole expression in terms of predicted error:

$$R_{t-1} = \frac{1}{\sqrt{\alpha_t}}(R_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\phi_\theta(R_t, t, G_t)) + \sigma_t z, \tag{3.4}$$

where $\phi_\theta(\cdot)$ is the predictor network, implemented with a neural network architecture, that predicts the coordinate system starting from the noised one. The learning architecture $\phi_\theta(G_t)$ takes in input a noisy graph G_t and predict the clean one G: this is done by considering the discrete-continuous approach, where for discrete class it returns a categorical distribution over the classes and for continuous feature it return the clean coordinate system.

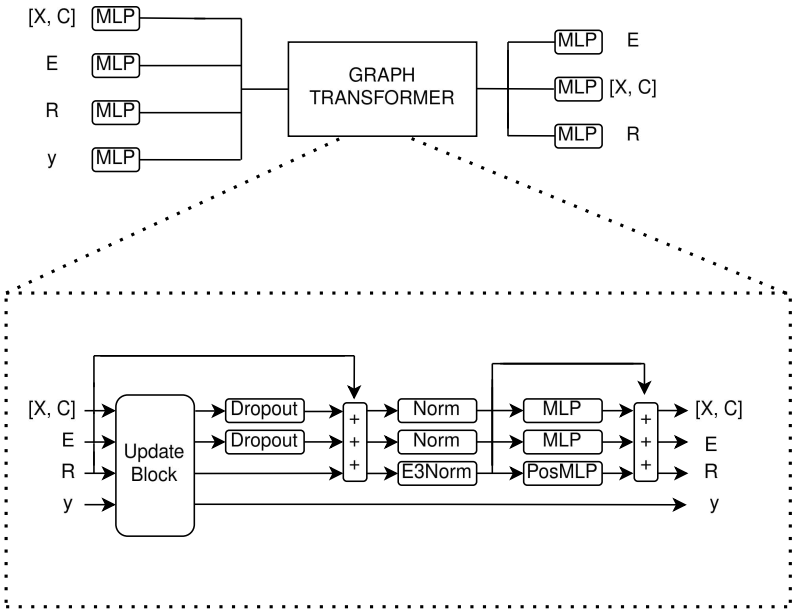


Figure 3.1: MiDi model architecture, at the top, the outline of the overall architecture is visible. Within the dotted lines, the details of the transformer architecture are shown.

The denoising network is implemented using a Graph Transformer, where coordinate features are treated in order to guarantee SE(3) equivariance. SE(3) equivariance refers to the ability of a learning model to be consistent with respect rotation and translation in three dimensional space. It's possible to learn this by augmenting the dataset with multiple transformation, this lead to a consistent increase in computational burden, since the model is forced to learn by seeing the same molecule multiple times, but with different conformation in the space. To

avoid this in literature many others models have been proposed; [36, 37, 38] manage to solve this problem with spherical harmonics, leading to effective results but still extremely expansive. Other generative models [39, 40] use EGNN (Equivariant Graph Neural Network) layers [6]. In EGNN there is the attempt to learn graph neural networks that are equivariant to rotation, translation, reflections and permutations, by using some proxy variable instead of coordinate directly; in this case, the bias is not learned during training; instead, it is built directly into the model's definition. Coordinates are updated recursively using node, edge and proxy features:

$$r_i = r_i + \sum_j c_{i,j} \zeta(||r_i - r_j||, x_i, x_j, y_{i,j})(r_i - r_j); \tag{3.5}$$

where $c_{i,j}$ represents a normalization term and $\zeta()$ a MLP, the position is updated considering the weighted sum of all relative distances, this in order to ensure the translation and rotation equivariance, as proven in [6]. The key idea behind this method is to provide the learning function with only rotation and translation invariant parameters, such as the distance between atoms. MiDI model take in consideration the SE(3) problem and manage to solve it by considering a modified version of EGNN. In particular, the attention module is based on rotational invariant proxy descriptors such as pairwise distance $||r_i - r_j||$, magnitude of each coordinate $||r_i||$ and the angle between each coordinate $\cos(r_i, r_j)$. This approach is feasible because each molecule's coordinate system is centered on its center of mass, making it translation-equivariant. All these information are concatenated in one tensor:

$$[\Delta_r]_{i,j} = cat(||r_i - r_j||, ||r_i||, ||r_j||, \cos(r_i, r_j)). \tag{3.6}$$

This vector will be considered inside the self-attention mechanism to update node and edge embedding, and, also, to produce the embedding of coordinate data.
All the values reported are to be considered as centred with respect the center of mass, this incorporates information into a coordinate representation that is independent from rotation. Finally the coordinate is updated combining rotation-invariant message function, here called φ_m with node and edge information:

$$r_i = r_i + \sum_j \varphi_m(x_i, x_j, [\Delta_r]_{i,j}, e_{i,j})(r_j - r_i). \tag{3.7}$$

The approach is called rEGNN (relaxed Equivariant Graph Neural Network) [5] and it's im-

plemented inside the self attention module of the graph transformer. An overview of Graph Transforemer used in MiDi could be seen in Figure 3.1 and it's built using the framework explained in the previous Chapter 2, but with some modifications. The self-attention layer is implemented inside an update block that process all the feature simultaneously, it takes as input also the diffusion step t, which includes information about the amount of error injected in molecular representation. The edge features are first updated using positional Δ_r and node information; are then processed independently of attention representation. This is the first difference with respect the original graph transformer architecture [1]. Edge features are computed as follows:

$$e_{i,j} = \varphi_e(e_{i,j}, x_i \odot x_j, [\Delta_r]_{i,j}); \tag{3.8}$$

where φ are particular aggregator functions that helps to inject context information inside edge, node and position embedding as long as self-attention weights. In the case in analysis aggregator functions are represented by affine transformation of current embedding where the context information is passed through a linear layer before augmenting the embedding. Node features are updated with a self-attention mechanism, using global information of diffusion time step t, the positional information of Δ_r and the edge information E already computed. The procedure to obtain the self attention weight for node representation is nearly the same as the one exposed in previous chapter, it changes only how edge and positional representation are injected inside final node embedding. Authors of MiDi have used a PNA (Principal Neighbour Aggregator) [13] to pool pairwise information of (Δ_r, E) inside node representation, this is done for both coordinate and edge representation in the following way: $\text{PNA}(E)_i = W^T \text{Cat}(\text{mean}, \text{min}, \text{max}, \text{std})_j(e_{i,j})$. The aggregator works by concatenating summary statistics (mean, minimum, maximum and standard deviation) of the embedding (in this case, the vector $e_{i,j}$) and then multiply the results with a learnable weight matrix. It's possible to see the update block procedures in these formulas:

$$\alpha_{i,j} = \text{softmax}(\varphi_\alpha(e_{i,j}, W^T_{key} x_i, W^T_{query} x_j, [\Delta_r]_{i,j})), \tag{3.9}$$

$$x_i = \varphi_X(\sum_j \alpha_{i,j} W^T_{val} x_j, \text{PNA}(\mathbf{E})_j, \text{PNA}(\Delta_r)_j, t), \tag{3.10}$$

$$r_i = r_i + \sum_j \varphi_r([\Delta_r]_{i,j}, e_{i,j})(r_j - r_i). \tag{3.11}$$

These formulas represents what's is done inside the update block, as shown in figure 3.1 all computed embedding, except for time step information, are then processed with other layers. For node and edge representation a dropout layer followed by a normalization layer and finally a MLP layer are stacked one over the other, to produce a final rich embedding. Regarding coordinate features, these are passed through a normalisation layer and a MLP layer, which are specifically modification to take in consideration the SE(3) equivariance. The MLP layer is modified as follows:

$$\text{PosMLP}(R) = \psi_{CoM}(\text{MLP}(||R||)\frac{R}{||R|| + \delta}). \tag{3.12}$$

The norm $||R|| \in \mathbb{R}^{n\times 1}$ is the magnitude information of each coordinate vector, which is invariant to rotation. This modification is motivated by the necessity of not treat separately each coordinate, that would cause a break in the SE(3) equivariance, but consider a MLP that process the magnitude of each coordinate. The operator ψ_{CoM} is the projection of the coordinates into a linear subspace with center of mass at zero: $\psi_{CoM}(R)_i = r_i - \frac{1}{n}\sum_{i=1}^{n} r_i$. Also the normalization layer [36] is modified in order to maintain SE(3) equivariance, in particular the formulation is the following:

$$\text{E3Norm}(R) = \gamma\frac{R}{\bar{n} + \delta}, \tag{3.13}$$

where $\bar{n} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} ||r_i||^2}$, γ is a learnable parameter and δ is a small off-set.

In the end the whole procedure consists of an update block, with inside the self-attention mechanism, a dropout, a normalization and a multi-layer perceptron. The output values of each multilayer perceptron are added to the input values from before the MLP in a residual layer. The whole procedure is repeated many times.

Once that the whole learning architecture is defined it's necessary to consider the loss function that will be optimized during the training phase. Since the learning architecture is trained to denoise a corrupted representation of the molecular graph, considering both 2D and positional features, the whole loss function is formed of two main blocks: a cross entropy part, used to train the classification of the right edge, atom and charge class, and a mean square error part, used to solve the regression problem one the coordinates. The final loss is formed by the weighted sum of each component:

$$l(G, G_t) = \lambda_x CE(X, p_\theta(X_t)) + \lambda_e CE(\mathbf{E}, p_\theta(\mathbf{E}_t)) + \lambda_c CE(C, p_\theta(C_t)) + \lambda_{pos}||\hat{R} - R_t||^2, \tag{3.14}$$

where G_t is the noised graph at time t, G is the clean graph ,$CE(\cdot, \cdot)$ stands for cross entropy, $p_\theta(\cdot)$ is the results of the parametrized learning architecture which takes in input a noisy value and return the clean one, this for each feature set; regarding positional features, $\hat{R}$ represents the denoised coordinate matrix, obtained as result of the neural network model. It's possible to notice that this formulation derives directly from what written in Chatper 2. It's possible to encapsulate all the main procedure of MiDi model inside two algorithms:

Algorithm 3.1 MiDi - Training

1: $G = (X, \mathbf{E}, C, R)$
2: Sample $t \sim \mathcal{U}(1, ..., T)$
3: $z \sim \mathcal{N}(0, 1)$
4: $G_t \sim \mathcal{N}_{CoM}(\alpha_t R_{t-1}, \sigma_t^2 I) \times \text{Cat}(XQ_{\mathbf{E}}^t) \times \text{Cat}(\mathbf{E}Q^t) \times \text{Cat}(CQ_C^t)$
5: $\hat{p}_x, \hat{p}_e, \hat{p}_c, \hat{R} \leftarrow f_\theta(G_t)$
6: Optimize.step $l(G, \hat{G}_t) = \lambda_x CE(X, p_\theta(x)) + \lambda_e CE(\mathbf{E}, p_\theta(e)) + \lambda_c CE(C, p_\theta(c)) + \lambda_{pos}||\hat{R} - R||^2$

The learning algorithm follows the same procedure of what explained in previous chapter: sampling a noise graph, perform a forward pass, denoise the structure and finally optimize the loss function.

Values for λ hyperparameters are fixed inside [5] to these values: $\lambda_X = 0.4$, $\lambda_E = 2$, $\lambda_C = 1$ and $\lambda_{pos} = 3$.

In the sampling stage, the initial step involves drawing samples from the noise distribution. This is achieved by independently considering the limiting distribution for each component: a standard normal distribution for positional data and a uniform distribution for discrete data, such as class labels. Next, a denoising process is applied to each component, followed by computing the posterior, which ultimately generates the final graph.

