Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Generative Modelling for Chemical Design

Author:
Paddy Cahalane

Supervisor:
Prof. Stefano Sanvito

A thesis submitted for the degree of
Doctor of Philosophy
School of Physics
Trinity College Dublin

July 2025

Declaration

I, Paddy Cahalane, hereby declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the Library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

Abstract

The discovery of novel molecules and materials with desirable properties is a fundamental objective of chemical design. This involves searching through a vast space of different possibilities, with an estimated 10^{60} plausible molecules of drug-like size. As the demand for increasingly effective drugs and materials never ceases to grow, traditional approaches relying on human intuition and trial-and-error are becoming insufficient for the task. Data-driven solutions offer a paradigm shift, whereby large chemical databases enable this vast space to be efficiently navigated. This thesis demonstrates that generative models, based on deep-learning architectures, are capable of inverse design of novel molecules with targeted physicochemical and quantum-mechanical properties.

In this work, two different conditional generative-modelling frameworks are proposed. Firstly, a generative adversarial network (GAN) is used to generate molecular conformations with selective total energy values. A local inversion algorithm is developed to convert the generated atomic descriptors into interpretable atomic structures. The second generative-modelling framework, MolGPT, employs a generative pre-trained transformer (GPT) model with text-based chemical representations. A new representation called MolBlox is proposed, with a corresponding method to invert the generated text into valid molecules. The model is conditioned using a variety of physicochemical and quantum-mechanical properties. In both frameworks, the generated molecules are verified as being structurally valid, with accurate property matching, as confirmed by density-functional theory (DFT) calculations.

This thesis concludes that these generative frameworks offer scalable solutions to accelerate chemical design. Published work on local inversion and ongoing manuscripts on conformation generation and MolGPT underscore their impact. These tools are considered a promising avenue of further research, demonstrating the viability of automated methods in the discovery of novel pharmaceuticals and materials.

Keywords: *Machine Learning, Inverse Design, Generative Modelling, Machine-Learning Interatomic Potentials, DFT*

Acknowledgements

Firstly, I would like to thank my supervisor, Prof. Stefano Sanvito. He has provided me with continual support and invaluable direction since the very beginning. In the five years since I began working with him on my undergraduate research project, generative modelling has seemingly taken over the world. I will be eternally grateful for the opportunities he has opened up for me.

I am thankful for all of the advice, encouragement, and companionship that I have received from members of the Computational Spintronics group, past and present. Special thanks are given to Dr James Nelson who introduced me to machine learning, and gave me the confidence to pursue a PhD. To Dr Matteo Cobelli, without whose support and collaboration I would have surely faltered. To Dr Rajashi Tiwari and Dr Urvesh Patil for their technical expertise, which was always been freely given. And to Eoin Monaghan, for his friendship.

My PhD would not have been possible with those working behind the scenes, they kept the wheels turning. I am grateful to the Irish Research Council who provided the funding for this project (Project ID: GOIPG/2020/1428). To Stefania Negra and to Aoife Hill, for their help in seeing me to the end.

I would like to express my immense gratitude to my friends and family, you all kept me going. To my uncle Declan, whom I consider more like a second older brother. To all my friends, for making life more sweet; the seven of you especially. To my brother Jack, who has always been an inspiring role model for me. To my lovely partner Mia, who has brighten my life immensely. I dedicate this thesis to my parents, who have tirelessly supported me throughout this long, long journey. I cannot thank you enough.

List of Publications

- Cobelli, M., Cahalane, P., & Sanvito, S. Local inversion of the chemical environment representations *Physical Review B* **106**, 035402 (2022).
- Cahalane, P., Cobelli, M., Nelson, J., & Sanvito, S. De novo generation of molecular configurations for targeted energy values. (In preparation).
- Cahalane, P., & Sanvito, S. DFT-targeted generative language modelling of molecules using a fragment-based representation. (In preparation).

Contents

Chapter 1

Introduction

Chemical design involves the creation of novel molecules and materials which have desirable qualities. This is a seemingly impossible task, with there existing as many as 10^{60} possible small molecules[1] [1]. To put the size of this vast space of different molecules into context, there are approximately 10^{80} atoms contained within the universe [2], and only 10^{8} molecules have thus far been synthesised [3]. Navigating through this vast space of candidate molecules, and finding those few with the superlative properties that are desired, can be thought of as the ultimate objective of chemical design. This task is made harder by the fact that human progress continually demands compounds with outstanding properties, above and beyond those currently known [4]. It is a pre-eminent needle in a haystack problem, microscopically small needles in an astronomically large haystack.

The traditional approach to chemical design heavily relies on the intuition of the researcher, developed over years of careful study, combined with trial-and-error. The efficacy of this slow and steady approach, deeply rooted in the scientific method, is undeniable. However, as the demand for increasingly effective drugs and materials never ceases to grow, the potential of this traditional design paradigm can be said to decrease. There are fewer and fewer low hanging fruit which can be easily obtained [5], with each new discovery requiring more effort than the last. Despite the ever increasing levels of investment into research and development, there has been consistently fewer drugs that actually enter clinical trials and reach markets successfully [6, 7]. This downward trend can be described using Eroom's law (reverse of Moore's law)

[1] A small molecule is typically defined as having an molecular mass less than 500 daltons.

where the number of new drugs approved, per billion US dollars invested, halves every nine years [5]. Reasons for this decrease, include the 'better than the Beatles' problem, where the efficacy of new prospective drugs must surpass that of existing drugs. Increasing safety regulations and fewer low hanging fruit are also contributing factors. Despite, a notable recent deviation away from Eroom's law due to better information availability and improved decision making [8], it is expected that the underlying problems will continue to stagnate drug development [9]. It is therefore essential, to create innovative and effective tools which can be added into our chemical design arsenal.

The advent of computers has brought many new opportunities for improved chemical design techniques. There are now large chemical repositories freely available online [10–14], each containing various different classes of chemical compounds. A very successful computational approach to chemical design is called virtual screening [15–18]. This involves searching through some curated dataset until candidates with suitable calculated properties are found. Some of these online repositories [11, 13, 14] also contain unsynthesised virtual compounds, created using combinatorial chemistry techniques [19]. They are used to dramatically increase the database size, allowing for high-throughput virtual screening [20–22]. This generally entails using a series of increasingly stringent selection criteria, in order to find candidates worthy of further study. However, there are limitations to the virtual screening approach. Only regions of chemical space explicitly included in the training dataset are able to be explored, which is restricted by the intuition of the researcher curating the dataset [23, 24].

In this age of big data, generative machine-learning models are an emerging alternate paradigm for chemical design [25, 26]. They offer an efficient, scalable, customizable approach that is data-driven. There exist various different generative-modelling architectures [27–29], but they all have a common objective: to generate compounds similar to those contained in the training dataset. They achieve this objective by learning the underlying probability distribution of the dataset, which can then be sampled from at will. Generative models allow chemical space to be navigated, potentially discovering novel compounds in previously unexplored regions [30, 31]. Another major advantage of generative models is that they can be conditioned on some additional property information [32, 33], allowing for selective generation. This follows an inverse design approach [4] where desired properties are first specified, and

then the algorithm directly outputs corresponding samples. This eliminates the need for exhaustive searches through vast combinatorial libraries in the hopes of finding a compound with the desired property.

As previously stated, the ultimate goal of chemical design is have the ability to selectively generate molecules or materials for some specified target property. This thesis only considers molecular systems, but similar methods can be theoretically applied to periodic systems. There exists a huge variety of suitable chemical properties that can be targeted. Among the most commonly used properties in generative modelling are the so-called physicochemical properties [30, 33–36], such as hydrophobicity [37], topological polar surface area [38], chemical synthesisability [39], and drug-likeness [40]. These four properties are used as target properties in Chapter 6. These physicochemical properties are of particular relevance to drug design. For example, the polar surface area can be used as an indicator of the ability of a molecule to permeate through the blood brain barrier, which is essential for any drug to access the central-nervous system [41]. Properties derived from quantum-mechanical electronic structure calculations [42–44] can also be targeted [30, 45–48]. The HOMO-LUMO gap, for example, is the energy difference between the highest occupied molecular orbital (HOMO) and the lowest unoccupied molecular orbital (LUMO) [49]. High values of this property may imply chemical stability, while lower values may indicate chemical reactivity. It also affects optoelectronic properties, determining which wavelengths of light can be absorbed [50]. Finding molecules with desirable values for the HOMO-LUMO gap is essential in the creation of novel organic semiconductors [51].

There exist many other classes of properties that are suitable for use as targets in generative modelling, but are not considered in this work. These include ADMET properties (adsorption, distribution, metabolism, excretion, toxicity), which are crucial to drug design [52, 53]. Generating molecules with reasonable ADMET properties [54–56] is vital in reducing attrition in preclinical and clinical trials [57], with toxicity being of particular importance [58]. Another class of suitable properties are those relevant to target-based drug design [59–65]. This involves the tailoring of molecules to interact with specific receptors or enzymes. Related properties include binding affinities [66] and inhibition concentrations [67]. Structural properties such as polar surface area [68], molecular volume [69], and molecular docking scores [70, 71]

also influence bioactivity.

In order to train a generative model to output novel molecules for selective property values, some essential ingredients are required. Of primary importance, as in any machine-learning application, is access to a large amount of good-quality data. Such a training dataset would consist of pairs of molecules and associated property labels. The molecules can be expressed in one of a number of different chemical representations [72–75]. In scenarios where property labels are expensive to produce, it can be beneficial to pre-train a generative model on a large, unlabelled dataset, before fine-tuning it on a smaller, labelled dataset [76]. The next essential component is a suitable generative modelling architecture, one that is compatible with the chosen chemical representation. Some modifications must be made to the model architecture [32, 33] to facilitate the conditioning on property labels. The final required component is some means of verification, of the structural validity of the generated molecules, as well as the accuracy of the property conditioning. The desired property values can be compared to the actual generated values to assess the success of the property conditioning of the model. This necessitates having access to a property calculator similar to that used in the creation of the training dataset. When inputted desired property values, a successfully trained generative model will output random molecules, similar to those included in the training dataset.

Outline

A brief outline of this thesis is provided here. Besides the introduction and the conclusion, there are five distinct chapters: two focused on methods and three on results. Considering that the title of this thesis is 'Generative Modelling for Chemical Design', the first methods chapter explains the machine-learning techniques involved in generative modelling, while the second methods chapters details the physics of chemical systems, and the related computational methods.

Chapter 2: This chapter introduces the machine-learning (ML) methods that are used throughout this work. The fundamentals of ML are first described, in a supervised-learning scenario where data labels are present. The inner workings of the now-ubiquitous neural networks are then detailed. The different generative models used in this thesis are then introduced: the generative adversarial network (GAN) [77], and the generative pre-trained transformer (GPT) [76, 78].

Chapter 3: This chapter seeks to introduce the physics of the systems to which generative modelling is applied. The relevant quantum mechanics (QM) [79] are described, followed by the electronic structure calculation method of density-functional theory (DFT) [80–82]. Machine-learned interatomic potentials (MLIAPs) [83, 84] are then described, which are a computationally efficient alternative to DFT for some properties. The various chemical representations and model architecture used in the construction of these MLIAPs are then detailed. Particular attention is given to many-body representations such as bispectrum components [75], due to their high accuracy in QM property prediction tasks [85].

Chapter 4: This first results chapter is based on work published in Reference [86]. A method is developed that takes a many-body representation [75] and performs an iterative optimisation procedure to find corresponding the Cartesian coordinates. This is an approximate method, which only claims to perform a local inversion, and can sometimes struggle with convergence. An improved method of initialisation using multiple different structures is proposed. The inversion method is verified by comparing radial and angular distributions functions. Additionally, electronic structures of the original and the inverted configurations are also compared.

Chapter 5: This chapter is the natural progression from the previous Chapter 4. It applies the developed local inversion method to its intended use case of gener-

ative modelling. A generative adversarial network (GAN) [77] is trained to output molecular configurations in the form of many-body descriptors [75], which can then be inverted into human-interpretable Cartesian coordinates. The generated configurations are compared to those from the training set using radial and angular distribution functions. Different versions of the GAN are trained using molecular dynamics datasets [87] containing single and multiple molecule types. Another GAN is trained, conditioned on density-functional theory (DFT) [80–82] properties [88], allowing for the generation of configurations for specific properties values.

Chapter 6: This chapter uses language modelling to generate diverse molecules for specified property values. A novel fragment-based textual representation called MolBlox is developed as an alternative to the standard SMILES representation [72]. An inversion scheme to reconstruct generated MolBlox sequences into valid molecules is detailed. The standard generative pre-trained transformer (GPT) [76,78] is modified to form MolGPT [33], allowing for conditioning on molecular properties [37–40]. The generated molecules are verified, after both pre-training on the GuacaMol dataset [89] and fine-tuning on the QM9 dataset [14, 90]. The accuracy of conditioning on DFT properties [50] is verified using the ETKDG conformer generation method [91,92] and the PySCF implementation of DFT [42–44].

Chapter 2

Machine Learning

The various machine-learning methods used in this work are detailed here. An introduction to the basics of supervised learning is followed by a description of the workings of neural networks. The two different generative model architectures used are then illustrated: generative adversarial networks (GANs) and generative pre-trained transformers (GPTs).

2.1 Introduction

Machine learning (ML) is a sub-field of artificial intelligence (AI) that uses probability theory and statistical methods to learn to make predictions based on data. It is an automated procedure that learns on its own, with no requirement for human intervention. Machine learning has been around since the 1950s and in that time has seen booms and busts in interest. Among the first prominent applications of ML was a solver for the game of checkers by Arthur Samuel [93]. ML and AI are experiencing a huge surge in interest in the current age, particularly due to generative models such as ChatGPT [94]. In order to define machine learning, it can be useful to first consider what learning is. A concise definition, provided by Tom Mitchell [95], is that learning occurs when performance P at a task T increases with experience E. It can therefore be said that three essential elements are needed in any learning algorithm, some quantifiable metric of performance, a clearly defined task or objective, and a large amount of good-quality data to provide experience. There exists a huge variety of ML algorithms, but a core concept common to all is that they seek to learn the

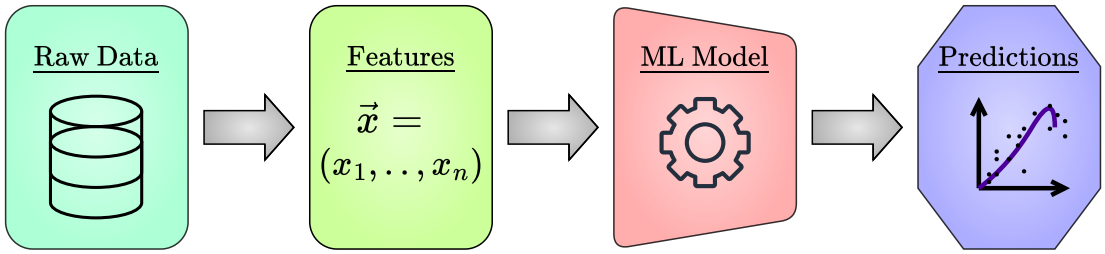

Figure 2.1: Diagram showing the basic workflow of a machine-learning algorithm. Raw input data is taken and preprocessed into the appropriately scaled feature vectors. The ML model is then trained on the dataset of feature vectors and once training is completed, the model predictions are made.

function that maps input data to output data. Data lies at the heart of all machine learning, enabling a model to learn to recognise patterns contained within and to extrapolate out onto previously unseen data.

It is common to separate out ML algorithms into distinct branches such as supervised, unsupervised, reinforcement, and generative. Each branch of ML contains many unique algorithms but they share a similar paradigm. Supervised models are among the most common and straightforward and are briefly illustrated below. This chapter primarily draws upon the textbook "Deep Learning" by Goodfellow [96]. The notation has been adapted for consistency. The generative models covered in Section 2.4 and Section 2.5 are largely based upon their original proposal papers [27,29,76,77].

2.2 Supervised Learning

A dataset consists of many individual examples or samples, where each sample can take a variety of forms such as a sentence, an image, a vector, or a matrix. The appropriate ML model can be determined based on what form each sample takes, and what the task is. The representation that is used to describe each sample is called a feature vector. For example, if the dataset consists of a series of images, the colour of each pixel would be a feature [97–99]. Supervised learning algorithms are used in the scenario where each dataset sample has a corresponding label or target. These targets might, for example, be binary labels signifying whether an image is of a dog or

a cat. The objective of the model is then to learn to predict the correct target given the feature vector, as shown in Figure 2.1. More formally, this can be considered as learning the mapping f from each input feature vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_{n_x})$ to output targets $\boldsymbol{y} = (y_1, y_2, \ldots, y_{n_y})$, where n_x is the number of features and n_y is number of targets for every one of the m samples contained in the dataset $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{m}$. The format of the samples being inputted into the model is important, with appropriate feature vectors often constructed from the raw samples in order to better facilitate learning [95]. Each target can either be a scalar y or a vector $\boldsymbol{y}$. Supervised learning tasks are often distinguished based on whether the targets are continuous (e.g., house prices) or discrete (e.g., cat or dog). If the targets are continuous, then the task is called regression and if they are discrete, it is then called classification [96]. The objective of a supervising learning model may be generalised as

$$f : \boldsymbol{x} \rightarrow \boldsymbol{y}\,. \tag{2.1}$$

The objective of a machine-learning model f is characterised by containing a set of learnable parameters or weights, $\boldsymbol{\theta}$, which are multiplied, in various ways, with the features $\boldsymbol{x}$ to yield the predicted targets $\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ [100]. During the training of the model, the weights $\boldsymbol{\theta}$ are tuned so as to minimise the difference between the true target $\boldsymbol{y}$ and the predicted target $\hat{\boldsymbol{y}}$ [101]. This difference is quantified by the loss function $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$ of a model, of which there are many possibilities depending on the task and on the format of the data [96]. The objective of a supervised learning algorithm is to minimise this loss function over all m samples contained within the training dataset. In a regression task, where targets are continuous, a mean-squared error (MSE) loss function would be appropriate. If however, the targets are discrete as in a classification problem, then a cross-entropy (CE) loss function might be used. In a classification problem, every ith target $\boldsymbol{y^{(i)}} = (y_1^{(i)}, y_2^{(i)}, \ldots, y_{n_y}^{(i)})$ is a vector representing a probability distribution with n_y categories. The MSE and CE loss functions [96] are defined as

$$L_{\text{MSE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{n_y} \sum_{j=1}^{n_y} \left(\boldsymbol{y}_j^{(i)} - \hat{\boldsymbol{y}}_j^{(i)} \right)^2, \tag{2.2}$$

$$L_{\mathrm{CE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n_y} \boldsymbol{y}_j^{(i)} \log(\hat{\boldsymbol{y}}_j^{(i)}) \,. \tag{2.3}$$

Regularisation

The learning process must be carefully monitored throughout, otherwise the performance of the model may start to become worse with additional training. This monitoring is typically done by separating out the dataset into different sub-datasets, each serving a distinct purpose. There different splits are commonly used: the training set, the test set, and the validation set [96]. The training dataset is used by the model to update its weights $\boldsymbol{\theta}$ to minimise the defined loss function. The test set is used to evaluate the performance of the model on unseen samples, with no update to the model weights. And lastly, the validation set is used to select the best from multiple different models, which all have been trained using different hyperparameters. These hyperparameters are parameters which must be defined before training, such as the model size and the learning rate. A model with sufficient learning capacity tends to overfit on the training set, essentially learning each individual sample, losing the ability to generalise [96]. Therefore, as the model is trained, its performance on the test set is monitored and the optimum model weights $\boldsymbol{\theta}^*$, which yield the best performance on the test set are chosen. Examples of underfitting, optimum fitting, and overfitting are shown in Figure 2.2. Overfitting can occur for numerous reasons including too many training iterations and using an overly complex model. Early stopping is commonly used, a process that halts the model training once the model performance on the test set no longer increases [102]. Most methods of preventing overfitting fall under the umbrella term of regularisation, which seeks to increase the model's generalisability at the cost of a small decrease in performance on the training set.

Another commonly used method of regularisation is to include an additional term into the loss function of the model that penalises weights that grow too large [96]. This has the effect of reducing the risk of overfitting on the training dataset by stopping the model from focusing on specific features. There is a coupling hyperparameter λ that controls the strength of the regularisation [84]. Perhaps the most typically used form of regularisation is called L_2 regularisation or ridge regression [103] where the

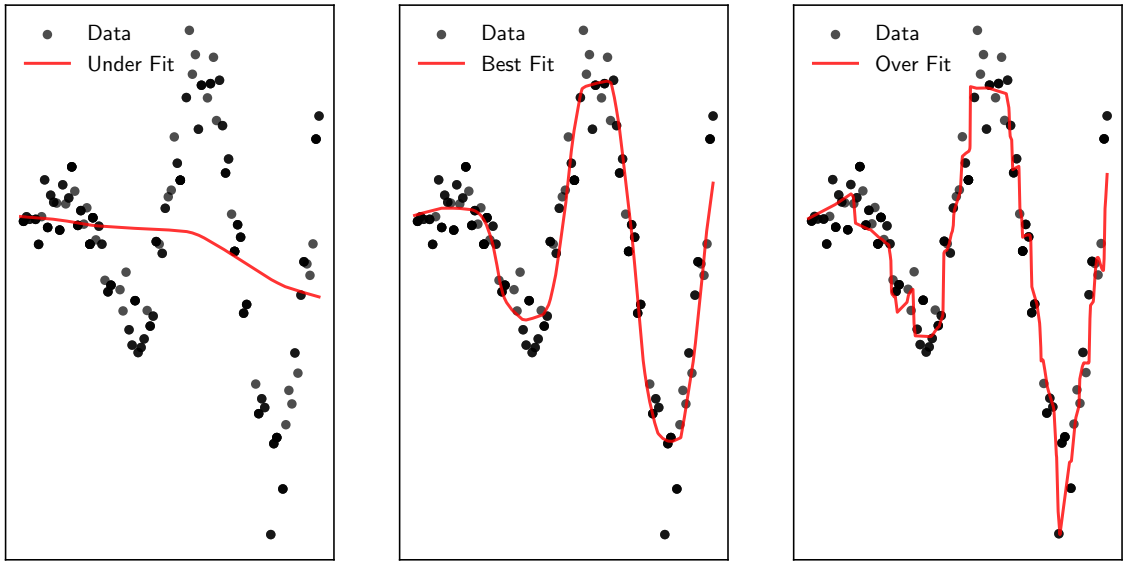

Figure 2.2: Diagram showing a machine-learning model (a deep neural network specifically) fitted to some random-noise polynomial data. The model predictions are shown in red. Early in training, the model exhibits underfitting (left). As training continues, the optimum fit is obtained (middle). Finally, the model overfits (right), learning individual samples instead the underlying data distribution.

squared L_2-norm is added to a loss function such as from Equation (2.2) as follows:

$$L_{\text{Reg}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = L_{\text{MSE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \lambda \sum_{\theta_i \in \boldsymbol{\theta}} \theta_i^2 , \tag{2.4}$$

where L_{Reg} is the new regularised loss function [104].

Gradient Descent

In order to obtain the optimal weights $\boldsymbol{\theta}^*$, some optimisation scheme must be employed to minimise the loss function L. Most machine-learning algorithms will use an optimisation scheme based on gradient descent [105]. Model optimisation is a broad field of study, meaning there are many different versions [106–108] of gradient descent to choose from. Gradient descent is an iterative procedure that seeks to update the model weights $\boldsymbol{\theta}$ such that the loss function L decreases at each iteration step. Considering that the model predicts the targets $\boldsymbol{y}$ as $\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$, any arbitrary loss

function can be restated as $L(\boldsymbol{\theta}) = L(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))$, if $\boldsymbol{x}$ and $\boldsymbol{y}$ are ignored as arguments. Optimising the weights $\boldsymbol{\theta}$ is achieved by considering the N-dimensional surface formed by the loss function $L(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. The gradient of $L(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ is $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$, which represents a vector pointing in the direction of steepest ascent along this surface. At each iteration, $\boldsymbol{\theta}$ is updated along the opposite of this vector as follows:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}), \tag{2.5}$$

where η is the learning rate, a hyperparameter controlling the size of the step that the gradient descent algorithm will take down the surface. Due to the nature of the performance metric employed, the surface formed by the loss function $L(\boldsymbol{\theta})$ is often non-convex, with many local minima contained within. Therefore, care must be taken to set an appropriate value for the learning rate η. The optimization procedure will be slow, if it is set too small, and if it is set too large, then $\boldsymbol{\theta}$ may oscillate around the local or global minima. There exist a large amount of advanced techniques for improving and stabilising the gradient descent procedure, learning rate schedulers, adaptive optimisation [106, 108], and gradient momentum [106, 107].

The number of samples m contained in the dataset is often too large to be computationally feasible to calculate $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ over all m samples at each step of the gradient descent procedure. This motivates the use of mini-batch gradient descent [109], where only a small subset of m_b samples are used to update $\boldsymbol{\theta}$ at each step. In the case where $m_b = 1$ it is then termed as stochastic gradient descent, but is rarely used due to the inherently high levels of noise. It has been found that using an appropriately sized mini-batch of samples provides the optimum balance between computational efficiency and convergence stability [109]. Using a mini-batch also introduces some stochasticity into the algorithm, which can help with overcoming local minima in the loss function surface [110]. Mini-batch gradient descent is therefore the de facto standard in most machine-learning applications. More formally, it updates the parameters $\boldsymbol{\theta}$ as in Equation (2.5) using an average of the gradient of the loss function as

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \frac{1}{m_b} \sum_{i=1}^{m_b} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}), \tag{2.6}$$

where $\nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta})$ is the gradient of the loss function $L(\boldsymbol{\theta})$ for each sample i in the mini-batch.

One epoch is said to have occurred when the model has been updated using all m training samples. This is equivalent to a total of m/m_b parameter updates as described by Equation (2.6). The number of epochs is a hyperparameter which controls the total training time, assuming that no early stopping regularisation halts the training first.

Feature Scaling

Feature scaling is an essential preprocessing step within the field of machine learning whereby the dataset values are shifted to all have similar ranges. This has the effect of giving all features an equal importance, restricting those features with higher values from dominating. For example, consider a regression task of predicting the price of a house y given features $\boldsymbol{x}$ such as square footage, number of bedrooms, etc. There would be orders of magnitude of difference between the values of square footage (in the 1000s) and number of bedrooms (around 3). This would artificially cause the model to give a much higher degree of importance to square footage than may be appropriate. Feature scaling eliminates this problem by ensuring that the distance between the maximum and minimum values for each feature are similarly far apart. The targets $\boldsymbol{y}$ are also scaled in a similar fashion. In most machine-learning models, proper learning will not take place unless appropriate feature scaling is performed. There are two primary methods of feature scaling commonly employed. The first is normalisation, and it involves rescaling data to be contained within a specific range, usually between zero and one. The second method is called standardisation, and it works by transforming data to have a mean of zero and a standard deviation of one.

Consider a dataset of samples $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}\}$ that has a total of N samples, with every ith sample $\boldsymbol{x}^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \ldots, x_{N_f}^{(i)}\}$ having N_f features. Feature scaling is applied independently to every feature j, calculating the necessary statistics using the jth features across all samples: $\boldsymbol{x}_j = \{x_j^{(1)}, x_j^{(2)}, \ldots, x_j^{(N)}\}$. The equations for normalisation and standardisation for each feature j are given respectively as

$$\boldsymbol{x}'_j = \frac{\boldsymbol{x}_j - \min(\boldsymbol{x}_j)}{\max(\boldsymbol{x}_j) - \min(\boldsymbol{x}_j)}, \tag{2.7}$$

$$\boldsymbol{x}'_j = \frac{\boldsymbol{x}_j - \mu}{\sigma}, \tag{2.8}$$

where $\boldsymbol{x}'_j$ is the feature-scaled version of $\boldsymbol{x}_j$, with μ mean and σ standard deviation [96].

2.3 Neural Networks

An artificial neural network (NN) is a mathematical model inspired by the physiology of the brain, and has become the most widely used machine-learning algorithm. A biological neuron is a type of cell contained with a brain in huge numbers, forming a dense network which communicates through electrical signals. Each neuron receives many input signals from other neurons it is connected to, processes the signals within the cell body, and transmits out one signal. An artificial neuron is a simple mathematical representation of these biological functions [100]. Many such neurons interconnected together is called a neural network. These computational models of biological neural systems were first proposed in the 1940's [111], but did not find any notable success until Rosenblatt et al. [100] trained a network capable of learning to perform binary classification. A so-called AI winter followed due in part to the lack of sufficient computational resources. A major advancement was made by Rumelhart et al. [101][1] in developing the backpropagation algorithm that allowed for the training of deep, multi-layer networks capable of learning complex, non-linear functions. It can be said that the great potential of deep learning was first demonstrated by Krizhevsky et al. [99][2], with was a confluence of deep networks, trained on huge datasets [98] in a parallel fashion [120].

The widespread use of neural networks comes from the fact that they are universal approximators, which means that a neural network with a sufficiently large number of learnable weights can approximate any measurable function to any desired degree of accuracy [121]. Other reasons for the success of neural networks are that they can

[1]The secondary author of this landmark paper is Geoffrey Hinton, winner of the Nobel Prize in Physics 2024. This is but one of the many landmark papers he has authored over the decades. He is cited numerous times throughout this thesis [99, 101, 112–119].
[2]Another revolutionary paper authored by Hinton!