Algorithm 3.2 MiDi - Sampling stage

$G_T \sim \mathcal{N}(0,1) \times q_X(d_x) \times q_E(d_e) \times q_C(d_c)$
for $t = T$ to 0
  $\hat{p}_x, \hat{p}_e, \hat{p}_c, \hat{R}_t = f_\theta(G_t)$
  $z \sim \mathcal{N}(0,1)$
  $R_{t-1} = \frac{1}{\sqrt{\alpha_t}}(R_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\hat{R}_t) + \sigma_t z$
  $p_\theta(x_i^{t-1}|G_t) = \sum_x q(x_i^{t-1}|x_i = x, x_i^t)\hat{p}_i^X(x)$
  $p_\theta(e_{ij}^{t-1}|G_t) = \sum_e q(e_{ij}^{t-1}|e_{ij} = e, e_{ij}^t)\hat{p}_{ij}^E(e)$
  $p_\theta(c_i^{t-1}|G_t) = \sum_c q(c_i^{t-1}|c_i = c, c_i^t)\hat{p}_i^X(x)$
  $(X_{t-1}, C_{t-1}, E_{t-1}) \sim \prod_i p_\theta(x_i^{t-1}|G_t) p_\theta(c_i^{t-1}|G_t) \prod_{i,j} p_\theta(e_{ij}^{t-1}|G_t)$
  $G_{t-1} = (X_{t-1}, \mathbf{E}_{t-1}, C_{t-1}, R_{t-1})$
end for
return G_0

4

Proposed Model

This chapter will explain the main details at the base of the model proposed in this work. The main core of this model is to apply diffusion on atom distance matrix; this approach deletes all issues related to SE(3) equivariance and enriches the model with additional information. Specifically, the direct relationship between distances and bond types can be used to generate more realistic structures.

The general outline of the proposed model is similar to what seen in MiDI case: let's consider a molecule $\mathcal{M} = (X, E)$, where x_i and $e_{i,j}$ have the same interpretation given in Chapter 2. Let's further consider also formal charge information c_i, as done in Chapter 3. It's possible to define the distance matrix related to a molecular graph as $D \in \mathbb{R}^{n \times n}$, where n is the number of atom for each molecule. This type of information is an edge level information that is propagated for all the molecular graphs, the vector $d_i \in \mathbb{R}^n$ represents the distances between the i-th atom and all other atoms within the molecule.

The diffusion process is done with the same steps considered in Chapter 3: during the forward process, a gaussian noise is applied to each distance matrix with the same procedures seen in Chapter 2 for continuous diffusion. As pointed out before, in this case there is no need to consider a gaussian noise with center of mass, but it's worth to use a simple gaussian transition kernel. The forward process is written like in MiDI, so as follows:

$$q(G_t|G_{t-1}) \sim \mathcal{N}(\alpha_t D_{t-1}, \sigma_t^2 I) \times \text{Cat}(X^{t-1}Q_X^t) \times \text{Cat}(\mathbf{E}^{t-1}Q_E^t) \times \text{Cat}(C^{t-1}Q_C^t); \qquad (4.1)$$

Similarly the parametrized reverse process is written as the product of all the marginal posteriors, considering the fact that distance matrix is a edge level feature:

$$p_\theta(G_{t-1}|G_t) = \prod_i^n p_\theta(x_i^{t-1}|G_t)p_\theta(c_i^{t-1}|G_t) \prod_{i,j}^n p_\theta(e_{i,j}^{t-1}|G_t)p_\theta(d_{i,j}^{t-1}|G_t); \tag{4.2}$$

where the distance $d_{i,j}^{t-1}$ is the distance between i-th and j-th atom at time t, all the other elements have the same meaning seen in Section 2.4 of Chapter 2. Each term in computed by marginalizing over network prediction as explained in Section 2.4 of Chapter 2. In the case of distance the the reverse pass is done as in a standard continuous case:

$$D_{t-1} = \frac{1}{\sqrt{\alpha_t}}(D_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\phi_\theta^D(D_{t-1}|D_t, G_t)), \tag{4.3}$$

where $\phi_\theta(\cdot)$ is the learning architecture that takes in input a noisy graph and return the denoised representation. The learning architecture is based on graph transformer seen in Chapter 2, some modification are implemented following what seen in Chapter 3 and other are proposed specifically for this model. It's possible to see at Figure 4.1 a visual explanation of the transformer architecture.

The encoding part is similar to what seen in original graph transformer: an embedding is computed for bond type tensor, distance matrix and atom type matrix (as before all the computations done inside the transformer consider the concatenated version of atom type matrix and formal charge). The main difference with respect MiDi formulation is that, in this case, also the positional embeddings are used. In order to exploits fully the topological information at the base of graph structure, an embeddings of non trivial eigenvector associated to laplacian matrix are produced, these are then summed with the node/charge embedding as seen in the original graph transformer architecture in Chapter 2.

This version of graph transformer is based on several self-attention module, dropout layers, normalization layer and feedforward layers; it could be visualized in Figure 4.1. The architectural choices to implement self-attention stage are expressively designed to encapsulate also external information inside node attention weight. All the modifications done to ensure SE(3) modification are not considered anymore, because of this all the dropout, normalization and feedforward layer are in the usual version. The update block, which contains the self-attention

mechanism, is built using implementation choices from [5] and from [4], and it could be visualized in Figure 4.2. In first stage, the output embedding for edge information are processed using information from distance, node and diffusion time step embedding. In order to incorporate these information inside the edge embedding a FiLM (Feature-wise Linear Modulation layer) [7] is used. This particular method is used for a dynamic modulation of the edge embedding, injecting external information inside it; this is done by considering affine transformation of some external features: let's consider M as a general external features and A as the embedding matrix in which inject external information, the network will learn an affine transformation that will augment values of attention scores in this way:

$$FiLM(A, M) = MW_1 + (MW_2) \odot A + A; \tag{4.4}$$

where W_1 and W_2 are two matrices of learnable parameter associated to affine transformation.

In this case, a single FiLM layer is used to inject global diffusion step information inside edge representation. In the case in analysis, in which it's necessary to consider the edge embedding, let's consider E as the initial edge embedding. Then, the FiLM layer associated to diffusion time step embedding is:

$$\hat{E} = FiLM(E, y) = yW_1^y + (yW_2^y) \odot E + E; \tag{4.5}$$

where W_1^y and W_2^y are learnable weights that produces an affine transformation of edge embedding. The result of this operation is then passed to a linear layer and finally used in self-attention.

Once that the edge embedding is computed, the proper self-attention take place. This compute the key and query matrix as explained in Chapter 2. The matrix A represents the scaled outer product of Q and K matrices, that contains valuable information about how each node is relate to other node. A FiLM layer is used to inject bonding and distance information together with diffusion time step inside the attention weights. This is done as shown before by learning an affine transformation through parameterized linear transformation of distance, bonding and time step information. In particular:

$$A_1 = FiLM(A, \hat{E}) = \hat{E}W_1^e + (\hat{E}W_2^e) \odot A + A, \tag{4.6}$$

$$A_2 = FiLM(A_1, D) = DW_1^d + (DW_2^d) \odot A_1 + A_1, \tag{4.7}$$

$$A_3 = FiLM(A_2, y) = yW_1^y + (yW_2^y) \odot A_2 + A_2; \tag{4.8}$$

where W_1 and W_2 learnable weights associated to each feature injected inside self-attention layer to produce an affine transformation.

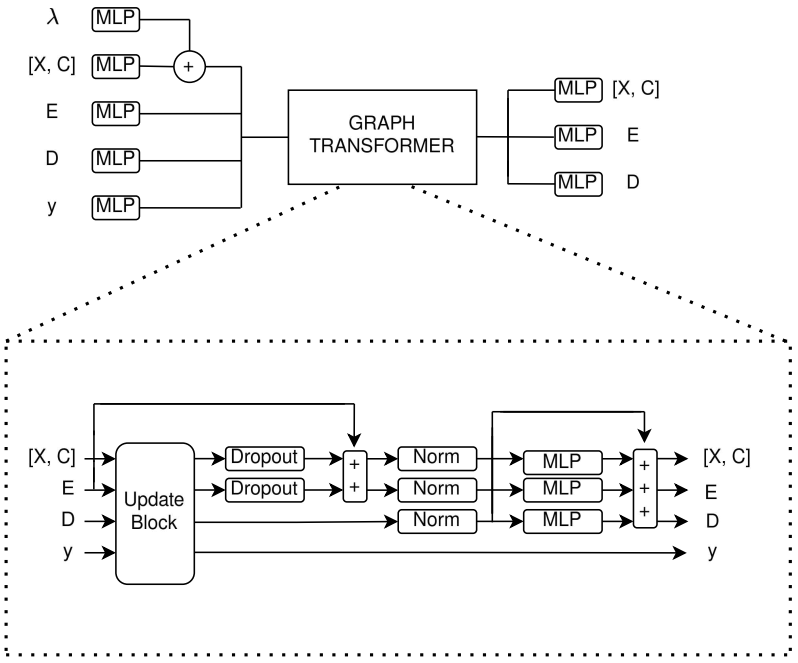


Figure 4.1: MiDi model architecture, at the top, the outline of the overall architecture is visible. Within the dotted lines, the details of the transformer architecture are shown.

It's important to consider that distance embeddings of the self-attention phase are computed through a dense layer starting from the attention score matrix. This allow to have all the information about the atomic structure and the bonding scheme of the molecule inside the final distance representation.

The remaining part follows the general graph transformer structure explained in Chapter 2; in the end, following what is done in [4], another FiLM layer is used, to inject again diffusion time step information inside the final representation.

As mentioned earlier, after the update block, a dropout layer is followed by a normalization layer and a feedforward layer, stacked sequentially for a chosen number of layers. The dropout layer is not used for distance embedding, in fact the regularization introduced by dropout lead

to a decrease of performance in generating distances. In the case of distances, also the skip connection between initial embedding and the results of update block is removed for the same reason.

The optimization is done in the same way as in MiDi, the learning architecture is trained to denoise each graph representation together with distance values. The objective function is different from what was seen before. Inside the learning architecture there are indeed four bond type, the three classes corresponding to single, double and triple bond and a class for no bond case. All the learning architecture exploited up to now have considered the no bond class inside the cross-entropy cost function, equally to other bonds. In the proposed model the loss associated to bond terms is split in two part: a binary cross entropy that control the presence or absence of bond and, if the bond is present, a cross entropy on the type of bond. The cost function to optimize is the following:

$$l(G, \hat{G}_t) = \lambda_x CE(X, p_\theta(x)) + \lambda_e(\text{BCE}(B, p_\theta(b)) + CE(\mathbf{E}_b, p_\theta(e_b)) + \lambda_c CE(C, p_\theta(c)) + \lambda_D ||\hat{D} - D||^2. \tag{4.9}$$

where BCE stands for binary cross entropy, B is the one hot encoding matrix with bond/no bond information, E_b is the one hot encoding tensor with bond type information. It's important to notice that the cross entropy on bond type is computed only when the bond actually exists between two atoms.

As done for MiDI it's possible to write the whole training and sampling procedure inside two algorithms. The training phase is built as explained in previous chapters, sampling a time step and then training the learning algorithm to denoise the noised representation. The main differences with respect to previous examples is the new loss function and in the usage of distance matrix.