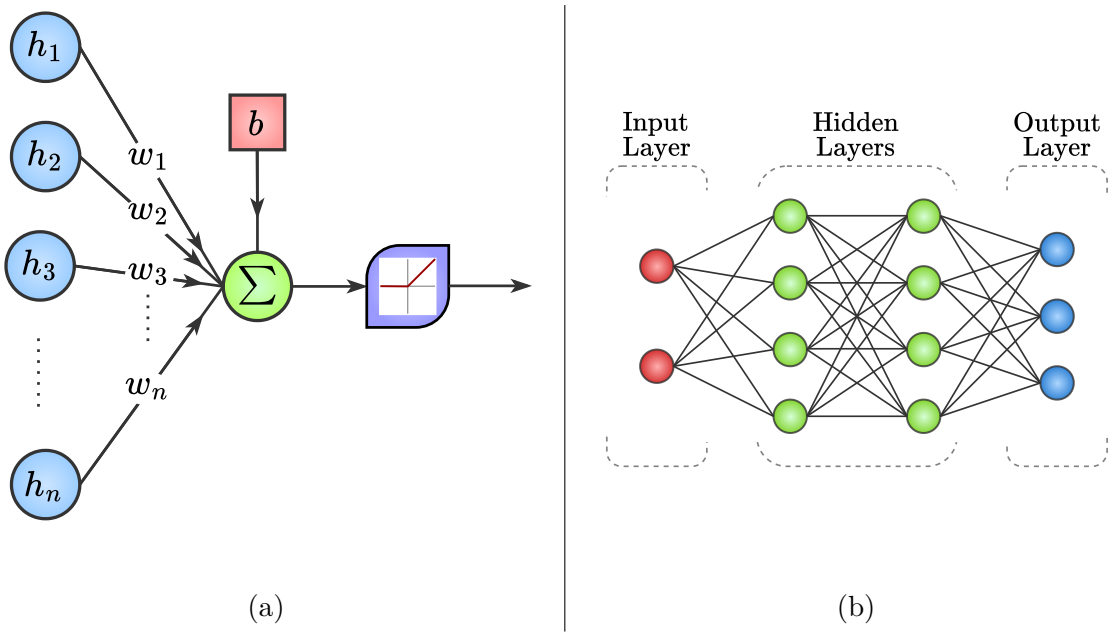

Figure 2.3: Diagram showing the fundamentals of neural networks. Panel (a) shows a single neuron which sums a scalar bias b term onto the dot product of the input activation vector $\boldsymbol{h}$ and weight vector $\boldsymbol{w}$. A non-linear activation function (ReLU) is then applied to form the scalar output activation. Panel (b) shows a diagram of a basic fully connected neural network with two input neurons, two hidden layers of four neurons each, and three output neurons.

generalise well to unseen test data, that they can be trained in a fully parallel manner, and that their training time scales linearly with the training set size. The current widespread adoption of neural networks stems from the availability of computational power in recent decades, which the graphics processing unit (GPU) [120] has been particularly noteworthy. Neural networks are most famously used in deep learning, which entails using a network with an input layer and an output layer and then an arbitrary number of hidden layers sandwiched in between. Deep neural networks serve as powerful machine-learning algorithms, which have been applied with great success to a wide range of problems including computer vision [97], natural language processing [29], speech recognition [122], reinforcement learning [123], and generative modelling [77].

Consider a dataset $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{m}$ which contains a total of m data pairs with feature vectors $\boldsymbol{x} = (x_1, x_2, \ldots, x_{n_x})$ and targets $\boldsymbol{y} = (y_1, y_2, \ldots, y_{n_y})$, where n_x is the number of features and n_y is the number of targets per sample. A deep neural network

consists of N_L sequential layers of neurons, with a total of $n^{(l)}$ neurons per layer l. The input layer of a deep neural network $f_{\boldsymbol{\theta}}$, with weights $\boldsymbol{\theta}$, takes the feature vectors $\boldsymbol{x}$, performs no operations, and feeds them as $\boldsymbol{h}^{(0)}$ into the neurons of the first hidden layer $l=1$. The number of neurons $n^{(l=0)}$ in the input layer must match the number of feature vectors n_x. Each neuron j in layer l takes inputs $h_k^{(l-1)}$ from each neuron k in from the previous layer $l-1$, performs a linear combination with learnable weights $\boldsymbol{w}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$, and passes the result through a non-linear activation function $\sigma^{(l)}$ with an output activation $h_j^{(l)}$. The activations vector $\boldsymbol{h}^{(l)}$ outputted by each layer l can be expressed as follows:

$$\boldsymbol{h}^{[0]} = \boldsymbol{x}\,, \tag{2.9}$$

$$\boldsymbol{h}^{(l)} = \sigma^{(l)}\Big(\boldsymbol{b}^{(l)} + \sum_{k=1}^{n^{(l-1)}} \boldsymbol{w}_k^{(l)} \cdot \boldsymbol{h}_k^{(l-1)}\Big)\,, \qquad 1 \leq l \leq N_L \tag{2.10}$$

$$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(N_L)}\,, \tag{2.11}$$

where $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$ is a bias parameter also commonly learned by each neuron to add flexibility in modelling complex functions. The input to activation function $\sigma^{(l)}$ can be succinctly expressed as $\boldsymbol{z}^{(l)}$. The total set of learnable parameters $\boldsymbol{\theta}$ include both the weights $\boldsymbol{w}$ and the biases $\boldsymbol{b}$ as $\{(\boldsymbol{w}^{(l)}, \boldsymbol{b}^{(l)})\}_{l=1}^{N_L}$. Each layer in the model $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is commonly termed as a linear layer, to which a non-linear activation function may be applied.

Activation Functions

Activation functions are critical to the success of neural networks and are responsible for their ability to learn complex patterns contained within the training data. The purpose of activation functions is to introduce non-linearities into the network, without which only simple linear functions could be described. There exists a whole zoo of different activation functions to choose from [119, 124–126], as shown in Figure 2.4. By far the most common choice for the activation function in the hidden layers is the rectified linear unit (ReLU) [119]. This ReLU function is said to be biologically inspired [127], replicating real neurons that either fire or do not fire by outputting zero for all negative inputs. Multiple variations of ReLU exist such as leaky ReLU

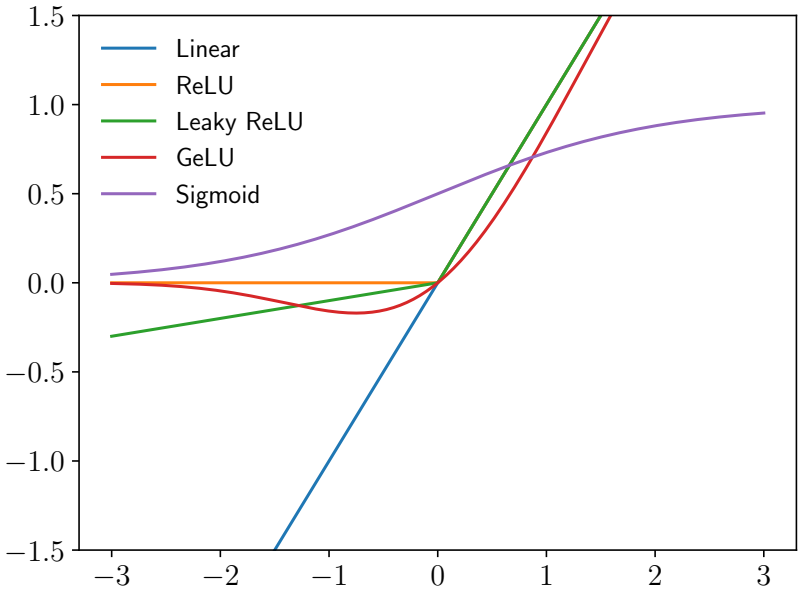

Figure 2.4: Diagram showing various common neural network activation functions that appear in this work.

(LeakyReLU) [125] where a small slope $\alpha \sim 0.1$ is defined in order to avoid issue of vanishing gradients caused by ReLU. However, the leak in LeakyReLU increases linearly which can cause undesirable effects if the neuron pre-activation output becomes sufficiently negative. This motivated the Gaussian error linear unit (GeLU) [124], which is smooth everywhere and still goes to zero as $z \to -\infty$. In the output layer L, the range of the outputted predictions $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \hat{\boldsymbol{y}}$ must match that of the true targets $\boldsymbol{y}$. Therefore, the final activation function is often left linear, leaving the model unconstrained and able to match the range of $\hat{\boldsymbol{y}}$ to $\boldsymbol{y}$. In classification tasks, the softmax activation function [128] is often used to assign each class i of n_y total classes to a probability distribution. Similarly, the sigmoid activation function [126] is used to assign a probability between $[0, 1]$ for binary classification tasks as well as being used for regression tasks where $y_i \in [0, 1]$. These activation functions are now defined:

$$\text{ReLU}(z) = \max(0, z)\,, \tag{2.12}$$

$$\text{LeakyReLU}(z) = \max(\alpha z, z)\,, \tag{2.13}$$

$$\text{GeLU}(z) = z * \Phi(z)\,, \tag{2.14}$$

$$\text{Softmax}(z_i) = e^{z_i} / \big(\sum_{j}^{n_y} e^{z_j} \big)\,, \tag{2.15}$$

$$\text{Sigmoid}(z) = (1 + e^{-z})^{-1}\,, \tag{2.16}$$

where Φ is the cumulative distribution function for the Gaussian distribution and z is the pre-activation output of a neuron.

Backpropagation

A neural network $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \hat{\boldsymbol{y}}$ is machine-learning algorithm that seeks to improve its performance as measured by the loss function $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$ by optimising its tunable weights $\boldsymbol{\theta}$ through gradient descent. The gradient descent method depends on taking the derivative of the loss function with respect to each parameter $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))$ as shown in Equation (2.5). This derivative $\nabla_{\boldsymbol{\theta}} L$ must be calculated with respect to every learnable parameter $\{\boldsymbol{\theta}_i\}_{i=1}^{N_{\boldsymbol{\theta}}}$, where $N_{\boldsymbol{\theta}}$ is the number of parameters. The feature vector $\boldsymbol{x}$ is propagated through the model in the forward pass as described in Equations (2.9) through (2.11). In the backward pass, the gradients $\nabla_{\boldsymbol{\theta}} L$ are calculated for each layer subsequently using the chain rule. The pre-activation output of each neuron in layer l can be defined as $\boldsymbol{z}^{(l)}$, simplifying Equation (2.10) to $\boldsymbol{h}^{(l)} = \sigma^{(l)}(\boldsymbol{z}^{(l)})$. A loss function such as mean-squared error from Equation (2.2) might be used, where $L = (\boldsymbol{y} - \boldsymbol{h}^{(L)})^2$ in the case of a single sample $m = 1$. The gradients of the loss function L with respect to the input :

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial \boldsymbol{h}^{(N_L)}} \cdot \frac{\partial \boldsymbol{h}^{(N_L)}}{\partial \boldsymbol{z}^{(N_L)}} \cdot \frac{\partial \boldsymbol{z}^{(N_L)}}{\partial \boldsymbol{h}^{(N_L-1)}} \cdot \frac{\partial \boldsymbol{h}^{(N_L-1)}}{\partial \boldsymbol{z}^{(N_L-1)}} \cdot \ldots \cdot \frac{\partial \boldsymbol{z}^{(1)}}{\partial \boldsymbol{x}}, \tag{2.17}$$

where N_L is the number of layers. The gradients of L flow back through the network in this fashion. Using the appropriate gradient terms at each layer l, the gradients of the loss function L with respect to the weights $\boldsymbol{w}^{(l)}$ and the biases $\boldsymbol{b}^{(l)}$ can then be calculated as

$$\frac{\partial L}{\partial \boldsymbol{w}^{(l)}} = \frac{\partial L}{\partial \boldsymbol{z}^{(l)}} \cdot \boldsymbol{h}^{(l-1)}, \tag{2.18}$$

$$\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial L}{\partial \boldsymbol{z}^{(l)}}. \tag{2.19}$$

All parameters $\theta = \{\boldsymbol{w}, \boldsymbol{b}\}$ can then be updated using Equation (2.5). This parameter update forms the crucial step of the gradient descent algorithm.

2.4 Generative Adversarial Networks

The majority of machine-learning models seek to learn to predict some quantity from samples contained in the training dataset. After the model has learned on the training dataset, it is applied to previously unseen test data, allowing new insights to be gained. In contrast, generative models obey a different paradigm, seeking to learn to generate samples similar to those in the training dataset. They primarily function by approximating the underlying dataset distribution, which can then be sampled from at will. After training has concluded, the generation stage or inference begins, where this approximate distribution is sampled from. Generative models are typically able to generate samples much faster at a much lower cost, be it a computational cost or otherwise, than traditional methods [129]. The two main types of generative models that are used in this work are (1) generative adversarial networks (GANs) [77] and (2) transformers [29, 76].

The foundations for the field that would become generative modelling was laid in the mid-20th century, with early work on probabilistic and statistical modelling, such as Shannon's theory of information [130] and the Metropolis algorithm [131]. In the subsequent decades there were some preliminary successes in generative modelling using Markov chains [130, 132, 133], Boltzmann machines [114, 115], and autoencoders [101, 113]. It was not until 2014 with the introduction of GANs by Ian Goodfellow [77] that the true potential of the field of generative modelling was demonstrated. Since then, an explosion of interest has occurred, with the state-of-the-art progressing rapidly each year. Deep convolutional GANs in 2015 [134] and Progressive GANs [135] in 2017 both represented major advancements. Newer models have since been developed that have surpassed the performance of GANs in pure generative-modelling tasks such as normalizing flows [136, 137] and diffusion models [28, 138], but GANs are still commonly used, especially when incorporated into hybrid models [139–142]. GANs still perform at the state-of-the-art level in some domains such as text to speech synthesis [143]. The last few years has also seen the rapid rise of transformer models [29, 76, 78, 144, 145], which were initially limited to the field of natural language but have since been successfully applied to computer vision tasks as well [97]. Transformer-based large language models (LLMs) have since become synonymous with artificial intelligence (AI), being the most prominent machine-learning application that is used

on a daily basis by non-experts. The generative models utilised in this work primarily include GANs and transformers.

GANs consist of two separate sub-models that play a zero-sum game [146] against each other [96]. These sub-models are:

Generator: a model that creates samples similar to those in its training dataset.

Discriminator: a model that classifies samples as being either real (from the training dataset) or fake (generated by the Generator).

The GAN is said to have converged when the generator G produces samples that the discriminator D is unable to distinguish from those in the training dataset. When successfully converged, G is considered to have learned a distribution p_g that approximates the underlying training data distribution p_{data}. A GAN is called a latent variable model, where G seeks to learn the mapping from an arbitrary latent space vector $\boldsymbol{z} \in \mathbb{R}^d$ of dimension d to the vector space of training samples $\boldsymbol{x}$:

$$G : \boldsymbol{z} \to \boldsymbol{x}\,. \tag{2.20}$$

This latent space can be thought of as a prior distribution p_z from which to sample $\boldsymbol{z}$, typically from either a standard Gaussian distribution $z_i \sim \mathcal{N}(0,1)$ or from a uniform distribution $z_i \sim \mathcal{U}(0,1)$ for each component i out of dimension d. The discriminator D has a scalar output quantifying its belief in whether $\boldsymbol{x}$ came from p_{data} or from p_g. Both G and D are typically neural networks with weights θ_g and θ_d, but any other differentiable machine-learning model could be used. In the results presented in this work, only deep, fully connected neural networks are used. Other model types, such as convolutional neural networks [99, 109, 147] are commonly used in high-dimensional computer vision tasks [134, 135, 148].

2.4.1 Vanilla GANs

The original GAN architecture presented in 2014 [77] is commonly referred to as the 'vanilla GAN'. This is done in order to differentiate it from the numerous subsequent versions. The fundamentals of the GAN architecture have remained the same, two

neural networks competing in a zero-sum game [146] in order to learn the underlying data distribution p_{data}. The output activation function of the discriminator D is a sigmoid function, see Equation (2.16), which quantifies the network's belief that the inputted sample is either real or fake as a probability between zero and one, with one being real and zero being fake.

The defining feature of the vanilla GAN is the use of the Jensen-Shannon (JS) divergence [149] to measure the difference between the real data distribution p_{data} and the generated data distribution p_g at each training iteration. The loss functions for G and D are respectively given as

$$L_G = -\mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_z}[\,\log(1 - D(G(\boldsymbol{z})))]\,, \tag{2.21}$$

$$L_D = -(\mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_z}[\log(1 - D(G(\boldsymbol{z})))] + \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p_{\text{data}}}[\,\log(D(\boldsymbol{x}))])\,, \tag{2.22}$$

where the notation $\mathbb{E}$ means the expectation value or average value over the entire mini-batch. Equation (2.22) is based on the binary cross-entropy (BCE) loss. This is a version of the cross-entropy loss from Equation (2.3), where there are only two possible class labels $y \in \{0, 1\}$ and a predicted class $\hat{y} \in (0, 1)$. Despite the BCE loss being used in Equation (2.22), the GAN is still effectively minimizing the JS divergence between p_{data} and p_g. A commonly made modification to vanilla GANs to try to avoid the problem of vanishing gradients, is to train G to instead minimise $-\mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_z}[\log(D(G(\boldsymbol{z}))]$ [150].

2.4.2 Wasserstein GANs

GANs have been reformulated and improved numerous times since being originally proposed. Using Wasserstein GANs (WGANs) [27] has become standard practice in order to address mode collapse, which is a common problem in vanilla GANs. Mode collapse is characterised by the variation in the samples outputted by G becoming drastically reduced. This occurs when D cannot distinguish the real from the generated, meaning that negligible gradients are produced, which effectively halts all learning. WGANs address this problem of mode collapse by changing the loss function metric from the original Jensen-Shannon divergence [149] to a Wasserstein loss (also known as Earth Mover's distance) [151]. The Jensen-Shannon divergence is

noted to contain some undesirable properties that contribute to the problem of mode collapse. These properties include not being a true metric that satisfies the triangle inequality [27], being bounded between 0 and log(2), having no notion of the distance between two non-overlapping distributions. The Wasserstein distance does not have these unwanted properties and also benefits from being continuous everywhere and differentiable almost everywhere [151]. This allows for much more stable training of both D and G, while effectively eliminating the risk of mode collapse. The new and improved loss functions of G and D in a WGAN are, respectively,

$$L_G = -\mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_z}[D(G(\boldsymbol{z}))], \tag{2.23}$$

$$L_D = \mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_z}[D(G(\boldsymbol{z}))] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p_{\text{data}}}[D(\boldsymbol{x})], \tag{2.24}$$

where D is required to be a 1-Lipschitz function [152] due to the Kantorovich-Rubinstein duality [151]. Lipschitz functions are a family of continuous differentiable functions where the absolute value of the slope never exceeds some defined maximum, which in this case is one. The output of D in a WGAN uses a linear activation function, no longer representing a probability score, instead quantifying its belief that the inputted samples are from the real distribution p_{data} or from the fake distribution p_g. D seeks to maximise the Wasserstein distance between each distribution by assigning larger positive scores to real samples and larger negative score to fake samples. Some form of regularization must be applied to enforce D to be a 1-Lipschitz continuous function. If not done correctly then the model gradients will explode in magnitude while maximising the outputted Wasserstein score. This can be avoided in two main ways, by weight-clipping [27] or by adding a gradient-penalty term to Equation (2.24) [153]. Weight clipping simply means to clamp the weights θ_d of D to the interval $\theta_d \in [-c, c]$ where c is some small constant, typically $c = 0.01$. This works by effectively constraining the magnitude of the output of D but it also negatively reduces the learning capacity of the network. The more sophisticated approach of introducing a gradient penalty term that penalises the gradients of D that deviate too much from a norm of one. It is shown to improve performance over weight clipping, but with a drawback of longer training times due to needing an additional backward pass to calculate the required gradients [153]. The gradient penalty term

is added with a weighting hyperparameter λ_{gp} to the discriminator loss function from Equation (2.24) as follows:

$$L_D = \mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_z}[D(G(\boldsymbol{z}))] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p_{\text{data}}}[D(\boldsymbol{x})] + \lambda_{\text{gp}} \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim p_{\tilde{x}}}[(||\nabla_{\tilde{\boldsymbol{x}}} D(\tilde{\boldsymbol{x}})||_2 - 1)^2], \tag{2.25}$$

where $\tilde{\boldsymbol{x}}$ are interpolated samples drawn from the distribution $p_{\tilde{x}}$ formed along the linear interpolation between pairs samples from p_{data} and p_g. This random linear interpolation is performed as $\tilde{\boldsymbol{x}} = \gamma \boldsymbol{x} + (1-\gamma)\boldsymbol{x}_g$ where γ is a random number drawn from the uniform distribution $\gamma \sim \mathcal{U}(0,1)$. The recommended default [153] of $\lambda_{\text{gp}} = 10$ is used throughout this work.

2.4.3 Conditional GANS

Consider a dataset, where every sample $\boldsymbol{x}$ has an associated label $\boldsymbol{y}$. A vanilla GAN has no way of making use of these additional data labels. One can imagine a scenario, where it is desirous to have the ability to generate samples $\boldsymbol{x}_g$ for a specified target label $\boldsymbol{y}_g$. These target labels $\boldsymbol{y}_g$ can be user-defined or randomly drawn from the same distribution as the training set labels p_y. For example, in image generation, some text-based prompt $\boldsymbol{y}_g$ is used to generate a corresponding image $\boldsymbol{x}_g$. Conditional generative models faciliate this selective generation, with conditional GANs (CGANs) [32] taking $\boldsymbol{y}$ as a secondary input into both $G(\boldsymbol{z}, \boldsymbol{y})$ and $D(\boldsymbol{x}, \boldsymbol{y})$. It this original formulation of conditional GANs [32] $\boldsymbol{y}$ is simply just concatenated onto the inputs $\boldsymbol{z}$ and $\boldsymbol{x}$, with no further modifications. This method performed reasonably well, but various improvements were subsequently made. One prominent innovation was made by auxiliary classifier GANs (ACGANs) [139], where a secondary task of predicting $\boldsymbol{y}$ as $\hat{\boldsymbol{y}}$ was integrated into D alongside the primary task of evaluating $\boldsymbol{x}$ as real or fake.

The labels $\boldsymbol{y}$ can represent discrete classes such as 'robin, hummingbird, nail,...' as in the ImageNet dataset [98] or they can be continuous values such as for molecule properties. Suitable properties include electronic observables such as total energy, HOMO-LUMO gap[3], and dipole moment [50]. In this work, one of the primary purposes of using conditional GANs is to allow for the selective generation of molecules, which have specific values of a given target property. Since the properties considered

[3]The HOMO-LUMO gap energy is not an actual observable, but is commonly used as a computational proxy for real observables such as electron excitation energy.

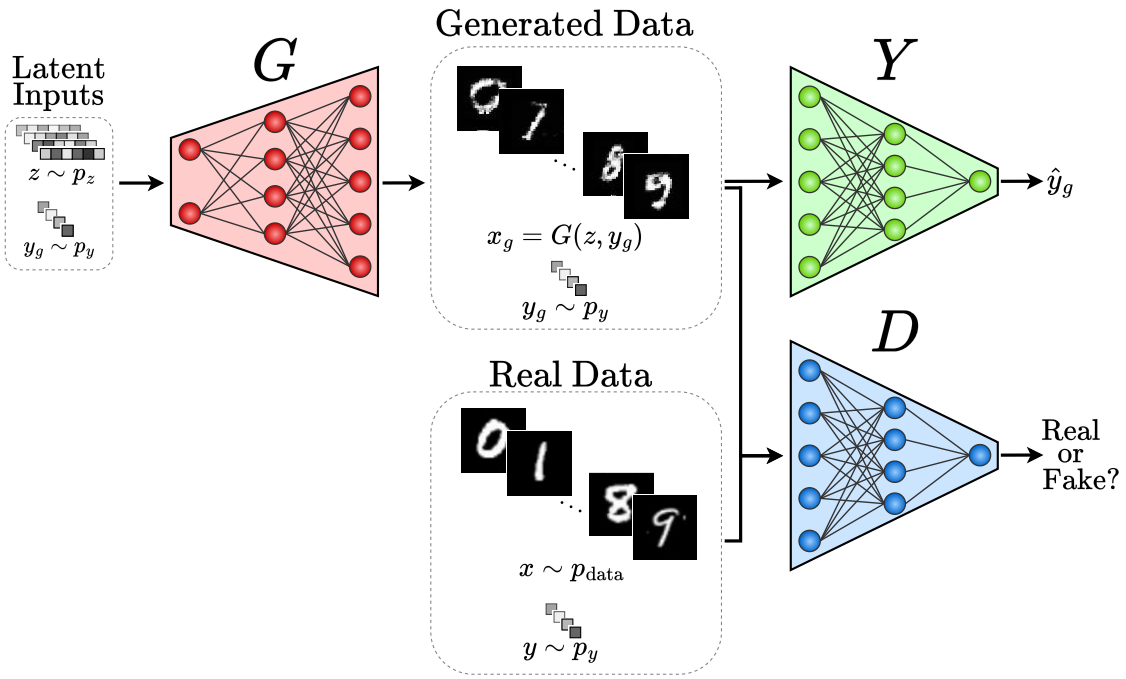

Figure 2.5: Diagram showing the conditional GAN architecture used in this work. The MNIST dataset of handwritten digits [109] is used. The generator G takes latent vectors $\boldsymbol{z}$ and target labels $\boldsymbol{y}_g$ as inputs and outputs a generated image $\boldsymbol{x}_g$ that corresponds to $\boldsymbol{y}_g$. The discriminator D is alternately inputted real images $\boldsymbol{x}$ and generated images $\boldsymbol{x}_g$. The score outputted by D signifies its belief in whether the inputted $\boldsymbol{x}$ is real or generated (fake). An additional network Y, that is pre-trained on real data, predicts the class label $\hat{\boldsymbol{y}}_g$ given generated images $\boldsymbol{x}_g$. A secondary training objective of minimising the difference between $\boldsymbol{y}_g$ and $\hat{\boldsymbol{y}}_g$ is added to the GAN.

are all continuous labels $\boldsymbol{y}$, it means that most conditional GAN architectures are unsuitable as they are designed for use with discrete labels. This motivated research into other possible architectures.

It is possible to assign the primary task (real or fake) and the secondary task (predicting labels) to different models: a standard discriminator $D(\boldsymbol{x}, \boldsymbol{y})$ and a new label prediction model $Y(\boldsymbol{x}) = \hat{\boldsymbol{y}}$. The architecture of this style of conditional GAN is shown in Figure 2.5. This would cause D to lose out on possible performance gains from the additional task [154], but allows for specialised architectures to be used for Y and the ability to pre-train Y on real samples before adversarial training [88]. This

can be thought of as a multiple discriminator GAN, of which there are many variants depending on the nature of the labels $\boldsymbol{y}$ [155–157]. There are other variants which add an additional term onto the loss function G from Equation (2.23) [158, 159]. This forces G to learn to generate samples that tend to have lower values of this extra loss term.

The approach taken here expands upon prior work [88, 160, 161], where just such a label prediction model $Y(\boldsymbol{x}) = \hat{\boldsymbol{y}}$ is incorporated into the workflow as an additional term in the generator loss function[4]. If $\boldsymbol{y}$ represents discrete class labels then Y would be a classification model. If $\boldsymbol{y}$ are instead scalar values, then Y is a regression model. The nature of the additional loss term that is added to the generator loss function of Equation (2.23) depends on whether Y is a classification or a regression model. In the example shown in Figure 2.5, where Y seeks to predict the handwritten digit shown in each image, a cross-entropy loss term, like in Equation (2.3), is appropriate. If Y is a regression model, then a mean-squared error (MSE) loss term is required, as in Equation (2.2).

Assuming that $\boldsymbol{y}$ contains continuous scalar values, the difference between the predicted $\hat{\boldsymbol{y}}_g$ of the generated samples $\boldsymbol{x}_g$ and the target $\boldsymbol{y}_g$ is summed onto the generator loss function using a weighting hyperparameter λ_{MSE} as follows:

$$L_G = -\mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_z}[D(G(\boldsymbol{z}))] + \lambda_{\text{MSE}} \mathop{\mathbb{E}}_{\boldsymbol{y}_g\sim p_y}[(\boldsymbol{y}_g - \hat{\boldsymbol{y}}_g)^2], \tag{2.26}$$

where $\hat{\boldsymbol{y}}_g = Y(G(\boldsymbol{z}))$ and λ_{MSE} defaults to a value of 10. Training a GAN using such a loss function will force G to output samples with predicted labels $\hat{\boldsymbol{y}}$ that align with target labels $\boldsymbol{y}_g$. Depending on the nature of the dataset, it may be possible to determine, computationally or otherwise, the actual true labels $\tilde{\boldsymbol{y}}$ of the generated samples $\boldsymbol{x}_g = G(\boldsymbol{z})$. The efficacy of this proposed method of conditioning a GAN can then be estimated by the difference between the desired label $\boldsymbol{y}_g$ and the actual label $\tilde{\boldsymbol{y}}$. This extra verification step is necessary in order to account for the error introduced by using a ML model Y to predict $\hat{\boldsymbol{y}}$. It is assumed that the method of determining $\tilde{\boldsymbol{y}}$ is much more expensive than Y, and is therefore not viable for inclusion within the

[4]Prior work was undertaken by this author as an undergraduate project where 2D Ising model [162, 163] lattices were generated using a conditional GAN, similar to [88]. Pre-trained regression models for temperature and energy (energy was analytically calculated) were incorporated into the GAN. Additional mean-squared error terms were added to the loss function of G so as to enforce the generated lattices to align with the desired property values.

GAN inner training loop[5]. For example, in the case of images of handwritten digits as shown in Figure 2.5, it must be verified that using a target label of $y_g = 2$ actually causes the generated image $\boldsymbol{x}_g$ to display a number 2. In this scenario, the true labels $\tilde{\boldsymbol{y}}$ can be determined by eye, but in other cases rigorous calculations may be needed.

2.5 Transformers

It seems that the history of machine learning is defined by periodic major breakthroughs that lead to an explosion of interest and research. Examples include LeNet5 from 1998, used for handwritten digit recognition [109], and AlexNet from 2012 [99] which influentially demonstrated the efficacy of deep learning, facilitated by parallel training on GPUs [120] using large quantities of labelled data [98]. The introduction of transformers in the seminal paper by Vaswani et al. [29] is undoubtedly just such a breakthrough. It has revolutionised the field of natural language processing and is largely responsible for AI becoming an practical tool accessible to even those without technical expertise.

The original transformer architecture [29] consisted of two separate models: the encoder and the decoder, with an intended use case of machine translational. The encoder model would take sentences in English as input, and then use a novel attention mechanism to create context-rich embedding vectors, which the decoder model would then use to output the appropriate French translation. However, it was quickly noted that the proposed attention mechanism, used to quantify relationships between symbols in a sequence (e.g., words in a sentence), had great potential for more than just machine translation. Widespread experimentation ensued, with encoder-only models [145, 164, 165], decoder-only models [33, 78, 97, 166], and encoder-decoder variants [167, 168] all being proposed. For tasks which involve extracting meaningful insights from sequences, then encoder-only models are more appropriate. Such tasks may include text classification [169] or information extraction [170]. The most well known encoder-only transformer is BERT (Bidirectional Encoder Representations

[5]To put this verification process within context, assume that $\boldsymbol{x}_g$ are generated 3D molecules and Y predicts molecule energies $\hat{\boldsymbol{y}}$. An electronic structure code [42–44] that implements density-functional theory (DFT) [80–82], can be used to determine the actual energies $\tilde{\boldsymbol{y}}$ of the generated molecules $\boldsymbol{x}_g$. Comparing $\tilde{\boldsymbol{y}}$ to the desired or target energies $\boldsymbol{y}_g$ gives a good estimate of how accurate the conditioning is.

from Transformers) [145]. These models can be adapted to different downstream tasks by fine-tuning with different output layers and by reformatting the data.