Algorithm 4.1 Proposed Model - Training

1: $G = (X, E, C, D)$
2: Sample $t \sim \mathcal{U}(1, ..., T)$
3: $G_t \sim \mathcal{N}(\alpha_t D_{t-1}, \sigma_t^2 I) \times X^{t-1} Q_E^t \times \mathbf{E}^{t-1} Q_E^t \times C^{t-1} Q_C^t$
4: $\hat{p}_x, \hat{p}_e, \hat{p}_c, \hat{D} = f_\theta(G_t)$
5: Optimize.step $l(G_t, \hat{G}_t) = \lambda_x CE(X, p_\theta(x)) + \lambda_e(\text{BCE}(B, p_\theta(b)) + CE(\mathbf{E}_b, p_\theta(e_b)) + \lambda_c CE(C, p_\theta(c)) + \lambda_D ||\hat{D} - D||^2$

Algorithm 4.2 Proposed Model - Sampling stage

$G_T \sim \mathcal{N}(0,1) \times q_X(d_x) \times q_E(d_e) \times q_C(d_c)$
for $t = T$ to 0
  $\hat{p}_x, \hat{p}_e, \hat{p}_c, \hat{D}_t = f_\theta(G_t)$
  $z \sim \mathcal{N}(0,1)$
  $D_{t-1} = \frac{1}{\sqrt{\alpha_t}}(D_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\hat{D}_t) + \sigma_t z$
  $p_\theta(x_i^{t-1}|G_t) = \sum_x q(x_i^{t-1}|x_i = x, x_i^t)\hat{p}_i^X(x)$
  $p_\theta(e_{ij}^{t-1}|G_t) = \sum_e q(e_{ij}^{t-1}|e_{ij} = e, e_{ij}^t)\hat{p}_{ij}^E(e)$
  $p_\theta(c_i^{t-1}|G_t) = \sum_c q(c_i^{t-1}|c_i = c, c_i^t)\hat{p}_i^X(x)$
  $(X_{t-1}, C_{t-1}, E_{t-1}) \sim \prod_i p_\theta(x_i^{t-1}|G_t) p_\theta(c_i^{t-1}|G_t) \prod_{i,j} p_\theta(e_{ij}^{t-1}|G_t)$
  $G_{t-1} = (X_{t-1}, E_{t-1}, C_{t-1}, D_{t-1})$
end for
return G_0

The sampling stage is implemented in the same way as done for the previous chapter, the only difference stay in the fact that in this case we are treating distance matrices.

The model proposed is based on atomic distances, for this reason the ability to recover the three-dimensional structure from these distance relationships is crucial for this work. Computed coordinates should maintain the relative distances between each atom and ensure the triangular inequality. The method used in this work for generating three-dimensional coordinates from a distance matrix relies on principles of distance geometry and linear algebra techniques: in literature this method is called Principal Coordinates Analysis or classical Multidimensional scaling [41, 12]. This methods is based on the eigendecomposition of the Gram matrix associated to distance matrix, in particular the eigendecomposition enables the extraction of key geometric information encoded within the distance matrix, facilitating the rapid and accurate reconstruction of the spatial structure. There are other extensions of the Multidimensional Scaling (MDS) algorithm, such as non-metric MDS and generalized MDS [12]. These approaches are more flexible as they handle non-Euclidean spaces. However, since they rely on an iterative optimization process, they tend to be slower compared to classical MDS, which is based on pure linear algebra. Since coordinates of atoms are expressed in euclidean space, this work will use classic MDS.

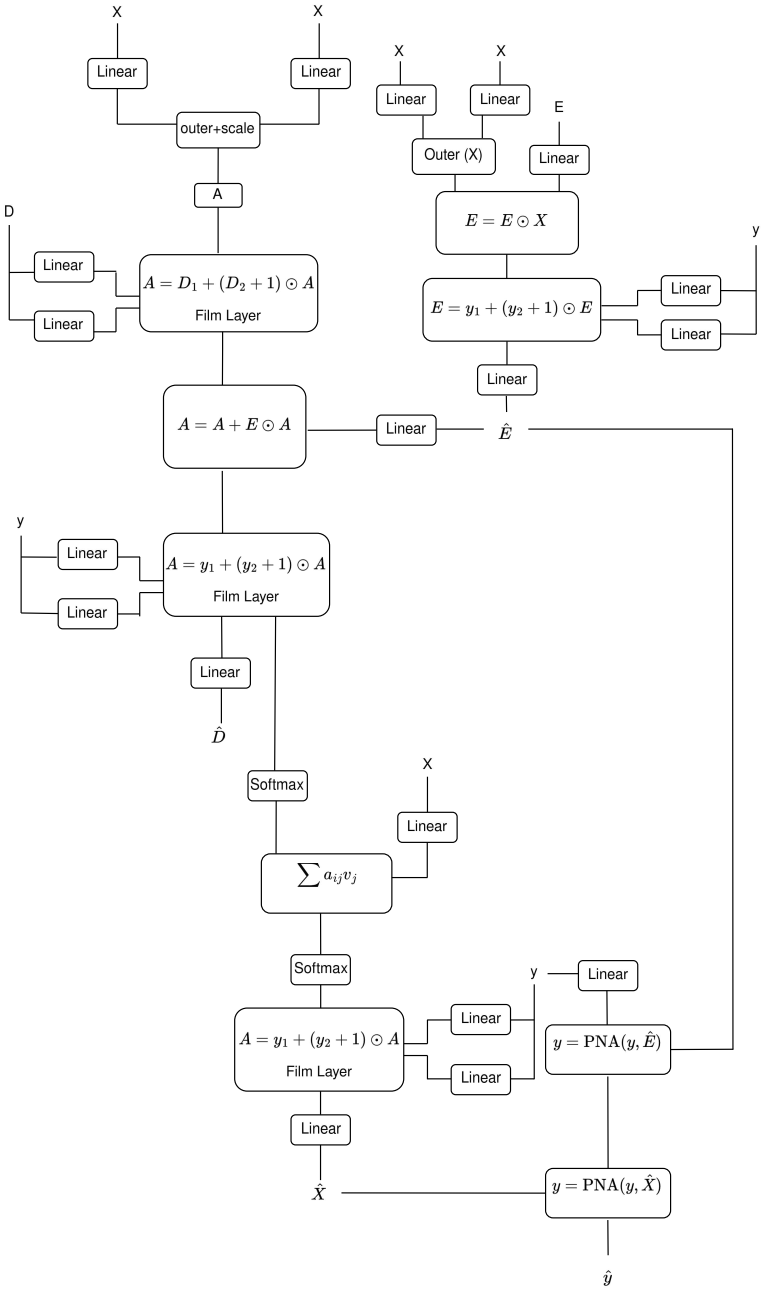


Figure 4.2: Update block architecture of proposed model

5 Experiments

In this chapter the benchmark and the proposed model will be evaluated with respect to unconditional molecular generation. The whole methodology is done for both the proposed and the benchmark model, asserting the goodness of the generated molecules from a chemical point of view and from a geometrical prospective, comparing the generated distance with the test distribution.

5.1 Dataset

Both the benchmark model and the proposed one will be tested using two datasets. Both the dataset will be split in 75% training, 15% validation and 10% testing. In the context of generative models the usage of each split dataset is not trivial, so I will sped few lines to explain what each part will be used for; the training set is used to train the generative model, the neural network architecture is fed with data from this to learn how to denoise corrupted graph structures. The validation set is used to guide the training process and monitor the model's real generalization capabilities. The evaluation of the model's denoising performance is done on the validation set, using a dataset that was not seen during training. Finally the test set is used as a benchmark to assess the quality of generated structure. The main focus is to compare generated molecules with a set that was not seen during the training.

It is also important to highlight the primary reasons for using two different datasets in the experiments: the main objective is to evaluate the robustness of the proposed methods com-

pared to benchmark models. In second stage, it's extremely important to consider how models scales with molecular dimension, especially when considering position and distance matrix. This work will treat two different dataset built with small molecules, the first one is QM9 [8] and the second is GDB13 [9]. The first dataset contains molecules up to 9 heavy atoms and the second up to 13. There exists other dataset with bigger molecules, like ZINC [42], where the average number of atom for molecule is 23 and the maximum 38, or Geom-Drugs [43] in which the average number of atoms is 44, with a maximum of 181; these datasets lead to a huge computational load both in training and in sampling phase, which made them unfeasible since the computational resources available for this thesis.

5.1.1 QM9

The first dataset used is QM9, short for "Quantum Mechanics 9" [8]. This dataset has become widely utilized in computational chemistry and machine learning, and over time, it has been established as one of the benchmarks datasets for evaluating molecular generative models.

It contains 134.000 small orgamic molecules, built using carbon (C), nitrogen (N), oxygen (O) and fluorine (F). These molecules have up to 9 heavy atoms. Regarding bonds type the dataset is built considering single, double and triple bonds. The dataset contains molecular information in SMILES format. In addition to the linear notation, the dataset also provides geometric information for each molecule, including the spatial coordinates of atoms.

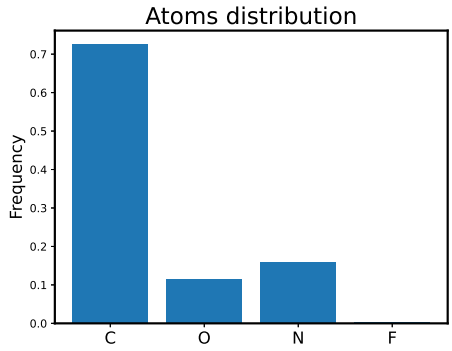
The dataset results extremely unbalanced with respect both atom and bond classes; this makes totally sense since it reflects the distribution in natural compounds. As it could be seen from Figure 5.1, the carbon is the most abundant atom type in the dataset, with more than the 72% of frequency with respect the training set. Oxygen and nitrogen together account for approximately 27% of the remaining atom types in the dataset, with fluorine being the least abundant, comprising about 0.3% of the atom types.

Things goes similarly for bond type class: the single bond forms the 87% of the bond in the training set, double bond about the 10% and triple bonds only the 3%.

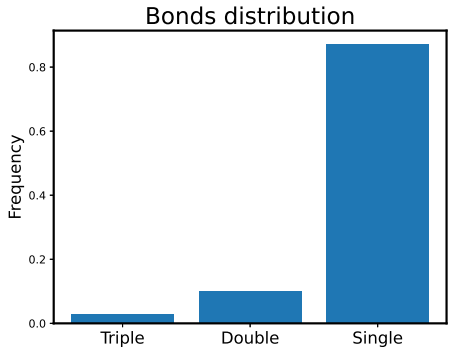
In order to fully understand proposed results, it's important to consider also how QM9 dataset was built. QM9 is a subset of GDB17 [10], a database of 166 billion of molecules up to 17 heavy atoms. This subset has been selected among GDB17, considering just atoms with up to 9 atoms and only the nicest one from a chemical point of view. In particular regarding this last point, QM9 is a manually curated dataset built only by energetic stable molecules without isomers; this provide a set of chemical compounds with an simple and regular chemical struc-

ture, from which is possible to compute experimentally geometrical and quantum features, which could be helpful in different applications. In particular in this work, coordinate features are extracted directly from the dataset, since they were prior computed experimentally.