Transformers fundamentally work by processing sequences of tokens into a context-aware representation. Each layer in the model seeks to include more and more contextual information. It can be useful to develop an intuition about what is actually meant by the context of a symbol or a token within a sequence. Consider the meaning of the word 'it' in the following sentence: 'she poured water from the vase into the cup until it was empty'. It is clear to us that 'it' refers to the vase, which becomes empty. There can be said to be a strong relationship or correlation between the words 'vase', 'it', and 'empty'. However, consider a new sentence: 'she poured water from the vase into the cup until it was full'. Now the word 'it' instead refers to the 'cup', which becomes 'full', with a strong relationship between these three words. The attention mechanism seeks to quantify the strength of these relationships as a vector. As the model is trained, it will learn and update these vectors so as to capture both the semantic meaning of each word and also the context of each word within the sentence. Similar logic applies to any type of sequence of tokens, not just words in a sentence.

This body of work is concerned with generative modelling and therefore only the most suitable variant of transformers will be discussed: the Generative Pre-trained Transformer (GPT). The GPT model is a decoder-only transformer, first proposed by Radford et al. [78] of OpenAI. Multiple different iterations of the GPT model have been trained, with minor architecture differences, on successively larger datasets [76, 144, 171]. The number of learnable weights has increased dramatically at each iteration, from 117 million in GPT1 [78] to over one trillion in GPT4 [171]. This massive increase is a testament to the fact that deep learning has worked; scaling a neural network allows it to become predictably better at learning any arbitrary function [172].

2.5.1 Generative Language Modelling

The GPT model is a generative language model, which means that its objective is to predict the next token in a sequence of tokens. A token can represent any arbitrary symbol in a sequence, such as a word in a sentence [78] or a visual patch in an image [97]. By learning to predict the next token, the GPT model approximates

the underlying probability distribution of tokens. In a natural language scenario, the goal is to have the ability to generate coherent, human-like text that obeys the laws of grammar and is contextually relevant. One of the primary differences between encoder-only and decoder-only models, is that decoder-only models (such as GPT), are trained in an autoregressive fashion. This means that at each token prediction step, the model can only see the preceding tokens, all succeeding tokens are masked. This intuitively corresponds to how someone might go about memorising some body of text: by first concealing every word, then progressively unmasking one word at a time, reciting and learning from their mistakes as they go. A GPT model essentially does just this, it predicts the next token in a sequence, adds it on to the end, and repeats until the end of the sequence in reached. After each prediction, the output of the model is fed back in as a new input, which is what is meant by autoregressive or causal learning. An encoder-only model can see all tokens to the left and right when making a prediction, which allows for insightful contexts but poor sequence generation.

More formally, consider a dataset that contains m samples $(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)})$, where each sample $\boldsymbol{x}$ consists of a sequence of tokens $(t_1, t_2, \ldots, t_n)$ of variable length n. The set of all unique tokens $\mathcal{V}$ contained within the dataset is called the vocabulary. Given a partial sequence of tokens $(t_1, t_2, \ldots, t_j)$ of length j, where $j < n$, the training objective of a GPT model is to correctly estimate the probability that any token in the vocabulary $\mathcal{V}$ could be the next token t_{j+1}. Using this probability distribution, an appropriate next token t_{j+1} is then randomly sampled from $\mathcal{V}$. It is assumed that sequences have an implicit natural ordering, with the probability of a sequence $\boldsymbol{x}$ occurring being modelled as the chained probabilities of all the preceding tokens [173, 174]. The probability $P(\boldsymbol{x})$ of a sequence $\boldsymbol{x} = (t_1, t_2, \ldots, t_n)$ occurring can therefore be expressed as

$$\begin{aligned} P(\boldsymbol{x}) &= P(t_1)\, P(t_2|t_1)\, P(t_3|t_1, t_2)\,, \ldots, P(t_n|t_1, t_2, \ldots, t_{n-1})\,, \\ &= \prod_{i=1}^{n} P(t_i|t_1, t_2, \ldots, t_{i-1})\,, \end{aligned} \tag{2.27}$$

where $P(t_2|t_1)$ is the probability of token t_2 after token t_1. Equation (2.27) is fundamental to all generative language models and is therefore worthy of note.

Tokens

The training dataset for a transformer consists of sequences of symbols or tokens, which represent different things depending on the task. The actual transformer input is a sequence of integers, where each integer is the index of the corresponding token within the vocabulary $\mathcal{V}$. A token could represent a musical note in a song [166], a visual patch within an image [97,175], or a chemical bond of a molecule [33]. To apply generative language modelling to the English language, a naive approach would be to simply include every word and every punctuation mark as a separate token within the model vocabulary. However, there are hundreds of thousands of words in the English language, resulting in a prohibitively large model and vocabulary size. This is due to the fact that the output layer of a decoder model is a linear layer (a fully connected neural network layer) with one neuron for every token in $\mathcal{V}$. Similarly, if individual characters were used as tokens, then the training of the model would become too expensive due to the increased sequence length and the lack of semantic information contained in a character-level representation.

It is advantageous to break a raw sequence of tokens into more manageable pieces in order to aid learning. A subword tokenisation algorithm such as byte-pair encoding (BPE) [176, 177] does just this, by essentially compressing the dataset. Once the desired vocabulary size is defined, the BPE algorithm will calculate the frequency of each character in the dataset, and will iteratively merge common pairs of consecutive characters until the specified vocabulary size is reached. It stores these merged characters in a lookup table, which is used to tokenise any further sequences downstream. All sequences generated by the model will be in this compressed form, and will need to be 'detokenised' in order to become human interpretable. A word-level tokenisation algorithm would struggle with out-of-vocabulary words, assigning them as unknown <unk> tokens, while a character-level tokeniser would perform well. BPE seeks to strike a balance between word-level and character-level tokenisation, by being reliable with few <unk> tokens, while still retaining computationally efficiency and semantic expressivity.

A number of additional special tokens are commonly defined and used for specific purposes. These include the aforementioned unknown token <unk>. Some special tokens are required to be included in every sequence. Given a sequence $\boldsymbol{x} = (t_1, t_2, \ldots, t_n)$

of variable length n, a start-of-sequence token <sos> and an end-of-sequence token <eos> are added such that $\boldsymbol{x}$ becomes (<sos>, $t_1, t_2, \ldots, t_n$, <eos>). The maximum sequence length T is a hyperparameter of every transformer, which has a large impact on the computational efficiency. The length of every sequence $\boldsymbol{x}$ is made uniform by appending the appropriate $T-(n+2)$ number of padding tokens <pad>. Any number of other task-specific special tokens can be included as required. Examples of which include a <sep> token for separating questions and answers, a <cls> token for classifying entire sentences, and language tokens such as <en> and <fr> for English and French respectively. After the raw samples $\boldsymbol{x}$ have been tokenized using the BPE algorithm, the necessary special tokens are included as an additional post-processing step.

2.5.2 GPT Architecture

There exists a whole family of different GPT models [76, 78, 94, 144, 171], which are the most successful example of decoder-only transformers. These models are used for tasks such as text completion and text generation. The current state-of-the-art large language models (LLMs) such as BERT [145] and GPT4 [171] contain huge amounts of parameters and are trained on vast amounts of data. Training a LLM on that large of a dataset is very computationally expensive, hence only large companies such as Google [168] and OpenAI [94] will have the necessary resources. The term GPT has become part of everyday language since October 2022 due to the revolutionary success of ChatGPT [94]. Part of the success of GPT models is owed to their versatility, which is characterised by the pre-training that is used. This means that a model is initially trained in an unsupervised fashion, on a very large, diverse dataset that lacks any data labels. This pre-trained model can then be fine-tuned on smaller, more task-specific datasets which do contain data labels[6]. The high degree of parallelism inherent in the transformer architecture, facilitates large scale, distributed training on multiple GPUs, which is instrumental to the success of GPT models. Due to OpenAI transitioning away from being open source to being for-profit, GPT2 is the last version with all of its architecture details publicly available [76], and it is therefore this version that

[6]It is possible to fine-tune a GPT model with just a domain-specific dataset and no labels. This essentially consists of restarting training on this smaller dataset, perhaps with a smaller learning weight. However, only supervised fine-tuning will be considered in this work.

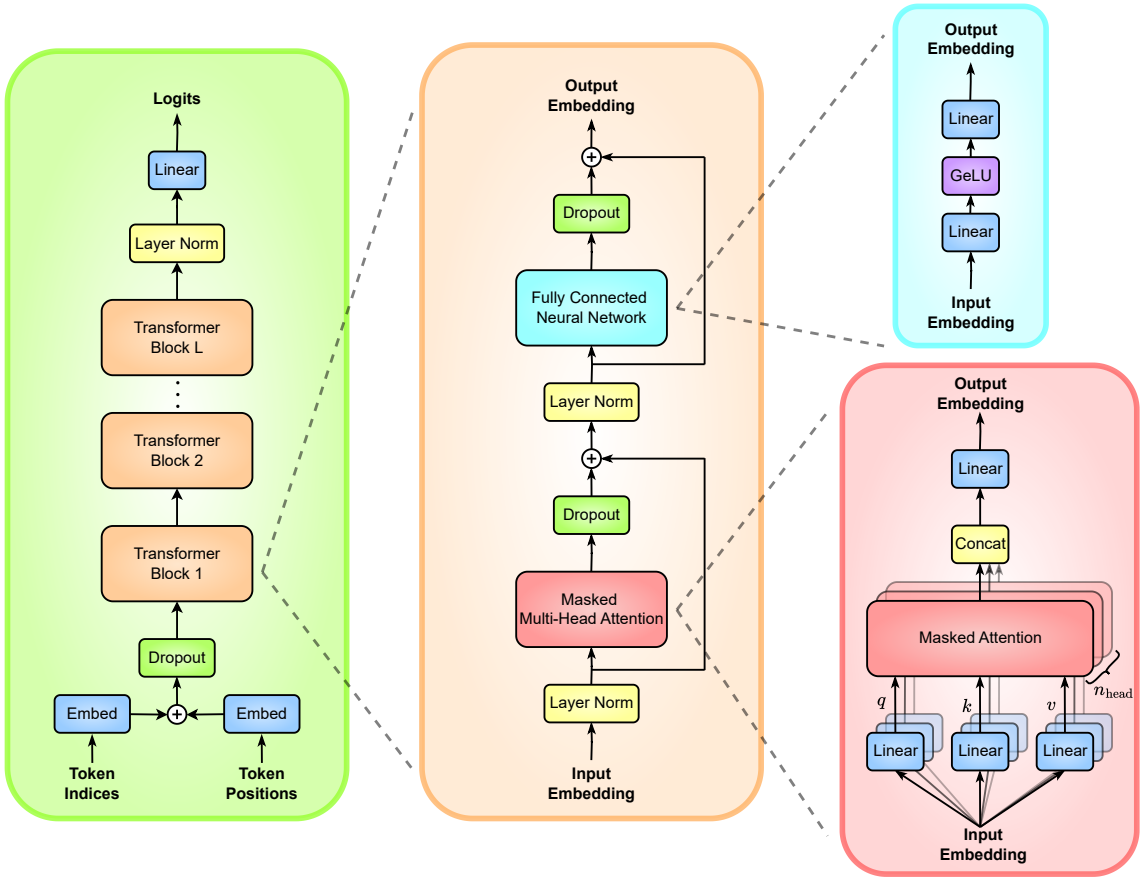

Figure 2.6: Diagram showing the full GPT architecture. The input of the GPT model (green) is the sequence of token indexes in the vocabulary $\mathcal{V}$. The model output is an unnormalised probability distribution that each token in $\mathcal{V}$ will be the next token in the sequence. There are a total of L transformer blocks (orange), with each block consisting a masked multi-head attention mechanism (red) and a fully connected neural network (blue).

is used throughout this thesis. The GPT model used in Chapter 6 is based on the open-source implementation nanoGPT [178].

A GPT model consists of a series of L transformer blocks, which are powered by the aforementioned attention mechanism[7]. Each block consists of two core components: a self-attention mechanism and a fully connected neural network. Every layer l in a transformer model performs its operations on context-aware embedding vectors

[7]Considering that in a decoder-only model, attention is calculated entirely within the same sequence, it can be more accurately termed as self-attention. In an encoder-decoder model, each transformer block of the decoder takes in attention scores from the encoder, which is termed as cross attention. It is beneficial for sequence to sequence tasks, but only self-attention scores are required for the sequence generation task of a decoder-only model.

$\boldsymbol{h}_l \in \mathbb{R}^{T \times d_{\text{model}}}$ where T is the number of tokens in the sequence and d_{model} is the dimension of the embedding space. The layout of this section on the architecture of the GPT model is now explained. The nature of input layer and the output layer of the model will be illustrated first. Then, the transformer block and the attention mechanism will be detailed. After which the sampling procedure used to select the next token in the sequence from the probability distribution of all tokens is described. Lastly, the specifics of fine-tuning a model on a labelled dataset are given.

Inputs and Outputs

All GPT models [76, 78] are based the original transformer architecture [29], whose objective is to learn to transform any input sequence into a desired output sequence. The initial input to the GPT model is a sequence of integers $\boldsymbol{x} \in \mathbb{Z}^T$, where each integer is the index of the corresponding token in the vocabulary $\mathcal{V}$. These integer indexes are then transformed into the token embedding vectors $\boldsymbol{e} \in \mathbb{R}^{T \times d_{\text{model}}}$ using embedding layers [179]. An embedding layer is a learnable transformation that converts categorical data into continuous, dense vectors of fixed size. These vectors aim to capture semantic content of the input data, grouping similar inputs closer together in vector space. Embedding layers are essentially a matrix $\boldsymbol{W}_e \in \mathbb{R}^{V \times d_{\text{model}}}$ that functions as a lookup table, where V is the size of the vocabulary $\mathcal{V}$. Each token index $i \in \{0, 1, \ldots, V-1\}$ corresponds to the ith row of $\boldsymbol{W}_e$. The elements in the matrix $\boldsymbol{W}_e$ are both the embedding vectors and the learnable weights, which are updated during backpropagation.

Some notion of the relative positions of the tokens within the sequence must be supplied. Transformers are not sequential in nature, processing all tokens simultaneously, meaning they do not inherently possess any way of distinguishing different orderings of the same sentence. However, there exists a simple method of differentiating tokens at different positions. An index $j \in \{0, 1, \ldots, T-1\}$ is assigned to every position and then transformed into a learnable position embedding vector $\boldsymbol{p} \in \mathbb{R}^{T \times d_{\text{model}}}$ using a another embedding layer with weight matrix $\boldsymbol{W}_p$. These position embeddings $\boldsymbol{p}$ are then directly summed onto the token embeddings $\boldsymbol{e}$ resulting in embeddings $\boldsymbol{h}_0$, which are inputted into the first transformer block.

The input embeddings $\boldsymbol{h}_0$ are passed through a total of L transformer blocks,

gaining a richer contextual representation after each block, resulting in final embeddings $\boldsymbol{h}_L$. The output layer is called a language modelling head, which is just a single fully connected linear layer transforming the final embedding $\boldsymbol{h}_L^{(t)} \in \mathbb{R}^{d_{\text{model}}}$ for each token $t \in \{0, 1, \ldots, T-1\}$ into unnormalised probability values for all V unique tokens in $\mathcal{V}$. These output values are called logits, and after applying a softmax activation function as in Equation (2.15) to normalise the distribution (makes sure it correctly sums to one), they represent the model's estimate for the probability of each token in $\mathcal{V}$ being the next token in the sequence. Actually predicting an appropriate next token is a process called sampling or decoding, and is detailed after the workings of the transformer block are explained. No bias terms are included in this linear layer, resulting in learnable weights $\boldsymbol{W}_{\text{lm}} \in \mathbb{R}^{V \times d_{\text{model}}}$, whose values are fixed to be the same as those as of $\boldsymbol{W}_e$ from the input token embedding layer. This is done to enforce the input embeddings $\boldsymbol{h}_0$ and output embeddings $\boldsymbol{h}_L$ to remain consistent throughout the model.