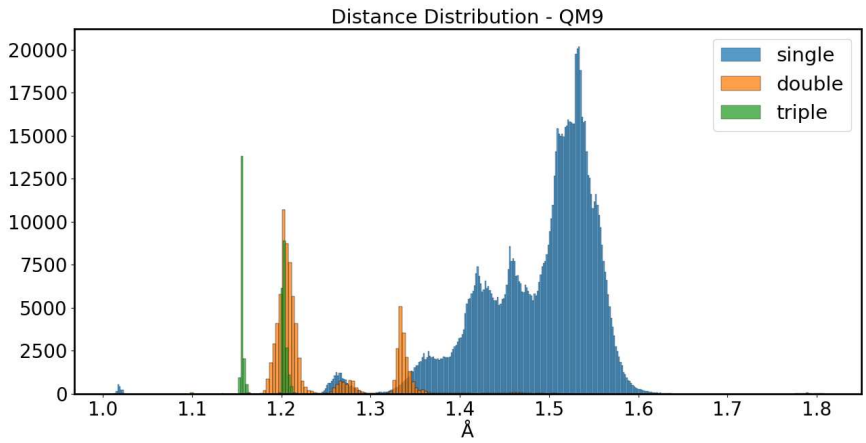
During the part dedicated to the purposes of this work, it was mentioned the fact that distances information is conditioned to bond type information. This could be visualize by looking at Figure 5.1c: it's easy to see that distances relative to single bonds are generally higher respect to distances associated to double and triple bonds. At the same time, this holds also for double bond distances respect to triple bond. In this sense there is an interesting fact to notice: at 1.2A there is an overlap between the mode of double and triple bond.



(a) Atom distribution for QM9 training set



(b) Bond distribution for QM9 training set



(c) Distance histrogram conditioned to bond type - QM9

Figure 5.1: Exploratory analysis - QM9

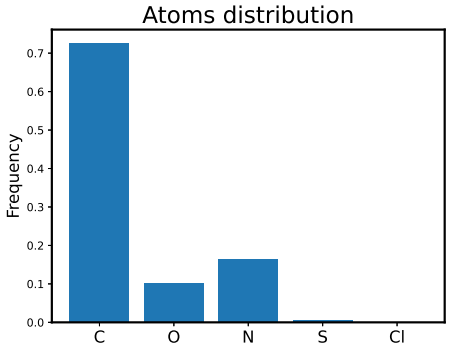
5.1.2 GDB13

The second dataset used in this work is GDB13 [9], namely Generated Database 13. This dataset is formed by small organic molecules that are generally slightly bigger than QM9 ones. In fact it's mainly built using molecules of 11, 12 or 13 atoms. The whole dataset contains about 1 billion of chemical compounds, due to computational constraint, in this work only a random subset of 169.550, will be considered. In this dataset molecules are built with carbon (C), nitrogen (N), oxygen (O), sulfur (S) and chlorine (Cl); regarding bond types, also in this case, there are only single, double and third bonds.

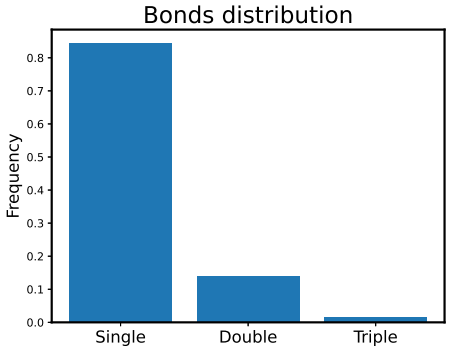
As for the previous case, also in this one, the dataset results extremely imbalance for both atom and class types. The most abundant atom is carbon, with the 72.5% of all the atoms in the dataset. Together oxygen and nitrogen represents the 26.7% of the atom types (respectively 10.2% for carbon and 16.5% for nitrogen). Lastly sulfur is present in 0.7% of atoms and the chlorine in 0.1%. The distribution of bond types is similar to that in QM9: single bonds make up 84% of all bonds, double bonds account for 14%, and triple bonds represent 1.6% of the total bonds.

Unlike QM9, in this case, the only constraint for building this dataset is the maximum number of atoms per molecule. This results in a set of molecules that are more challenging to handle, as there is no experimental supervision to focus solely on structures with a well-defined energetic configuration. For this reason, in this case the dataset doesn't contain any structural information about the conformation of each molecule in the space, but indeed just the SMILES string. To compute the coordinate of each string, a Universal Force Field (UFF) [44] optimization process is used, taking advantage of the implementation provided in RDKit [45]; UFF method is a computational method used to model and simulate a chemical system, the optimization process adjusts the atom positions to minimize the energy of the system, producing a physically realistic 3D structure. This method is not the most precise, but it is the most versatile that work with pretty much all the molecules, since each dataset could contains organic compounds with different chemical conformation.

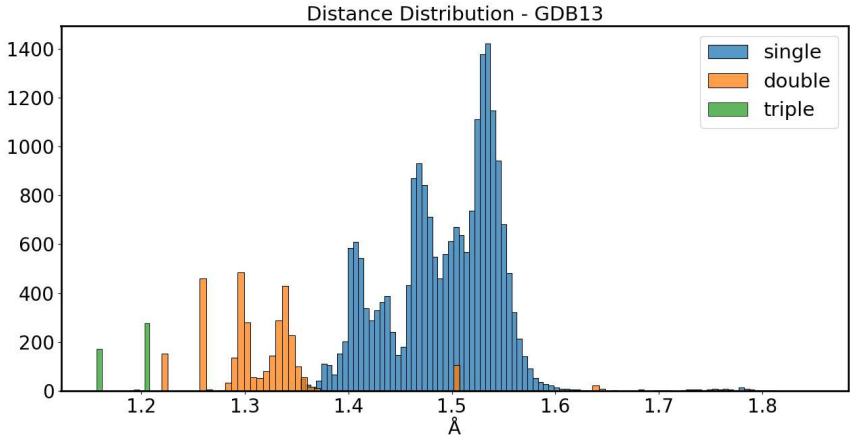
It's possible to see from Figure 5.2c, the histogram of training distance: in this case, the difference between the three distance distributions, conditioned on bond type, is more pronounced; generally the main modes of the three distribution are similar to what seen in QM9 case, double bonds are little shifted far from 1.2 A, not causing anymore an overlapping.



(a) Atom distribution for GDB13 training set



(b) Bond distribution for GDB13 training set



(c) Distance histogram conditioned to bond type - GDB13

Figure 5.2: Exploratory analysis - GDB13

5.2 Pre-Processing

Both of the previously considered datasets are built using SMILES notation, making it essential to have a processing pipeline that converts the molecular line notation into a graph format. Before this stage, it is important to perform certain preprocessing operations directly on the SMILES notation. The first operation is the removal of all hydrogen atoms from each molecule; as pointed out before, this doesn't affect the goddness of generated molecules, since it's generally possible to infer hydrogens from heavy atom presence. After this step each SMILES is kekulized: this procedure allow to rewrite all the aromatic bond in an alternating form of single and double bond. This procedure is essential in order to produce more precise graph represen-

tation when converting from SMILES string. Finally canonicalization algorithm is applied to each SMILES to ensure that there is an unique representation between molecule and string notation. After these transformation, all the SMILES are converted to a molecular graph representation and used for training the generative model.

In the case of GDB13, it is also necessary to generate three-dimensional coordinates. This is achieved by using SMILES before converting them into molecular graphs, and by leveraging RDKit's implementation of the Universal Force Field (UFF). The UFF takes the SMILES representation as input and returns a set of coordinates that minimize the molecule's energy landscape.

A schematic representation of both the processing pipelines is shown in figure 5.3 for both QM9 and GDB13 case.

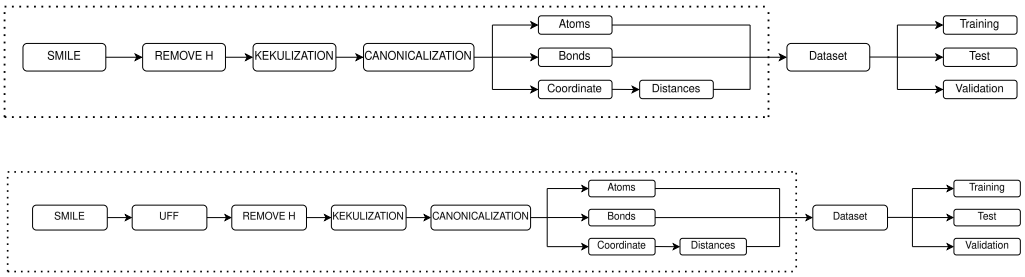


Figure 5.3: Input pipeline for creating the dataset, the part inside dotted line is done once, when the dataset is processed for the first time

5.3 Evaluation Metrics

In order to avoid overfitting, the final testing procedure is done on the best checkpoint in terms of denoising loss computed on the validation set; all the results of generated examples are computed using a sample of 10.000 generate molecules. Proposed model and MiDI results are shown in terms average and 95% confidence intervals on 5 generated examples set.

To evaluate the goodness of proposed model compared to benchmark one, it's important to consider some metrics that helps to understand if proposed generative model is able to create effective new molecules. To do so different types of metrics will be used [46].

First of all it's fundamental to assess if generated molecules are valid chemically: to do so the validity metric is used. Validity consider the percentage of generated molecules, which are chemically valid, over all the generated ones. To do so RDkit is used to verify the atomic composition and the bonding structure considering valency rules.

It's also important to assess the ability of the model to produce different molecules in output, the uniqueness measure the ratio of uniques molecules; evaluating this metric is essential, as a model may generate many valid molecules, but they could all end up being identical. This particular type of problem is called model collapse and it's a sign that the learning architecture fails in exploiting the unknown generative distribution.

In order to assess the generalization capabilities of the model, it' important to measure the number of generated molecules that are different from the ones present in the dataset; this is done with novelty metric, which measure the percentage of new generated molecules with respect the training set.

It is also essential to establish a point of comparison between the benchmark model and the proposed model to evaluate the quality of the three dimensional conformation. Since the benchmark model works with coordinates, we can compute a distance matrix from these coordinates and compare it with the one generated by the proposed model. In fact, the original paper uses generated distance as a measure to evaluate the quality of three-dimensional conformers. The two generated distribution will be compared using the Wasserstein distance method [28]. The Wasserstein distance is a distance function of two probability distribution P and Q defined as follows:

$$W(P, Q) = \inf_{\gamma \sim \Pi(P,Q)} \mathbb{E}_{(p,q)\sim\gamma}[|p - q|], \tag{5.1}$$

where $\Pi(P, Q)$ is the set of all possible joint probability distribution between P and Q and γ is a particular distribution among this set. From a practical point of view, it's possible to see this quantity as the measure of the smallest distance between two distribution weighted by the joint probability of the two. For discrete case probability it's easy to see that the previous equation could be rewritten as:

$$\inf_{\gamma \sim \Pi(P,Q)} \mathbb{E}_{(p,q)\sim\gamma}[|p - q|] = \sum_{p,q}^{\infty} \gamma(p, q)|p - q|. \tag{5.2}$$

In the case in analysis the main aim is to measure the distance between target distance values and generated ones. In this situation the Wasserstein distance represents the minimum "cost" of transforming one distribution into the other. To compute this metric in discrete case, the process involves the creation of histograms for both the generated and target distributions,

where each bin holds the counts of values that fall into it. These two discrete distributions are then sorted and the absolute difference between corresponding points are computed. To assess the real capabilities of the model, it's also necessary to consider distance as conditioned to bond information, since the type of bond has a huge impact on distance and viceversa. This allows to check if generated distances are also meaningful from a chemical point of view. To do so the Wasserstein distance metric is computed as follows:

$$W_{\text{bonds}}(D_t, D_g) = \sum_{b \in \text{bond types}} p(b)\, W(D_b^t, D_b^g), \tag{5.3}$$

where $p(b)$ is the frequency of a particular bond type inside the test set, D_b^t is the discretized distribution of distances inside the test set, marginalized by the bond type, D_b^g is the same distribution, but with distances generated by the model. Then the final sum is weighted by the bond frequency. This metric will be called distance error from now on.

5.4 Model Selection

Model selection is a fundamental process in any deep learning based method: the main scope is to choose the best-performing model version among different configurations. In this case, the proposed method consists in a graph transformer, where many architectural choices are done in according to what done [5] and [4]. Because of computational limitations all the model selection has been done only for QM9 case: the parameters chosen for this case will be used also in GDB13 one. Model selection was performed using a grid search strategy, focusing on two key hyperparameters: the learning rate and the loss function weights.

For the learning rate, various configurations were tested, beginning with two widely spaced values: 0.001 and 0.01. As values around 0.01 resulted in an unstable learning trajectory, additional configurations were tested starting from this value, decreasing incrementally by 0.002.

Regarding loss function weights, the strategy adopted was the same used for learning rate. In this case the starting configuration was the one used in [5], then other configuration were tested moving around this starting point with an initial step of 0.5, then using a smaller one of 0.1 around the chosen one.

The configurations of the learning rate and loss function weights were tested independently of each other. This approach is not the most correct, since they should be tested together, but,

due to computational constraint, this wasn't feasible.

5.5 Implementation details

All the model exposed here have been trained with a batch size of 1024 and for 2000 epochs. For reproducibility this whole experimental set is done in according to what proposed in MiDI article [5]. To ensure the full reproducibility with MiDi, its code has been integrated and tested into the framework utilized in this thesis: all the results are coherent with original ones.

The learning rate for both the dataset is fixed at 0.002, chosen as the best one among other different configuration. The selection of loss function hyperparameters was initially based on the starting configuration used for MiDI, followed by adjustments made by exploring neighboring configurations. In the end the final chosen parameters are: $\lambda_x = 1, \lambda_c = 1, \lambda_e = 2$ and $\lambda_d = 1$.

Regarding architectural choices, many decisions were made in alignment with the setup described in [5]. More in detail each MLP used in initial encoding phase is structured with two linear layer divided by a non linear activation function. For all the features, except for distances, the number of neurons are chosen in according to what seen in already mentioned articles, for distance feature the choice has been done testing several different configuration. The number of attention heads as well as the number of layers inside the graph attention are the same as [5]; the same is valid also for hidden dimension size of transformer embedding. After the attention phase, node, edge and diffusion time step embedding are processed, as usual in graph transformer, by a dropout, a normalization layer and a MLP layer, where the hyperparameters are chosen according to [5] and [4]. Distance embedding are processed without a dropout layer: this because the regularization performed lead to produce inaccurate distance embeddings (especially for double and triple bonds). In order to produce a richer embedding, the MLP layer for distance embedding is built by four layer, instead of two as for other embeddings: this configuration has been chosen after many attempt, considering deeper and shallower models, which leads to worse results. The proposed model uses the SiLU (Sigmoid Linear Unit) function for all activation layers. This choice was made because the SiLU function offers several key advantages: its smooth gradient and ability to handle both positive and negative input values. These properties enhance the model's stability and performance, particularly in generating accurate distance values within molecular structures.

All the models will be tested with the same experimental settings: in particular, since hydrogens atoms could be easily inferred form heavy atoms, in this work they will not be considered.

This procedure is extremely used in literature [5, 6, 4], mainly to reduce computational costs.

The proposed model is coded in Python using Pytorch [47], together with pytorch-lighting [48]. All the pre-processing and post-processing stages performed on molecules are done using RDkit [45], a popular open source toolkit for cheminformatics, with an python API. This package contains many methods that permit to deal with molecular object with computational chemistry tool.

In the end it's important to mention the model dimension together with training time for both the proposed and MiDi model. Proposed model consists of about 20 million of parameters (precisely 20.003.660) and it's trained for 3 days. MiDi model consists of 20 million of parameters (precisely 20.124.219) and it's also trained for three days. Both the models have been trained on one Nvidia A5000 GPU.

5.6 Results

Let's start considering the ability of proposed model in generating suitable 2D conformers with respect testing distribution, then chemical property of generated molecules will be revised and finally there will be the assessment of distance generation, with a comparison with MiDi model. To evaluate the quality of the generated examples, the first step is to check if the atom and bond distributions are similar to those in the test set: for QM9 all the results are reported in Tables 5.1 and 5.2; in these it's possible to see that generated distribution is extremely similar to test one, for both atom and bond type. The proposed model is able to deal equally with over represented data type (like single bond and carbon atom) and with under represented data (like Fluorine and triple bond).

	C	O	N	F
Test Distribution	0.725	0.114	0.159	0.003
Generated Distribution	0.745 ± 0.005	0.098 ± 0.005	0.156 ± 0.001	0.003 ± 0.001

Table 5.1: Generated vs test atom distribution for QM9

For GDB13 dataset (Table 5.1 and Table 5.4) the situation is similar to the previous one: in this case the dataset is much more complex but, in the end, the proposed model succeed in learning the atom and bond distribution. The model is robust enough to not overfit on the most distributed classes (like single bond and carbon atom) and it is also able to exploit information related to the least abundant classes.

	Single	Double	Triple
Test Distribution	0.870	0.099	0.030
Generated Distribution	0.889 $\pm$ 0.006	0.082 $\pm$ 0.008	0.031 $\pm$ 0.004

Table 5.2: Generated vs test bond distribution for QM9

	C	O	N	S	Cl
Test Distribution	0.725	0.102	0.165	0.007	0.001
Generated Distribution	0.759 $\pm$ 0.004	0.083 $\pm$ 0.001	0.151 $\pm$ 0.004	0.005 $\pm$ 0.001	0.002 $\pm$ 0.001

Table 5.3: Generated vs test atom distribution for GDB13

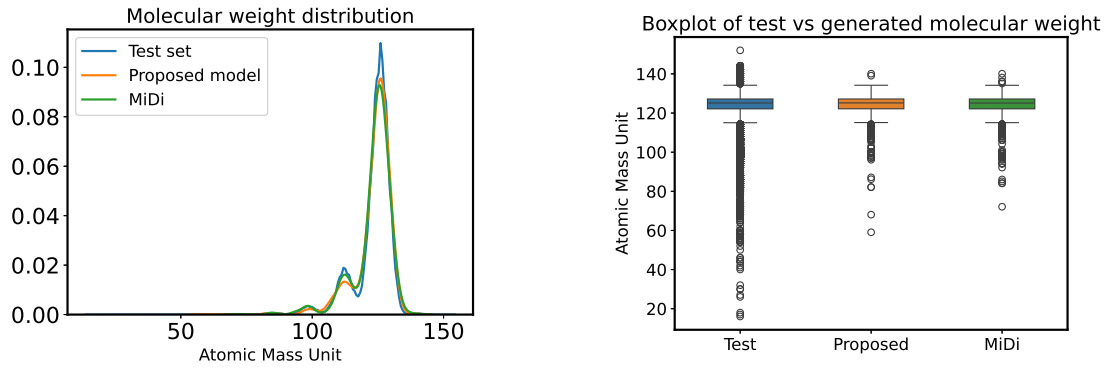
	Single	Double	Triple
Test Distribution	0.843	0.141	0.016
Generated Distribution	0.852 $\pm$ 0.013	0.118 $\pm$ 0.010	0.019 $\pm$ 0.004

Table 5.4: Generated vs test bond distribution for GDB13

It's necessary also to consider the property of each generated molecule with respect the test set. As shown in Table 5.5, for QM9, over 99% of generated molecules are chemically valid, this result agrees totally with MiDI results. Looking at uniqueness and novelty values, the proposed model generates a similar proportion of unique molecules compared to the baseline, however, many of these molecules are already present in the test set. In general, scientific literature emphasizes the importance of maximizing both uniqueness and novelty, in the specific case of QM9 this is not totally true. Authors of [4] and [49] highlight the fact that QM9 is a curated dataset which contains specific molecules that verify a set of constraint; given specific constraints, QM9 represent an exhaustive enumeration of molecular types, generating a large number of molecules outside this dataset could indicate that the learning architecture is struggling to capture the chemical space represented by QM9. Given this interpretation the proposed model achieve better results compared to MiDi, since it produces a lower value of novel molecules (41% against 49% of baseline model) and an higher number of uniques molecules among the generated ones.

In line with these considerations, it's important to verify whether the proposed model can generate molecular compounds that are structurally similar to those in the test set. Specifically, the generated molecules should not be excessively large or overly small. On average, the generated molecules contain 8.23 atoms, ranging from a minimum of 4 to a maximum of 9 atoms,

while the average molecule size in the test set is 8.4 atoms. Also looking at the molecular weight distribution, for both test and generated molecules of proposed model, it's possible to ensure that, generally, generated molecules are structurally similar to what's contained in the dataset. The molecular weight plots (Figures 5.4) reveal that both the proposed model and the MiDI model struggle to generate outlier structures; they tend to favor average-sized molecules, making it more challenging to produce particularly large or small molecules.



(a) (b)

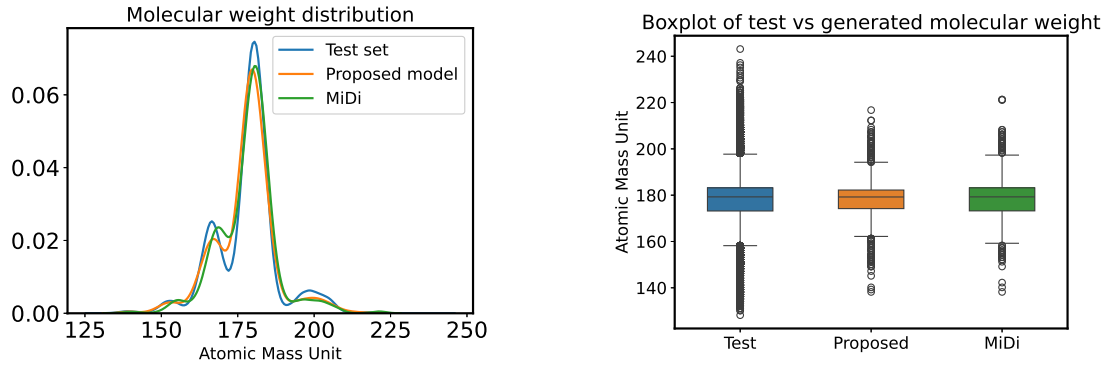
Figure 5.4: Density plot and boxplot of molecular weight - QM9

	Validity	Uniqueness	Novelty	Distance Error
MiDi	0.995 ± 0.001	0.958 ± 0.002	0.492 ± 0.000	0.008 ± 0.003
Proposed Model	0.995 ± 0.001	0.957 ± 0.001	0.412 ± 0.004	0.005 ± 0.001

Table 5.5: MiDi vs Proposed model metrics QM9

It's possible to write similar consideration also for GDB13 case. The amount of valid molecules is comparable between proposed and MiDi model, and in both case the number is generally extremely high. In this case, since the dataset is built without a specific supervision, the shape and configuration of molecules in the dataset is extremely heterogeneous, and so there are more possible configurations to learn. For this specific reason, in this case, the value of novelty is near to 1, just like the value of uniques molecules. The proposed model performs similarly to baseline model: both approaches generally succeed in generating chemically valid molecules that are unique and free from redundancy relative to the test set. By examining the molecular weights of the generated compounds, we observe that, on average, the proposed model produces molecules with structural characteristics closely resembling those in the test set. Specifi-

cally, the generated molecules have an average atom count of 12.2, which aligns well with the average atom count of molecules in the test set, which is 12. This similarity suggests that the model effectively captures the structural patterns and molecular composition of molecules also in the test data. On the other hand, also in this case both proposed and MiDI model struggle in dealing with outlier structures, as could be seen in Figures 5.5.