There are a number of hyperparameters that control the size and function of a transformer. These include the number of tokens in the vocabulary $n_{\mathcal{V}}$, the number of transformer blocks L, the number of attention heads n_{head}, the embedding dimension d_{model}, the attention vector dimension d_k, and the dimension of the fully connected neural network d_{fc}. The weights of the GPT model are updated using mini-batch stochastic gradient descent [110] with an Adam optimization scheme [106]. A cross-entropy loss function, as shown in Equation (2.3), is the standard training objective of a language model. This loss function measures the difference between the ground-truth next token and the predicted probabilities for the next token. A softmax function is used to convert logits to probabilities with the loss function. The ground-truth next token probability distribution is simply a one-hot encoded vector, with a one at the corresponding token index. This sparse ground-truth distribution means that the cross-entropy loss function of a GPT model can be simplified as

$$L_1(\boldsymbol{x}, \boldsymbol{l}) = -\frac{1}{T} \sum_{i=1}^{T} \log P(t_i | t_1, \ldots, t_{i-1}), \tag{2.28}$$

$$= -\frac{1}{T} \sum_{i=1}^{T} \log \left(\frac{\exp(l_{i,\text{true}})}{\sum_{j \in \mathcal{V}} \exp(l_{i,j})} \right), \tag{2.29}$$

where $\boldsymbol{x}$ is the sequence of tokens $\{t_1, \ldots, t_T\}$, $\boldsymbol{l} \in \mathbb{R}^{T \times \|\mathcal{V}\|}$ are the predicted logits with logit $l_{i,\text{true}}$ being the logit that corresponds to the ground truth token.

Transformer Block

The self-attention mechanism is responsible for encoding contextual information into the embedding vectors $\boldsymbol{h}$. For each token in the sequence, attention quantifies the relative importance of every other token. Unlike in recurrent neural networks (RNNs) [118], attention scores are not affected by the distance between the tokens. The attention mechanism works by first transforming the embedding $\boldsymbol{h}_l^{(t)} \in \mathbb{R}^{d_\text{model}}$ for each token $t \in \{0, 1, \ldots, T-1\}$ in each layer $l \in \{1, 2, \ldots, L\}$ into three separate vectors: the query $\boldsymbol{q}^{(t)}$, the key $\boldsymbol{k}^{(t)}$, and the value $\boldsymbol{v}^{(t)}$. Each of these three vectors are constructed using a fully connected linear layer, yielding a total of three weight matrices $\boldsymbol{W}_{\{q,k,v\}} \in \mathbb{R}^{d_{\{q,k,v\}} \times d_\text{model}}$ per layer l, where $d_{\{q,k,v\}}$ are the dimensions of the corresponding vector. These linear layers are classified as being position-wise layers, which means that they are independently applied to each token t resulting in final vectors $\{\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}\} \in \mathbb{R}^{T \times d_{\{q,k,v\}}}$. Given vectors $\boldsymbol{q}$, $\boldsymbol{k}$, and $\boldsymbol{v}$, attention is calculated as

$$\text{Attention}(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) = \text{softmax}\left(\frac{\boldsymbol{q}\,\boldsymbol{k}^T}{\sqrt{d_k}}\right)\boldsymbol{v}\,, \tag{2.30}$$

where $\boldsymbol{k}^T$ refers to transpose of $\boldsymbol{k}$, and $d_k = d_v$ is the dimension of key/query vectors and is used to scale inputs of the softmax function so as to prevent saturation and maintain stable gradients. This particular attention function is called the scaled dot-product attention [29] but other variations exist within the literature, including sparse sequences [180] and more efficient scaling with sequence length T [181].

Developing an intuition for the use of the query, key, and value vectors can be useful in understanding the self-attention mechanism. The query is the token for which it is desired to calculate attention. The key quantifies how relevant every other token is to the query token. And the value contains the actual meaningful content of each token. Attention can therefore be thought of as a weighted sum of the value vectors, where the weights are determined based on how similar the query and the key are.

The particular attention mechanism used in transformer is called masked multi-head attention. Masked simply means that the attention mechanism can only see

the preceding tokens in the sequence, the succeeding tokens are ignored. Multi-head means that the attention mechanism described thus far is repeated in parallel a total of n_{head} times. Each head i where $i \in \{0, 1, \ldots, n_{\text{head}}-1\}$ uses three different linear layers, as can be in Figure 2.6, resulting in a total of $3{*}n_{\text{head}}$ linear layers. Since all of these linear layers will have weights $\boldsymbol{W}^{(i)}_{\{q,k,v\}}$ that have different random initialisations, they will all result in different $\boldsymbol{q}$, $\boldsymbol{k}$, and $\boldsymbol{v}$ vectors. The attentions of every head are concatenated together, and passed through one final linear layer with a weight matrix $\boldsymbol{W}_o$ to yield an output embedding $\boldsymbol{h} \in \mathbb{R}^{T \times d_{\text{model}}}$, which has the appropriate shape. Multi-head attention is a very successful mechanism for attending to different aspects of the same sequence in multiple different ways. This provides an overall richer contextual representation, with the model learning many types of dependencies between tokens, such as long-range, short-range, semantics, syntax, style, etc. The entire procedure can be calculated in parallel, unlike recurrent neural networks, which massively increases scalability, facilitating the creation of the current generation of large language models.

The output embedding of the multi-head attention mechanism is then passed into a standard fully connected neural network. This neural network consists of two linear layers with a GeLU activation function as from Equation (2.14) in between. It is a position-wise neural network that uses the same weights $\{\boldsymbol{W}_1, \boldsymbol{W}_2\}$ for each token t in the sequence. The neural network's output embeddings $\boldsymbol{h}^{(t)}_{\text{fc}} \in \mathbb{R}^{d_{\text{model}}}$ have the same shape as the input embeddings as required. The dimension of the hidden layer is controlled using a hyperparameter d_{fc} that defaults to $4{*}d_{\text{models}}$ in the GPT2 architecture and in this work. Given the attention embeddings $\boldsymbol{h}^{(t)}_{\text{attn}} \in \mathbb{R}^{d_{\text{model}}}$ as inputs, the neural network will calculate output embeddings $\boldsymbol{h}^{(t)}_{\text{fc}}$ for each token t as

$$\boldsymbol{h}^{(t)}_{\text{fc}} = \text{GeLU}\left(\boldsymbol{h}^{(t)}_{\text{attn}}\,\boldsymbol{W}_1 + \boldsymbol{b}_1\right)\,\boldsymbol{W}_2 + \boldsymbol{b}_2\,, \tag{2.31}$$

where $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are bias parameter vectors for layers 1 and 2, respectively.

Besides the previously discussed attention mechanism and fully connected neural network, there are three further components in each transformer block. These can be listed in sequential order as (1) layer normalisation [117], (2) dropout [112], and (3) residual connections [182]. These three techniques are applied to both the attention layer and the neural network in that listed order. Layer normalisation scales

embeddings such that they have a mean of zero and a standard deviation of one, similar to Equation (2.7), using statistics calculated for each feature across the entire mini-batch. Dropout is a biologically inspired regularisation technique used to prevent overfitting, where neurons have a certain probability p of outputting zero. Residual connections directly sum input embeddings to output embeddings, aiding in the backpropagation of gradients of very deep networks. The places of application of these three techniques can be clearly seen in Figure 2.6. They are added to improve training stability, reduce overfitting, and help gradient flow.

Sampling

After the training of any generative language model such as GPT has concluded, the inference stage begins, where the objective is to generate novel, coherent sequences. Considering that the model simply estimates the probability that each token in the vocabulary $\mathcal{V}$ will be next in the sequence, a procedure to actually select this next token is required. This is referred to as decoding or as sampling. The outputs of the GPT model are logits, which are unnormalised probabilities for each token in $\mathcal{V}$. During the inference stage, these logits are passed through a softmax function, as defined in Equation (2.15), to yield normalised probabilities which sum up to one. A naive sampling procedure would be to greedily select the token in $\mathcal{V}$, which has the highest probability, but this can cause sequences to exhibit a high degree of repetition and incoherence. A more sophisticated method is to use beam search [183], which considers the conditional probabilities of the next b tokens in the sequence, with the best one being chosen in the end. This results in more coherent sequences than the greedy approach, but it is computationally expensive due to multiple model calls being required and it still has a tendency to repeat itself with common, safe phrases.

The sampling strategy that is used in both GPT2 [76] and in this work, is called top-k sampling [184]. It works by restricting the candidate pool of next tokens to only the top k highest-probability tokens. The probability distribution of these top k tokens is once again normalised such that it sums up to one. The next token is then chosen from this pool of k candidates, based on their probabilities. This sampling method strikes a balance between avoiding highly unlikely tokens and maintaining a high degree of coherence and diversity. An illustration of top-k sampling is shown

in Figure 6.3. However, there exists a more modern sampling procedure called top-p sampling [185], which restricts the candidate pool to include the minimum number of tokens whose cumulative probability exceeds some threshold p. This is the strategy used in GPT3 [144] and GPT4 [171]. It is not used in this work due to the lack of any open-source code that is optimised for use in a parallel training scenario.

Another commonly used method of controlling the inference process of a generative language model is to directly adjust the probability distribution generated by the softmax activation function in the output layer. This is achieved by modifying the original softmax function from Equation (2.15) to include a temperature term β, which controls how much weight is given to the less probable tokens. This modified softmax function is given as

$$\text{Softmax}(l_i, \beta) = \frac{e^{\beta l_i}}{\sum_{j \in \mathcal{V}} e^{\beta l_j}}, \tag{2.32}$$

where l_i is the logit for the ith token in $\mathcal{V}$. The temperature can be either defined as β or as T, where $\beta = \frac{1}{T}$. If more diversity and creativity is desired in the generated sequences then $T > 1$ will result in the less probable tokens being given a higher chance of selection. However, this can cause a decrease in the coherence of the generated sequences. Lowering the temperature $T < 1$ will result in the model choosing the safer, more likely tokens, increasing coherence at the cost of a lack of variation. Setting $T = 1$ will leave the softmax probability distribution unchanged from its original form in Equation (2.15). Choosing an appropriate temperature value during the inference stage is important. For example, when generating English language text using the ChatGPT API [94], a lower temperature value $T \sim 0.4$ is often advised in order to ensure that the text generated obeys proper grammar laws.

Fine-Tuning

After pre-training comes the next stage of fine-tuning, where the model undergoes further training on a smaller, domain-specific dataset. Fine-tuning is usually done in a supervised fashion on a labelled dataset. Utilising pre-trained weights, fine-tuning allows limited computational resources to be spent in obtaining a model that is trained for a specific task. Consider a dataset, where each sample is a sequence of token $\boldsymbol{x} = (t_1, t_2, \ldots, t_T)$ that has an associated data label $\boldsymbol{y}$. A sample $\boldsymbol{x}$ is passed

as an input into the GPT model as described, which outputs the Lth transformer block's final embeddings $\boldsymbol{h}_L$. An additional linear layer is defined, with weights $\boldsymbol{W}_y \in \mathbb{R}^{\|\boldsymbol{y}\| \times d_{\text{model}}}$, to predict $\boldsymbol{y}$ given the embedding vectors $\boldsymbol{h}_L^T$ for the last token in the sequence $\boldsymbol{x}$. The primary training objective of predicting the next token as in Equation (2.28), remains unchanged. A secondary training objective of predicting the target label $\boldsymbol{y}$ can be added as:

$$L_2(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \log P(\boldsymbol{y}|t_1, \ldots, t_T)\,. \tag{2.33}$$

These two different training objectives are then summed together with a coupling term λ as

$$L_3 = L_2 + \lambda\, L_1\,, \tag{2.34}$$

where L_3 is the updated loss function to be used to fine-tune the pre-trained transformer. However, if there are no data labels $\boldsymbol{y}$ available in this domain-specific dataset, then the original loss function from Equation (2.28) can instead be used for fine-tuning purposes.

2.6 Summary

This chapter includes all of the machine-learning (ML) methods and algorithms that are used throughout this work. A good knowledge of these algorithms is required for proper interpretation of the main results of Chapter 5 and Chapter 6. A brief summary of the contents of this chapter is now listed. First, a general introduction is given, where ML is defined, with the fundamental workflow being illustrated. Next, the most common ML paradigm of supervised learning was detailed, where each sample in the training dataset is labelled with additional information. Commonly used loss functions, proper training methods to prevent overfitting, and feature scaling procedures were all defined. The important gradient descent algorithm [105] was also briefly demonstrated. After this, neural networks were introduced, with activation functions [119, 125] and the crucial backpropagation algorithm [101] being described.

The majority of this chapter was spent introducing the two key generative models used in this work. The first of which are the generative adversarial networks (GANs) used in Chapter 5. The original GAN formulation, or the so-called the vanilla GAN, is first detailed. Then, the improved reformulation of Wasserstein GANs [27, 153], and the conditional GANs [32] used with labelled datasets are both explained. The second generative model that is described in this chapter are generative pre-trained transformers (GPTs) [76, 78], which are used in Chapter 6. The original transformer models [29] are introduced, followed by the principles of generative language modelling. The architecture of the GPT model is then comprehensively explained, with particular attention being given to the fundamental transformer block, which powers the model. Sampling methods used to generate new sequences with the model are then explained. The chapter closes with a description of how to fine-tune a GPT model on a domain-specific dataset.

Chapter 3

The Physics of Many Bodies

This chapter seeks to briefly detail the physics of interacting many-body systems, and also some related machine learning (ML) applications. First, the relevant quantum-mechanics (QM) equations are introduced. Density-functional theory (DFT) methods that are used to efficiently calculate QM properties are then detailed. Representations that accurately encode local chemical environments into ML feature vectors are illustrated. Suitable ML models for QM property prediction are then described.

3.1 Introduction

Given a system of atoms with known coordinates, quantum mechanics (QM) provides a full description of the atomic interactions by means of the Schrödinger equation, see Equation (3.1). However, the computational expense involved in performing full QM calculations limits the system size able to be considered. A series of simplifications can be made to increase this limit, at the cost of decreasing the level of accuracy [186]. Coupled cluster (CC) [187, 188] calculations provide the most accurate means of numerically solving the Schrödinger equation for most practical quantum-chemistry applications. However, CC has a prohibitively large scaling of at best $\mathcal{O}(n^6)$, which typically renders it only applicable to systems with tens of atoms. Therefore, density-functional theory (DFT) is used to computationally solve the Schrödinger equation, due to the more manageable scaling of $\mathcal{O}(n^3)$ [186]. There have been advances in a linearly scaling $\mathcal{O}(n)$ version of DFT, but it assumes that electronic interactions may be spatially truncated, which only true holds in very large systems. It also

has a large computational overhead due to the complexity of implementation. The scalings here mentioned are all relative to the number of basis functions n used to describe the electronic wavefunction, where n generally increases with the number of atoms. For these reasons, the traditional DFT implementation with its cubic scaling is chosen as the method of performing quantum-mechanical calculations in this work. All electronic structure theory presented in this chapter follows Reference [49].