(a) (b)

Figure 5.5: Density plot and boxplot of molecular weight - GDB13

	Validity	Uniqueness	Novelty	Distance Error
MiDi	0.997 ± 0.001	0.999 ± 0.001	0.999 ± 0.001	0.026 ± 0.008
Proposed Model	0.998 ± 0.001	0.999 ± 0.001	0.999 ± 0.001	0.004 ± 0.003

Table 5.6: MiDi vs Proposed model metrics GDB13

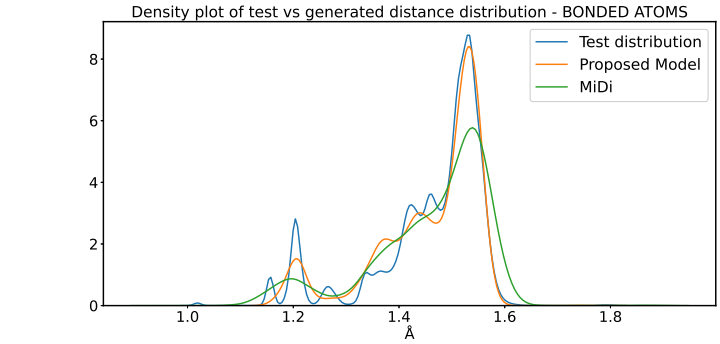
It's very important to look also at generated distances results. In the case of MiDI, distances are computed starting from generated positions. Generated distances from both models will be considered in a first stage, in terms of bonded and not bonded atoms: this will be done in order to assess the ability of both models to exploit the general structure of distances within a molecule. In a second stage, all the bonded distances will be considered as conditioned to bond type.

In QM9 case, from Table 5.5, it is possible to see that, overall, the distances generated by the proposed models are more realistic and similar to those in the test set compared to those generated by MiDI. In the case of bonded atoms, visible in Figures 5.6a 5.6b, the generated distances follows better the trend of test set and succeeds in generating values around the mode. The model struggles more in dealing with smaller distances (related to double and triple bonds)

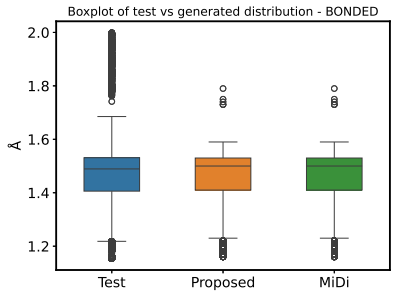
but performances are still better compared to MiDI case. Regarding distances of non bonded atoms, depicted in Figures 5.6c 5.6d, overall the proposed model is able to generate distances that are extremely close to the test distribution and, comparing with MiDI, it's possible to see that the generation is more precise with proposed model. The proposed model also outperforms MiDI in predicting single bond distances, as could be seen in Figures 5.6e 5.6f, generating a center of mass around the correct mode. Regarding double and triple bond distances, visible at Figures 5.6g 5.6h 5.6i 5.6j, performances drop significantly and this could be caused by two main facts: the first concerns the fact that these bonds together represents about the 15% of the bonds in the dataset, so it could be possible that there is not enough information to learn sufficiently the generated distribution; this is particularly valid for triple bonds, that are around the 1% of all bonds. Moreover to make things even harder there is the fact that, at $1.2\mathring{A}$, both double and triple bond distance distribution have a mode, this can make it more difficult for learning architecture to figure out the real shape of generated distribution. Overall it's possible to say that, even though the proposed model struggle in dealing with these situation, it performs better than MiDI, in placing the mass of generated examples in the right range of values.

Generated distances can also be considered for the GDB13 case: as pointed out in Table 5.6, it's possible to see that proposed model perform remarkably better than MiDI model in generating new distances. Density plot together with boxplots for bonded and non-bonded atoms, depicted at Figures 5.7a 5.7b 5.7c 5.7d, demonstrate that the proposed model captures more effectively the shape of the generated distance distribution than MiDI. The generation for non bonded atoms is indeed less precise respect the QM9 case, but it's worth to mention that GDB13 molecules are bigger and more complex, so it make sense that generally non boned distance are less precise. Regarding single bonds, seen in Figures 5.7e 5.7f, it's evident that the distances generated by the proposed model align well with the test distribution, outperforming MiDI significantly. Double and triple bonds distances depicted at Figures 5.7g 5.7h 5.7i 5.7j remain difficult to generate. For double bonds, the proposed model produced a unimodal distribution, whereas the actual distribution is notably more complex. Triple bonds distances are still difficult to manage, since the generated distribution of proposed model fails in detecting the real test distribution.

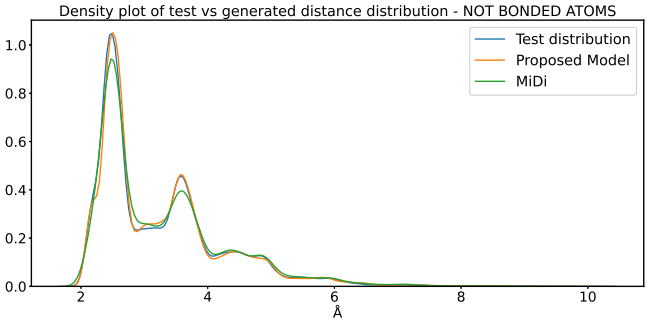
In Figures 5.8 and 5.9 they are reported 25 different molecules generated by proposed model trained on QM9 and GDB13 dataset, with both 2D and 3D. Three dimensional features are extracted using the method mentioned in Chapter 4.



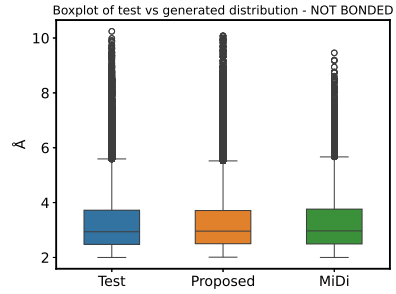
(a)



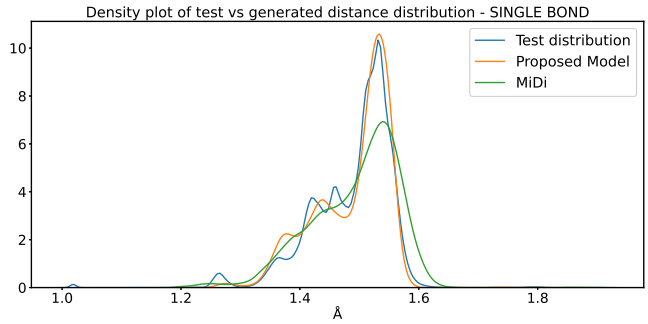
(b)



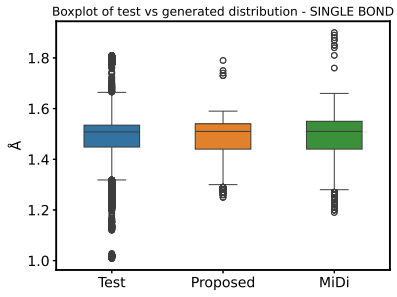
(c)



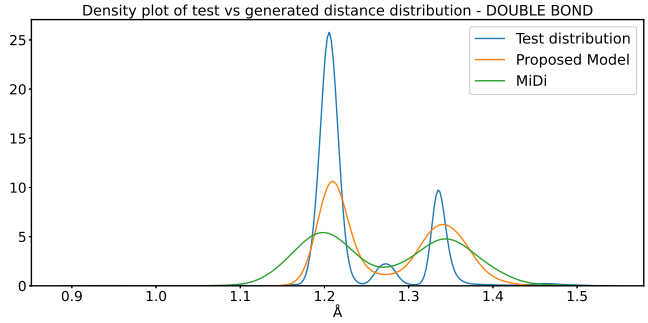
(d)



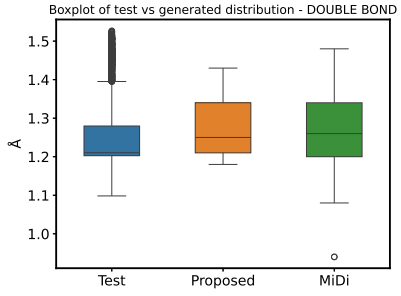
(e)



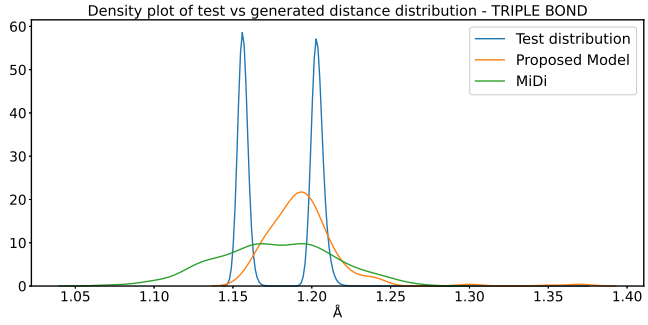
(f)



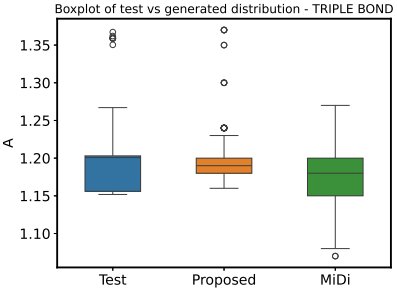
(g)



(h)

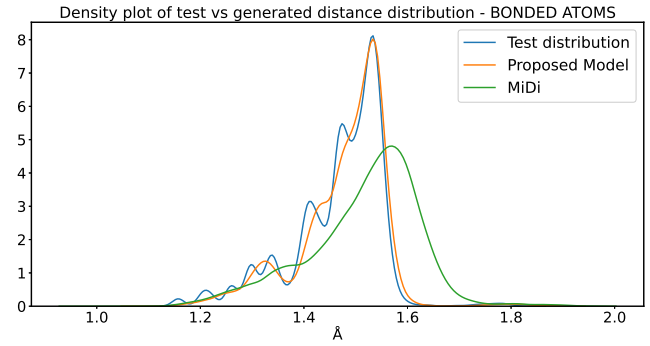


(i)

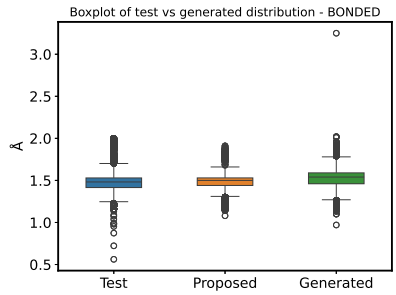


(j)

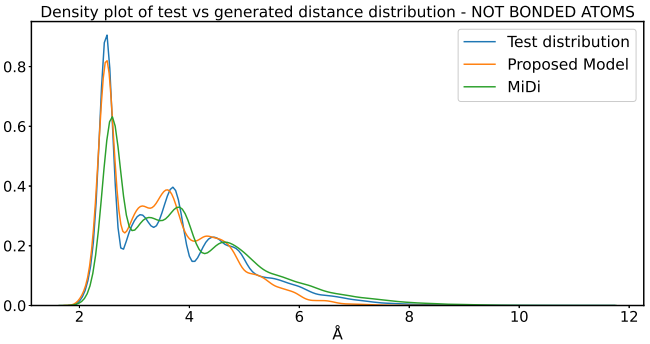
Figure 5.6: Density plot and boxplots for generated distances of QM9



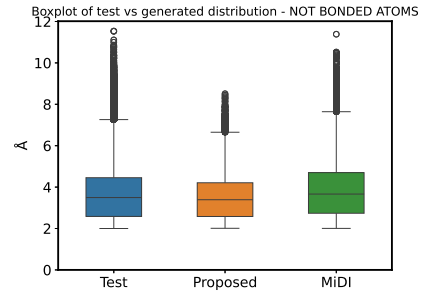
(a)



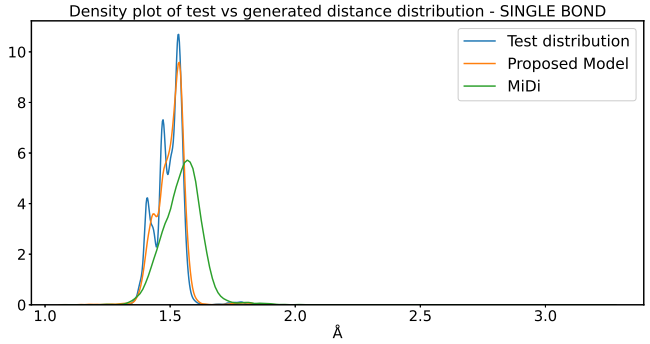
(b)



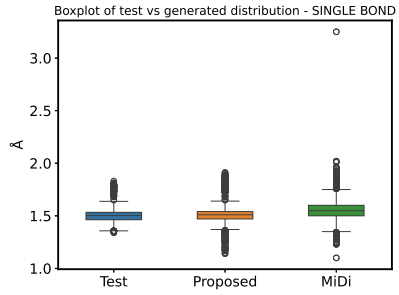
(c)



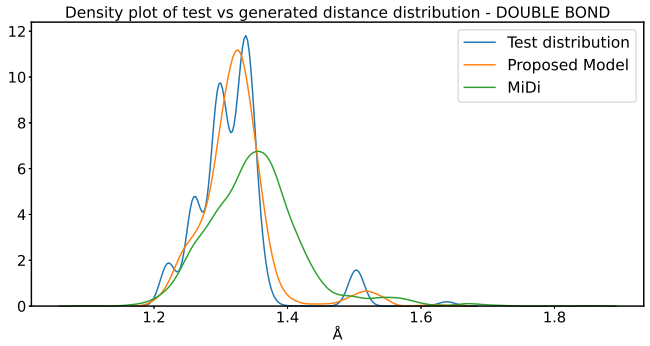
(d)



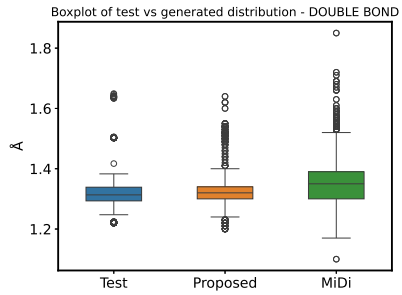
(e)



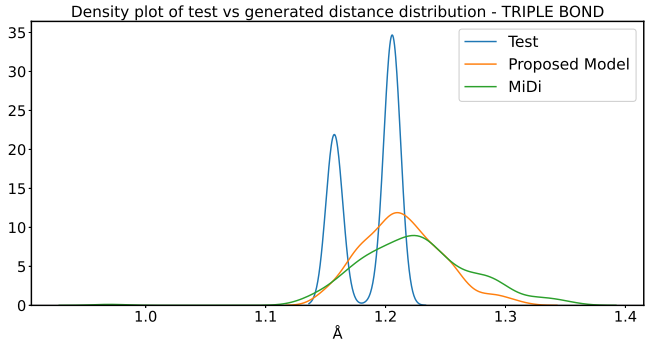
(f)



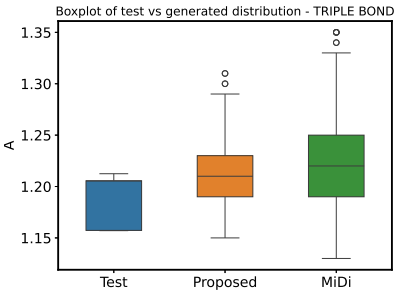
(g)



(h)



(i)



(j)

Figure 5.7: Density plot and boxplots for generated distances of GDB13

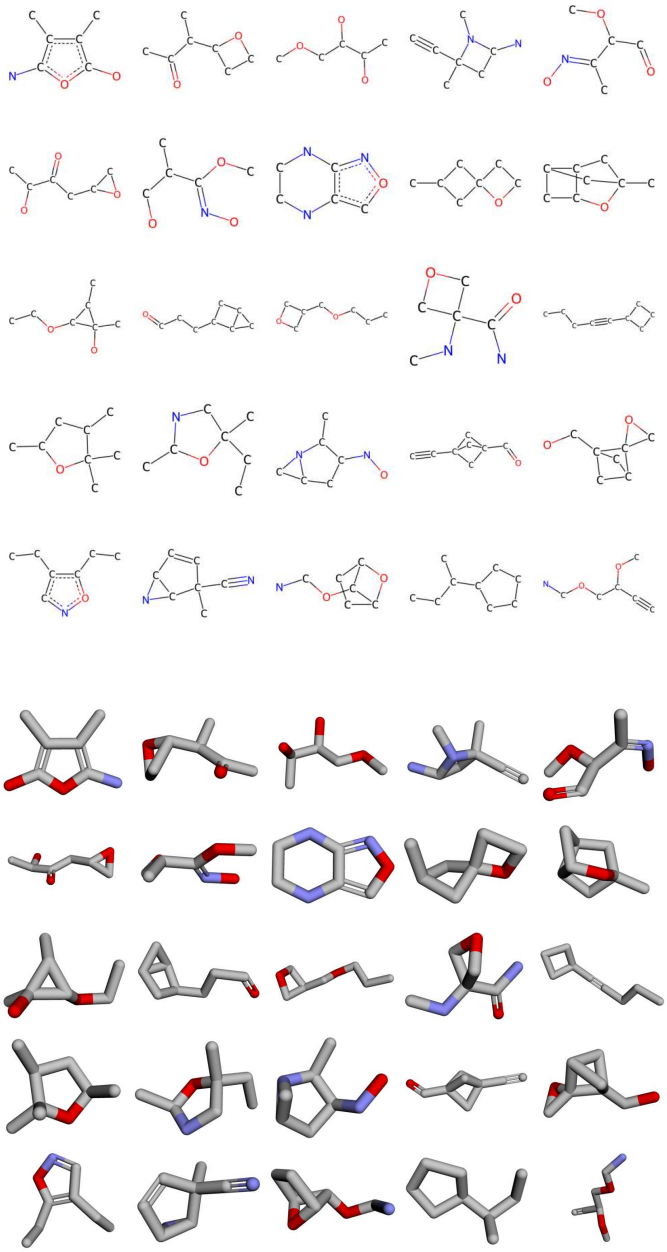


Figure 5.8: Non curated generated samples of proposed model - QM9

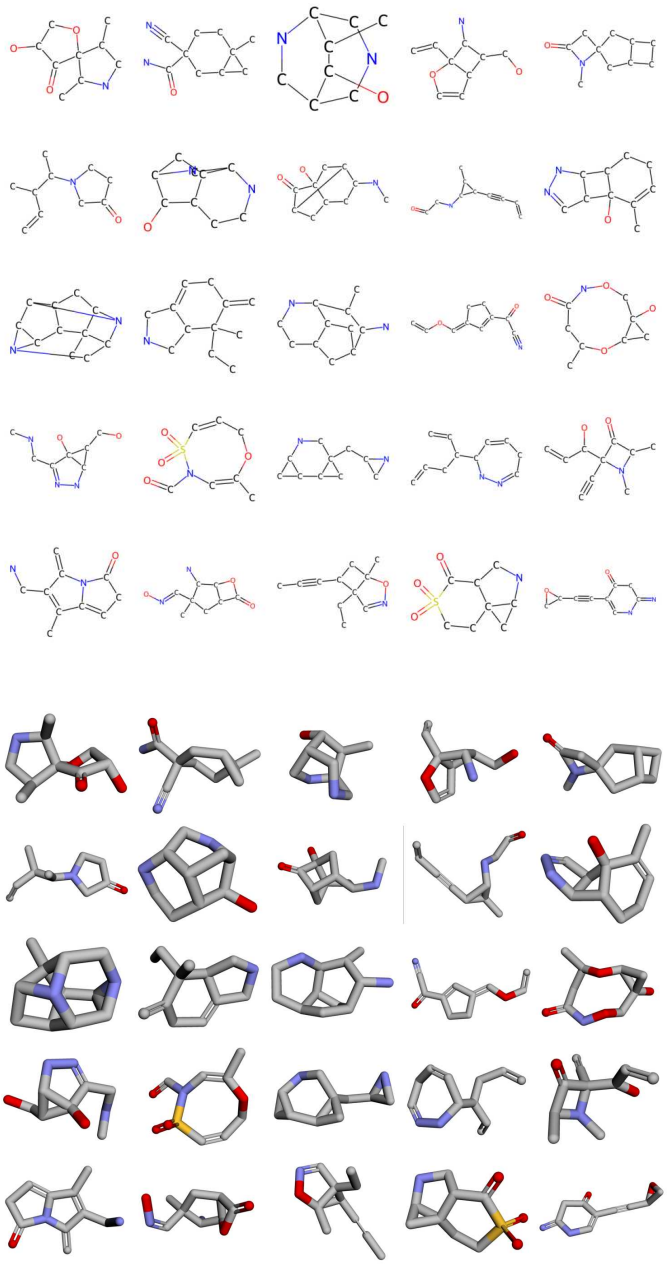


Figure 5.9: Non curated generated samples of proposed model - GDB13

6

Conclusion

Recently, deep generative models have made significant advances in addressing the complex challenge of modeling molecular structures. These models have demonstrated impressive progress in their ability to generate, analyze, and optimize molecular configurations, which allows researchers to explore chemical space with unprecedented depth and precision. In particular, since a chemical molecule exists in space, utilizing both 2D features, which represent the atomic and bonding scheme, along with the 3D conformation of atoms, is essential to fully exploit the information underlying the chemical space and generate realistic examples. All the methods available in the literature to exploit spatial conformation use Euclidean coordinates. This approach is problematic because it requires researchers to develop models that are rotationally invariant or capable of learning the rotational bias. This thesis propose a new approach in dealing with spatial conformation, by utilizing the atomic distance matrix instead of Euclidean coordinates. This approach overcomes issues related to coordinate rotation and enables the model to exploit new information about the strong relationship between atomic distances and the bonding scheme.

The model proposed in this work is built using a discrete denoising diffusion probabilistic model for graphs implemented with a graph transformer. The learning architecture is customized to facilitate the flow and reconstruction of the 2D molecular structure, including the distances between all atoms. Looking at results in chapter 5, it's possible to see that proposed model generates molecules that are chemically valid, unique and free from redundancy rela-

tive to the test set. This demonstrates that the model is robust and capable of generalizing effectively with respect to the test set. It also performs comparably to the baseline model described in chapter 5, based on key metrics used to evaluate the structural integrity of generated molecules. MiDi, along with many other models based on three-dimensional coordinates, employs a distance error metric to assess the quality of generated 3D conformers. When compared with the results of the proposed model, it is clear that, for both datasets used in this thesis, the performance of the proposed model significantly outperforms the baseline models. This is due to the fact that in the proposed model, distance information flows explicitly, whereas in the baseline models, it is computed implicitly, starting from coordinates. Proposed model outperforms baseline one in generation of both bonded and not bonded distances, despite this, both models have a problem in generating distances of triple bonds: this is probably due to the low number of these bonds inside the datasets in analysis.