3.2 Quantum Mechanics

The interactions between electrons and nuclei in a many-body quantum system can be described using the Schrödinger equation [189]. Consider a system with N_e electrons at positions $\boldsymbol{r}$ and N_n nuclei at positions $\boldsymbol{R}$. The time-independent, non-relativistic form of the Schrödinger equation [79] be written as

$$\hat{H}\psi(\boldsymbol{r},\boldsymbol{R}) = E\,\psi(\boldsymbol{r},\boldsymbol{R})\,, \tag{3.1}$$

where ψ is the wavefunction, $\hat{H}$ is the Hamiltonian operator [79], and E is the total energy of the system. The wavefunction ψ is a complex-valued eigenvector. Its physical meaning can be interpreted from its squared modulus $|\psi(\boldsymbol{r},\boldsymbol{R})|^2$ corresponding to the probability that the electrons and nuclei will be at positions $\boldsymbol{r}$ and $\boldsymbol{R}$, respectively.

The Hamiltonian operator includes kinetic energy and potential energy terms for both the electrons and nuclei as follows:

$$\begin{aligned}\hat{H} = &-\sum_{i}^{N_e}\frac{\hbar^2}{2m_e}\nabla_i^2 + \sum_{i<j}^{N_e}\frac{e^2}{|\boldsymbol{r}_i-\boldsymbol{r}_j|} - \sum_{i,I}^{N_e,N_n}\frac{Z_I e^2}{|\boldsymbol{r}_i-\boldsymbol{R}_I|} \\ &- \sum_{I}^{N_n}\frac{\hbar^2}{2M_I}\nabla_I^2 + \sum_{I<J}^{N_n}\frac{Z_I Z_J e^2}{|\boldsymbol{R}_I-\boldsymbol{R}_J|}\,, \qquad (3.2)\\ = &\hat{T}_e + \hat{V}_{ee} + \hat{V}_{en} + \hat{T}_n + \hat{V}_{nn}\,, \qquad (3.3)\end{aligned}$$

where m_e is the mass of the electron with charge e, and M_I is the mass of nuclei I with atomic number Z_I. Equation (3.3) more succinctly describes the individual Hamiltonian terms. The kinetic energies of the electrons and nuclei are $\hat{T}_e$ and $\hat{T}_n$ respectively. Additionally, there are three Coulombic interaction terms: the electron-electron repulsion $\hat{V}_{ee}$, electron-nucleus attraction $\hat{V}_{en}$, and the nucleus-nucleus repulsion $\hat{V}_{nn}$.

In the Born-Oppenheimer (BO) approximation [190], the wavefunction ψ is decoupled into electronic ψ_e and nuclear ψ_n contributions as $\psi(\boldsymbol{r}, \boldsymbol{R}) = \psi_e(\boldsymbol{r}, \boldsymbol{R})\,\psi_n(\boldsymbol{R})$ [191]. As the nucleus mass is at least three orders of magnitude larger than the electron mass, the nuclei can be assumed to remain stationary, while the electrons move dynamically. Therefore, when calculating the electronic wavefunction ψ_e, the nuclei positions are fixed at $\boldsymbol{R}$, allowing the kinetic energy term $\hat{T}_n$ to be neglected, and the nucleus-nucleus energy E_{nn} to become a classical additive term.

The electronic and nuclear Schrödinger equations under the BO approximation are respectively:

$$\left(\hat{T}_e + \hat{V}_{ee} + \hat{V}_{en}\right)\psi_e(\boldsymbol{r}, \boldsymbol{R}) + E_{nn} = E_e(\boldsymbol{R})\,\psi_e(\boldsymbol{r}, \boldsymbol{R})\,, \tag{3.4}$$

$$\left(\hat{T}_n + E_e(\boldsymbol{R})\right)\psi_n(\boldsymbol{R}) = E\,\psi_n(\boldsymbol{R})\,, \tag{3.5}$$

where E is the total energy, and $E_e(\boldsymbol{R})$ is called the potential energy surface (PES) [192]. This PES is a hypersurface used to describe how the energetics of a system change with respect to the nuclei positions. The PES is an important concept in the field of quantum chemistry, especially in the construction of suitably representative datasets used to train machine-learned interatomic potentials, see Section 3.5. In electronic structure calculations the kinetic energy of the nuclei is neglected, meaning that the PES energies E_e can be considered equivalent to the total energies E.

The nuclear Schrödinger equation is typically only solved in scenarios where a full quantum-mechanical treatment of nuclear motion is required. Calculating ground-state properties, predicting nuclear excitations, and modelling neutron stars are some examples of such scenarios [193]. It is common to approximate Equation (3.5) by using classical equation of motions, particularly in geometry optimisation [50] and molecular dynamics [194]. However, the electronic structure of the system must first be calculated, in a computationally feasible manner.

3.3 Density-Functional Theory

The electronic Schrödinger equation from Equation (3.4) depends on $3N_e$ coordinates, making it computationally intractable to solve directly for large systems [49]. Therefore, a means to solve this equation more efficiently is required. This is provided by density-functional theory (DFT), which reformulates the problem in terms of electron density $\rho(\boldsymbol{r})$, as described in the following section. Indeed, the Kohn-Sham formulation of DFT allows the previous many-body wavefunction ψ_e to be replaced by a single-body problem. Throughout this section, the Born-Oppenheimer approximation [190] is applied, with the nuclei remaining stationary at positions $\boldsymbol{R}$, which will be treated as implicit parameters hereafter.

The electron-nuclei interaction term $\hat{V}_{en}$ is commonly treated as some external potential, created by the nuclei, acting on each of the N_e electrons. This external potential operator $\hat{V}_{\text{ext}}$ is formed as a sum of the potentials on all electrons as

$$\hat{V}_{\text{ext}} = \sum_{i=1}^{N_e} V_{\text{ext}}(\boldsymbol{r}_i)\,, \tag{3.6}$$

with an expectation value for the external potential energy of:

$$E_{\text{ext}} = \int \rho(\boldsymbol{r}) V_{\text{ext}}(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r}\,, \tag{3.7}$$

where $\rho(\boldsymbol{r})$ is the single-electron density at position $\boldsymbol{r}$:

$$\rho(\boldsymbol{r}) = N_e \int |\Psi_e(\boldsymbol{r}, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_e})|^2 \mathrm{d}\boldsymbol{r}_2 \ldots \mathrm{d}\boldsymbol{r}_{N_e}\,. \tag{3.8}$$

The Hohenberg-Kohn (HK) Theorems [80] form the foundations of DFT. The first of the two theorems shows that the ground-state electron density $\rho(\boldsymbol{r})$ uniquely determines the external potential $V_{\text{ext}}(\boldsymbol{r})$, up to some constant. Since the external potential fully determines the electronic Hamiltonian $\hat{H}$, it implies that the ground-state wavefunction ψ_e, and all related properties thereof, are fully determined by the electron density $\rho(\boldsymbol{r})$. This theorem can be proven using the variational principle [80].

The second HK Theorem reformulates the ground-state total energy E as a functional of the electron density ρ as follows:

$$E[\rho] = T[\rho] + E_{ee}[\rho] + E_{\text{ext}}[\rho] + E_{nn}\,, \tag{3.9}$$

where T is the total kinetic energy of the N_e electrons, E_{ee} is the electron-electron energy, and E_{ext} is the external potential energy. It is noted that the nucleus-nucleus interaction energy E_{nn} is a classical additive term, and not a functional of density ρ, unlike the other terms. This second theorem can also be proved using a variational principle [80].

3.3.1 Kohn-Sham Scheme

The Kohn-Sham (KS) approach [81, 195] reformulates the many-body Schrödinger equation into a system of non-interacting electrons. A self-consistent field (SCF) procedure is iteratively performed until the non-interacting electrons generate the same electron density $\rho(\boldsymbol{r})$ as fully interacting electrons would.

Under the KS formulation, the total energy of a system of fully interacting electrons, in the presence of an external potential $V_{\text{ext}}(\boldsymbol{r})$ generated by some classical nuclei, can be expressed as

$$E[\rho] = T_s[\rho] + E_{\text{ext}}[\rho] + E_{\text{Hartree}}[\rho] + E_{\text{xc}}[\rho] + E_{nn}\,, \tag{3.10}$$

where T_s is the total kinetic energy of the N_e non-interacting electrons, and E_{Hartree} is the classical Coulombic electron-electron interaction energy, replacing the previous E_{ee}. Equation (3.10) resembles that of the HK total energy from Equation (3.9) with one additional term: the exchange-correlation energy E_{xc} that captures complex many-body quantum-mechanical effects into one term, which must be approximated. This new term consists of two components, the kinetic energy difference caused by the non-interacting approximation, and the non-classical electron-electron interaction energy that is not included in E_{Hartree}.

The three energy functionals that are different between these HK and the KS total energy equations can be given as

$$T_s[\rho] = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{N_e}\int |\nabla\varphi_i(\boldsymbol{r})|^2 \mathrm{d}\boldsymbol{r}\,, \tag{3.11}$$

$$E_{\mathrm{Hartree}}[\rho] = \frac{1}{2}\iint \frac{\rho(\boldsymbol{r})\rho(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|}\,\mathrm{d}\boldsymbol{r}\,\mathrm{d}\boldsymbol{r}'\,, \tag{3.12}$$

$$E_{\mathrm{xc}}[\rho] = (T[\rho] - T_s[\rho]) + (E_{ee}[\rho] - E_{\mathrm{Hartree}}[\rho])\,, \tag{3.13}$$

where $\varphi_i(\boldsymbol{r})$ are eigenstates of the following auxiliary KS equation for each electron i:

$$\left[-\frac{\hbar^2}{2m_e}\nabla^2 + V_s(\boldsymbol{r})\right]\varphi_i(\boldsymbol{r}) = \varepsilon_i\,\varphi_i(\boldsymbol{r})\,, \tag{3.14}$$

with orbital energies ε_i, and an effective potential $V_s(\boldsymbol{r})$ of:

$$V_s(\boldsymbol{r}) = -\sum_{I}^{N_n}\frac{Z_I}{|\boldsymbol{r}-\boldsymbol{R}_I|} + \int\frac{\rho(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|}\mathrm{d}\boldsymbol{r}' + \frac{\delta E_{\mathrm{xc}}[\rho]}{\delta\rho(\boldsymbol{r})}\,, \tag{3.15}$$

$$= V_{\mathrm{ext}}(\boldsymbol{r}) + V_{\mathrm{Hartree}}(\boldsymbol{r}) + V_{\mathrm{xc}}(\boldsymbol{r})\,. \tag{3.16}$$

Equation (3.15) makes the assumption that the external potential $V_{\mathrm{ext}}(\boldsymbol{r})$ only consists of electron-nuclei interactions, which is not always valid. This effective potential $V_s(\boldsymbol{r})$ is an approximate local potential in which the non-interacting electrons move. It is used to solve Equation (3.14), yielding the KS orbitals $\{\varphi_i(\boldsymbol{r})\}_{i=1}^{N_e}$. The electron density $\rho(\boldsymbol{r})$ can be calculated from these orbitals as

$$\rho(\boldsymbol{r}) = \sum_{i=1}^{N_e}|\varphi_i(\boldsymbol{r})|^2\,, \tag{3.17}$$

where the appropriate normalisation is enforced:

$$N_e = \int\rho(\boldsymbol{r})\mathrm{d}\boldsymbol{r}\,. \tag{3.18}$$

The KS equations are, typically, iteratively solved using a self-consistent field (SCF) cycle. A basis set [196] must be specified for the electron orbitals $\{\varphi_i(\boldsymbol{r})\}_{i=1}^{N_e}$, along with an initial guess for the electron density $\rho(\boldsymbol{r})$, as according to Equation (3.17). The corresponding effective potential $V_s(\boldsymbol{r})$ is then used to solve Equation (3.14), yielding updated orbitals and densities. This cycle is repeated until the difference in densities between subsequent iterations is less than some convergence thresh-

old. After convergence, the ground-state densities $\rho(\boldsymbol{r})$, KS orbitals $\varphi_i(\boldsymbol{r})$, total energies E, and orbital energies ε_i can all be returned. Related properties such as dipole moments and HOMO-LUMO gap energies can also be calculated [49]. The forces on the nuclei I are related to the derivatives of the total energy E with respect to the nuclei positions $\boldsymbol{R}_I$ as

$$\boldsymbol{F}_I = -\frac{\partial E}{\partial \boldsymbol{R}_I}. \tag{3.19}$$

The exact functional form for the exchange correlation term E_{xc} is unknown, and therefore, some level of approximation must be introduced. There exists a trade-off between accuracy and computational cost in the choice of functionals [197]. In this work, two different functionals are used: PBE [198] and B3LYP [199–202]. The first is the simpler of the two, and is used in the MD-17 dataset [87] in Chapters 4 and 5. The PBE functional falls under the class of generalized gradient approximations (GGA) [198], where the exchange-correlation energy depends on both the density $\rho(\boldsymbol{r})$ and its gradient $|\nabla\rho(\boldsymbol{r})|$. The B3LYP functional takes a more computationally expensive, semi-empirical, hybrid approach, and is used in the QM9 dataset [14, 90] in Chapter 6. It consists of a weighted sum of exchange and correlation energies from both GGA and a local density approximation (LDA) [81, 203], where LDA has no gradient dependency on the density $\rho(\boldsymbol{r})$. An exact exchange energy term as calculated by the Hartree-Fock methods [49, 204] is also included in the sum. The parameters of the weighted sum are determined by fitting to experimental data [201]. All DFT calculations in this thesis are performed using the PySCF package [42–44].

Electronic structure calculations can be used to perform geometry optimisation or structure relaxation. This allows some initial structure to be iteratively updated until the forces on each nuclei are at a minimum. It is hoped that this will correspond to the global minimum on the potential energy surface, but the structure may instead become trapped in some local minima. Geometry optimisation works by first performing a self-consistent field cycle on a structure, then calculating the nuclear forces, and finally updating the nuclei positions to minimise these forces. These steps are repeated until the structure is converged, which occurs when all forces are below some threshold. The resulting equilibrium structure is often a more appropriate candidate for reporting accurate QM property information.

3.3.2 HOMO-LUMO Gap

This section, which follows Reference [205], introduces a property which is used throughout Chapter 6. In the Kohn-Sham (KS) formulation of DFT, N interacting electrons are modelled as a fictitious system of non-interacting electrons. One key property often calculated using DFT is called the band gap (in periodic systems) or the HOMO-LUMO gap (in finite systems). This is the energy difference between the highest occupied molecular orbital (HOMO) and the lowest unoccupied molecular orbital (LUMO). In KS-DFT, the HOMO-LUMO gap $E_{\text{gap}}^{\text{KS}}$ is calculated as:

$$E_{\text{gap}}^{\text{KS}} = \varepsilon_{\text{LUMO}} - \varepsilon_{\text{HOMO}}\,, \tag{3.20}$$

where $\varepsilon_{\text{LUMO}}$ and $\varepsilon_{\text{HOMO}}$ are respectively the LUMO and HOMO energies. This $E_{\text{gap}}^{\text{KS}}$ refers to the energy difference between Kohn-Sham orbitals of the non-interacting system. In the true system of N interacting electrons, there exist two related observables called the ionisation potential (IP), which is the energy required to remove an electron, and the electron affinity (EA), which is the energy gained by adding an electron. Physically, the gap relevant for reactivity and optoelectronic behaviour [51] is the true gap or the fundamental gap E_{gap}, defined as:

$$E_{\text{gap}} = \text{IP} - \text{EA}\,. \tag{3.21}$$

Even in exact DFT, the HOMO–LUMO gap $E_{\text{gap}}^{\text{KS}}$ does not equal the fundamental gap E_{gap}. When an electron is added to the system, the exchange–correlation potential V_{xc} can jump discontinuously by a finite amount. This is known as the derivative discontinuity Δ_{xc}, separating the HOMO-LUMO gap from the fundamental gap:

$$E_{\text{gap}} = E_{\text{gap}}^{\text{KS}} + \Delta_{\text{xc}}\,. \tag{3.22}$$

Local and semi-local functionals [198,203] systematically underestimate the HOMO-LUMO gap due to their lack of the derivative discontinuity [205]. This effect is more prominent in molecular systems than in solids where $N \gg 0$. In conclusion, it should be noted the HOMO-LUMO gap is not a physically relevant observable, it is a proxy for the fundamental gap.