The key point of proposed model is in the dependency between bond class and distance information: in fact exploiting this information is beneficial, since the learning architecture can learn more about how one variable influences another, leading to more accurate and realistic generated outputs. This particular fact could be further investigate in future works, in fact this thesis represents just a first step for a more ambitious project: develop a denoising diffusion probabilistic model that generates distances conditioned to bond type information. This model by considering explicitly the dependence between distance and bond information, probably will generated even better molecules, with even more precise triple bond distance.

The implementation code of this work is available at `https://github.com/MrcBalla/D4-code`

7 Appendix

.1 Loss function - Diffusion Models

The training is done according the maximum likelihood method, the idea is to find out the parameters necessary to train the model.
Let's consider the following marginalization with importance sampling:

$$\begin{aligned} p_\theta(x_0) &= \int p_\theta(x_0, ..., x_T) dx_1, ..., dx_T; \\ &= \int p_\theta(x_0, ..., x_T) \frac{q(x_1, ..., x_T|x_0)}{q(x_1, ..., x_T|x_0)} dx_1, ..., dx_T; \\ &= \int q(x_1, ..., x_T|x_0) p(x_T) \prod_{t=1}^{T} \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} dx_1, ..., dx_T. \end{aligned}$$

This way of writing will be useful inside the actual proof of the loss function.
The loss function is derived using the maximum likelihood criterion:

$$\underset{\theta}{\text{argmax}}\, \mathbb{E}_{x_0 \sim q}\, log(p_\theta(x_0)) = \underset{\theta}{\text{argmax}} \int q(x_0) \log(p_\theta(x_0)) dx_0.$$

From now on the argmax term will be omitted; it's possible to plug in the formulation of p_θ

derived previously, by applying the Jensen inequality and the Markov property:

$$
\begin{aligned}
&\int q(x_0)\log(p_\theta(x_0))dx_0;\\
&=\int q(x_0)\log\int q(x_1,...,x_T|x_0)p(x_T)\prod_{t=1}^{T}\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}dx_1,...,dx_Tdx_0;\\
&\leq\int q(x_0,...,x_T)\log p(x_T)\prod_{t=1}^{T}\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}dx_0,...,dx_T;\\
&=\int q(x_0,...,x_T)\log p(x_T)\prod_{t=1}^{T}\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}dx_0,...,dx_T;\\
&=\mathbb{E}_q[\log p(x_T)\prod_{t=1}^{T}\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}];\\
&=\mathbb{E}_q[\log\prod_{t=1}^{T}\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}]+\mathbb{E}_q[\log p(x_T)];\\
&=\sum_{t=2}^{T}\mathbb{E}_q[\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}]+\mathbb{E}_q[\log\frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}]+\mathbb{E}_q[\log p(x_T)];\\
&=\sum_{t=2}^{T}\mathbb{E}_q[\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1},x_0)}]+\mathbb{E}_q[\log\frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}]+\mathbb{E}_q[\log p(x_T)];\\
&=\sum_{t=2}^{T}\mathbb{E}_q[\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)}\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}]+\mathbb{E}_q[\log\frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}]+\mathbb{E}_q[\log p(x_T)];\\
&=\sum_{t=2}^{T}\mathbb{E}_q[\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)}]+\mathbb{E}_q[\log\frac{q(x_1|x_0)}{q(x_T|x_0)}]+\mathbb{E}_q[\log p(x_T)]+\mathbb{E}_q[\log\frac{p_\theta(x_0|x_1)}{q(x_1|x_0)}];\\
&=\sum_{t=2}^{T}\mathbb{E}_q[\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)}]+\mathbb{E}_q[\log\frac{p_\theta(x_T)}{q(x_T|x_0)}]+\mathbb{E}_q[\log p_\theta(x_0|x_1)].
\end{aligned}
$$

The preceding formula could be written in terms of minimum and with kullback leibler divergence:

$$\mathbb{E}_q[D_{KL}(q(x_T|x_0)||p_\theta(x_T))+\sum_{t=2}^{T}D_{KL}(q(x_{t-1}|x_t,x_0)||p_\theta(x_{t-1}|x_t))-\log p_\theta(x_0|x_1)].$$

This is the analytical computable loss that needs to be minimized during the learning phase.

It's possible to recognise three elements depending on the time step of the reverse process: L_T, L_{t-1} and L_0.

References

[1] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv preprint arXiv:2012.09699*, 2020.

[2] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," *Journal of chemical information and computer sciences*, vol. 28, no. 1, pp. 31–36, 1988.

[3] ——, "Smiles. 3. depict. graphical depiction of chemical structures," *Journal of chemical information and computer sciences*, vol. 30, no. 3, pp. 237–243, 1990.

[4] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard, "Digress: Discrete denoising diffusion for graph generation," *arXiv preprint arXiv:2209.14734*, 2022.

[5] C. Vignac, N. Osman, L. Toni, and P. Frossard, "Midi: Mixed graph and 3d denoising diffusion for molecule generation," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2023, pp. 560–576.

[6] E. Hoogeboom, V. G. Satorras, C. Vignac, and M. Welling, "Equivariant diffusion for molecule generation in 3d," in *International conference on machine learning*. PMLR, 2022, pp. 8867–8887.

[7] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[8] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific data*, vol. 1, no. 1, pp. 1–7, 2014.

[9] L. C. Blum and J.-L. Reymond, "970 million druglike small molecules for virtual screening in the chemical universe database gdb-13," *Journal of the American Chemical Society*, vol. 131, no. 25, pp. 8732–8733, 2009.

[10] L. Ruddigkeit, R. Van Deursen, L. C. Blum, and J.-L. Reymond, "Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17," *Journal of chemical information and modeling*, vol. 52, no. 11, pp. 2864–2875, 2012.

[11] W. J. Wiswesser, "The wiswesser line formula notation," *Chem. Eng. News*, vol. 30, no. 34, pp. 3523–3526, 1952.

[12] W. S. Torgerson, "Multidimensional scaling: I. theory and method," *Psychometrika*, vol. 17, no. 4, pp. 401–419, 1952.

[13] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 260–13 271, 2020.

[14] D. P. Kingma, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[16] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International conference on machine learning*. PMLR, 2015, pp. 1530–1538.

[17] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[18] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*. PMLR, 2015, pp. 2256–2265.

[19] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.

[20] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *International conference on machine learning*. PMLR, 2018, pp. 2323–2332.

[21] C. Zang and F. Wang, "Moflow: an invertible flow model for generating molecular graphs," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 617–626.

[22] J. Ross, B. Belgodere, V. Chenthamarakshan, I. Padhi, Y. Mroueh, and P. Das, "Large-scale chemical language representations capture molecular structure and properties," *Nature Machine Intelligence*, vol. 4, no. 12, pp. 1256–1264, 2022.

[23] M. Xu, L. Yu, Y. Song, C. Shi, S. Ermon, and J. Tang, "Geodiff: A geometric diffusion model for molecular conformation generation," *arXiv preprint arXiv:2203.02923*, 2022.

[24] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg, "Structured denoising diffusion models in discrete state-spaces," *Advances in Neural Information Processing Systems*, vol. 34, pp. 17 981–17 993, 2021.

[25] M. Krenn, F. Häse, A. Nigam, P. Friederich, and A. Aspuru-Guzik, "Self-referencing embedded strings (selfies): A 100% robust molecular string representation," *Machine Learning: Science and Technology*, vol. 1, no. 4, p. 045024, 2020.

[26] D. Fauque, "1919-1939: the first life of the union," *Chemistry International*, vol. 41, no. 3, pp. 2–6, 2019.

[27] D. Weininger, A. Weininger, and J. L. Weininger, "Smiles. 2. algorithm for generation of unique smiles notation," *Journal of chemical information and computer sciences*, vol. 29, no. 2, pp. 97–101, 1989.

[28] L. N. Vaserstein, "Markov processes over denumerable products of spaces, describing large systems of automata," *Problemy Peredachi Informatsii*, vol. 5, no. 3, pp. 64–72, 1969.

[29] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International conference on machine learning*. PMLR, 2021, pp. 8162–8171.

[30] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," *arXiv preprint arXiv:2011.13456*, 2020.

[31] J. Jo, S. Lee, and S. J. Hwang, "Score-based generative modeling of graphs via the system of stochastic differential equations," in *International conference on machine learning*. PMLR, 2022, pp. 10 362–10 383.

[32] X. Chen, J. He, X. Han, and L.-P. Liu, "Efficient and degree-guided graph generation via discrete diffusion modeling," *arXiv preprint arXiv:2305.04111*, 2023.

[33] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[34] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[35] H. Huang, L. Sun, B. Du, and W. Lv, "Learning joint 2-d and 3-d graph diffusion models for complete molecule generation," *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[36] Y.-L. Liao and T. Smidt, "Equiformer: Equivariant graph attention transformer for 3d atomistic graphs," *arXiv preprint arXiv:2206.11990*, 2022.

[37] J. Gasteiger, F. Becker, and S. Günnemann, "Gemnet: Universal directional graph neural networks for molecules," *Advances in Neural Information Processing Systems*, vol. 34, pp. 6790–6802, 2021.

[38] N. Thomas, T. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley, "Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds," *arXiv preprint arXiv:1802.08219*, 2018.

[39] E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré, and M. Welling, "Argmax flows and multinomial diffusion: Learning categorical distributions," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 454–12 465, 2021.

[40] A. Schneuing, Y. Du, C. Harris, A. Jamasb, I. Igashov, W. Du, T. Blundell, P. Lió, C. Gomes, M. Welling *et al.*, "Structure-based drug design with equivariant diffusion models," *arXiv preprint arXiv:2210.13695*, 2022.

[41] G. Crippen, "Note rapid calculation of coordinates from distance matrices," *Journal of Computational Physics*, vol. 26, no. 3, pp. 449–452, 1978.

[42] J. J. Irwin and B. K. Shoichet, "Zinc- a free database of commercially available compounds for virtual screening," *Journal of chemical information and modeling*, vol. 45, no. 1, pp. 177–182, 2005.

[43] S. Axelrod and R. Gomez-Bombarelli, "Geom, energy-annotated molecular conformations for property prediction and molecular generation," *Scientific Data*, vol. 9, no. 1, p. 185, 2022.

[44] A. K. Rappé, C. J. Casewit, K. Colwell, W. A. Goddard III, and W. M. Skiff, "Uff, a full periodic table force field for molecular mechanics and molecular dynamics simulations," *Journal of the American chemical society*, vol. 114, no. 25, pp. 10024–10035, 1992.

[45] G. Landrum, "Rdkit documentation," *Release*, vol. 1, no. 1-79, p. 4, 2013.

[46] D. Rigoni, N. Navarin, and A. Sperduti, "A systematic assessment of deep learning models for molecule generation," *arXiv preprint arXiv:2008.09168*, 2020.

[47] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski *et al.*, "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 929–947.

[48] W. A. Falcon, "Pytorch lightning," *GitHub*, vol. 3, 2019.

[49] H. Huang, L. Sun, B. Du, and W. Lv, "Conditional diffusion based on discrete graph structures for molecular graph generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, 2023, pp. 4302–4311.