3.4 Chemical Representation

In any machine-learning (ML) application, the construction of suitable feature vectors is crucial for enabling accurate predictions to take place. These features are used to describe each of the individual samples in the dataset. In chemical systems, the representation determines a ML model's accuracy, efficiency, and generalizability. It is common to use simple, linear ML models [206] with a sophisticated chemical representation [84, 207, 208]. This highlights that the foundation of the efficacy of ML on chemical systems lies in the representation and not in the model. The availability of structural data, in the form of atomic Cartesian coordinates, broadly determines which representations can be chosen for any given task.

In scenarios where no three-dimensional (3D) structural data is available, then the chemical representation must be primarily composition based. There exist many ML applications that only consider chemical composition [209–211], mainly in predicting the bulk properties of materials. In molecular systems, two-dimensional (2D) representations [72,73,212,213] are widely used in the field of cheminformatics for property prediction and drug discovery [30, 31, 33, 34, 214–219]. These 2D representations consider molecules as a text-based graph with atoms being nodes and bonds being edges. They contain information such as chemical species, bond types, and atom connectivities but lack any notion of physical distance or atomic positions. This thesis is primarily concerned with ML applications on 3D structural data. However, 2D representations such as SMILES [72] and a newly proposed 'MolBlox' representation are both used in Chapter 6, see Figure 6.1.

Incorporating structural information about the atomic coordinates has been shown to dramatically improve the ability of ML models to successfully predict quantum-mechanical properties such as total energies, band gaps, etc [220–222]. The most naive choice for a 3D representation would be to use Cartesian coordinates $\boldsymbol{r} \in \mathbb{R}^{N_{at}\times 3}$, where each of the N_{at} atoms are described by its x, y and z components. However, there are problems associated with using Cartesian coordinates as a descriptor, primarily due to their lack of translational and rotational invariances. Cartesian coordinates are an over-complete representation, see Figure 3.4, where every structure corresponds to multiple different descriptors due to all the possible translations and rotations. This hinders learning as the model will struggle to recognise that structures

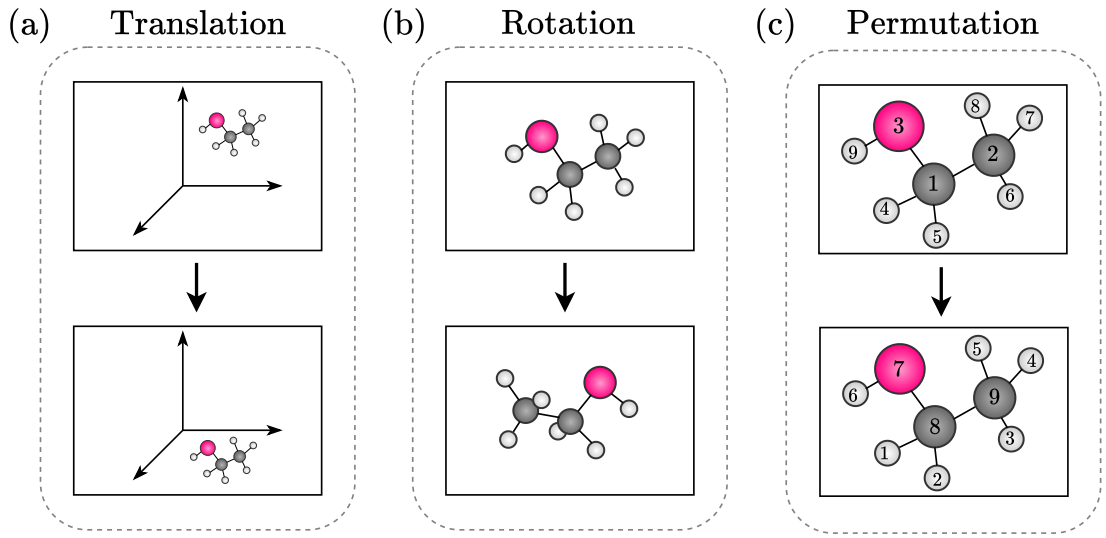

Figure 3.1: Diagram showing three different invariances that a suitable chemical representation should have. The representation's feature vectors must not change under these three transformations. Panel (a) shows a global translation of the position of a molecule. Panel (b) shows a molecule being rotated. Panel (c) shows a permutation of the atom indices.

may in fact be similar, despite having very different descriptors. They also lack any inherent physical meaning, e.g., no notion of bonding, electronic interactions, or other chemical properties are encoded. For these reasons, it is desirous to convert the raw Cartesian coordinates $\boldsymbol{r}$ into some descriptors $\boldsymbol{x}$ that are invariant with respect to translations, rotations, and permutations of atom indices, as shown in Figure 3.1.

3.4.1 Many-Body Representations

Numerous different types of 3D representations are commonly used for machine-learning property prediction [75, 85, 207, 208, 220–223] and inverse design tasks [224–229]. However, the most effective way to predict quantum-mechanical properties to a high degree of accuracy is to use many-body representations [75, 207, 208, 230–232]. This style of representation encodes the local chemical environment around each atom, using a series of many-body interactions as shown in Figure 3.2. They consider interatomic distances (2-body) and interatomic angles (3-body), as well as other higher-order interactions between atom triplets, quadruplets, quintuplets, etc. A key com-

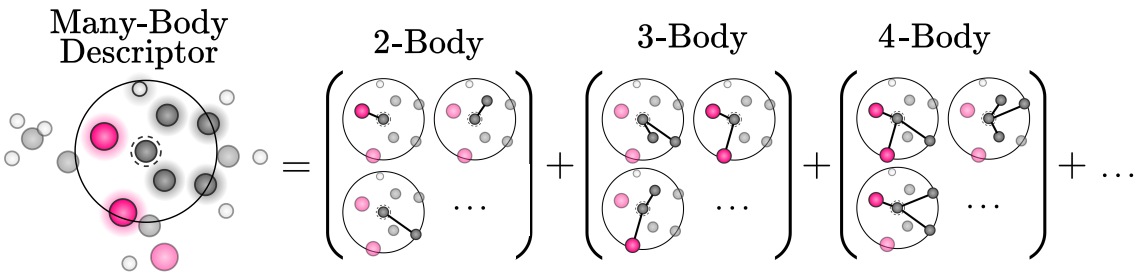

Figure 3.2: Diagram showing the constituents of a many-body descriptor for a single atom within a molecule of aspirin. The atom's local environment is encoded by considering interactions between neighbours within a sphere of some cutoff radius. Successively higher-order body terms are used, where n-body refers to interactions between n different atoms. The 2-body term only includes interatomic distances, while the 3-body term considers angles formed by triplets of atoms. The 4-body term incorporates dihedral angles, and other such interactions between four atoms. Further higher body terms can be included so as to increase the accuracy of the descriptor.

ponent of many-body representations is locality, where only nearby atoms are considered in the construction of the descriptors. As all interactions are internal, with no reference to some external coordinate system, the descriptors do not change with translations and rotations as required, making them an appropriately complete representation. Some especially sophisticated representations [207, 208] consider 5-body terms and beyond, making them essentially complete representations [233], with a one-to-one mapping between structure and descriptor. Of the various many-body representations available, symmetry functions [83, 232] and bispectrum components [75, 84] are detailed here.

Compositional information is also encoded within these 3D structure-based representations. Consider a dataset with m configurations $(\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}, \ldots, \boldsymbol{r}^{(m)})$ expressed in Cartesian coordinates. The dataset also contains the associated DFT energy values $(E^{(1)}, E^{(2)}, \ldots, E^{(m)})$, which are the target properties to be predicted. Just using the x, y, and z components that are contained in $\boldsymbol{r}$ for each of the N_{at} atoms is not sufficient, in that some notion of the chemical species Z_i of atom i must also be encoded. This means that additional features $(\boldsymbol{Z}^{(1)}, \boldsymbol{Z}^{(2)}, \ldots, \boldsymbol{Z}^{(m)})$, where $\boldsymbol{Z} \in \mathbb{Z}^{N_{at}}$, are required by any ML model to be used on atomic systems. These chemical species $\boldsymbol{Z}$

are typically in the form of integer atomic numbers for each atom, hence the notation Z. This assumes that the chemical composition varies for each sample. However, in many cases, the dataset may consist of conformations or isomers, all having the same composition $\boldsymbol{Z}$, but different structures $\boldsymbol{r}$. Throughout this thesis, when discussing some set of configurations $\{\boldsymbol{r}^{(i)}\}_{i=1}^{m}$, these composition features $\{\boldsymbol{Z}^{(i)}\}_{i=1}^{m}$ can be assumed to be present, despite not being explicitly referenced.

Many-body representations enforce locality by only considering interactions between neighbours contained within a sphere, of cutoff radius r_c, around the central atom i. These many-body interactions are weighted using a cutoff function $f_c(r_{ij})$ that smoothly approaches zero as $r_{ij} \rightarrow r_c$, where r_{ij} is the distance between atom i and atom j. Various cutoff functions exist, but the following is most commonly used:

$$f_c(r_{ij}) = \begin{cases} 0.5 \cdot \left[\cos(\frac{\pi\, r_{ij}}{r_c}) + 1 \right] & \text{if } r_{ij} < r_c\,, \\ 0 & \text{if } r_{ij} \geq r_c\,, \end{cases} \tag{3.23}$$

where the value of r_c is often heuristically defined so as to include a small number of nearest neighbours. It can be considered as a hyperparameter, with an optimal r_c being that which gives the lowest error on the validation dataset. However, this is rarely performed, as an entirely new dataset of feature vectors must be created for each different r_c, rendering most hyperparameter optimisation procedures [234, 235] too computationally expensive.

3.4.2 Symmetry Functions

One of the first prominent methods of encoding local chemical environments of an atomic system into feature vectors was introduced by Behler et al. [83]. The proposed chemical representation utilised atom-centered symmetry functions (ACSFs) [83, 232, 236], which are here detailed. A number of improvements to the original formulation have been proposed over the years [236, 237]. The local environment of each atom i is calculated as a sum of interactions between neighbouring atoms. A simple set of Gaussian distributions are used as the basis set. As required, the representation is invariant to translations, rotations, and permutations. Locality is enforced by using a cutoff function like that of Equation (3.23).

Symmetry functions consist of radial and angular functions. The radial functions

probe outwards in a series of spherical shells, measuring the frequency that neighbouring atoms occur within these shells. These are commonly called the 2-body terms in the ACSFs as they only consider interatomic distances between two atoms. The angular functions work similarly, by successively probing the density of neighbouring atoms at different solid angles. The angular functions encode 3-body terms into the ACSFs, enhancing the accuracy of the representation. Both functions probe their respective environments using successive Gaussian distributions. The widths and locations of these Gaussian distributions are varied to make each of the N_f different features. The functions used in generating these radial and angular components for each atom i are:

$$G_{i,m}^{\text{rad}} = \sum_{j\neq i}^{\text{all atoms}} e^{-\eta\,(r_{ij}-r_s)^2} f_c(r_{ij})\,, \tag{3.24}$$

$$G_{i,m}^{\text{ang}} = 2^{1-\zeta} \sum_{j,k\neq i}^{\text{all atoms}} (1+\cos(\theta_{ijk}-\theta_s))^{\zeta} \exp\left[-\eta\big(\tfrac{(r_{ij}+r_{ik})}{2}-r_s\big)^2\right] f_c(r_{ij}) f_c(r_{ik})\,, \tag{3.25}$$

where the index m is over the set of four hyperparameters: η, r_s, ζ, and θ_s. The Gaussian distribution widths are controlled by η and the peak locations by r_s. The parameter ζ specifies the width of the angular region probed and θ_s controls the peak location. Typically, the distribution widths η and ζ are fixed while multiple values for r_s and θ_s are used. This is a convention used to limit the number of hyperparameters that must be specified. The radial and angular components are then concatenated into a feature vector.

3.4.3 Bispectrum Components

Another chemical representation that is used to describe local atomic environments is that of called bispectrum components [75]. These are many-body representations that are used throughout this work. The accuracy of symmetry functions and bispectrum components is comparable [85] but the latter are chosen as they require fewer hyperparameters and generally use less feature vectors. Another reason for their adoption in this work, is that the LAMMPS [238, 239] implementation provides the derivatives of the descriptors with respect to the positions of the atoms, which are required in Chapter 4.

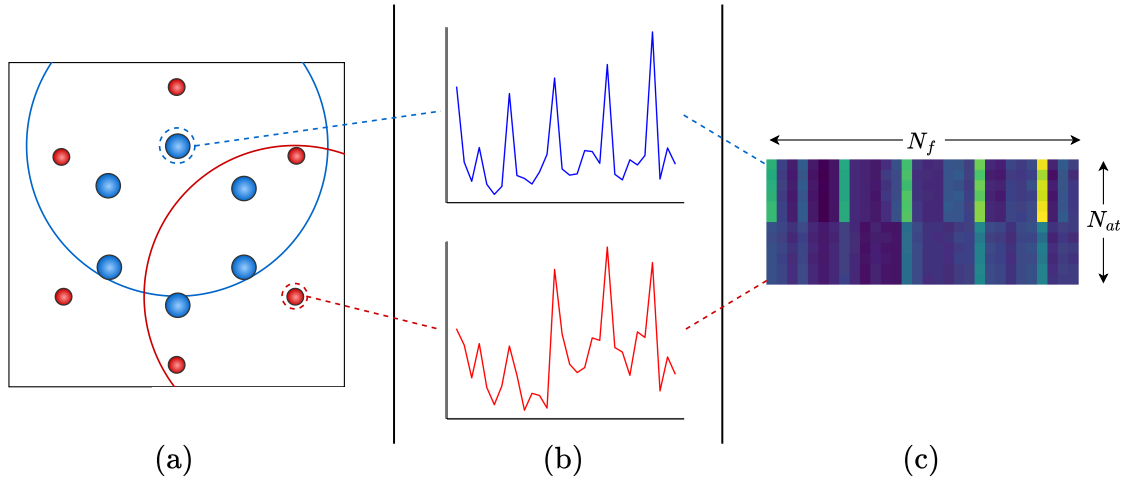

Figure 3.3: Diagram showing the construction of many-body descriptors as feature vectors encoding information about the local environment around each atom. Panel (a) shows a configuration of benzene with a cutoff radius r_c centered on atoms of each species present. Panel (b) shows bispectrum components feature vectors for a carbon atom and a hydrogen atom. Panel (c) shows the calculated bispectrum components for all atoms in the benzene molecule visualised as a 2D colour plot.

Bispectrum components are constructed for each atom i by first modelling the neighbouring atomic density distribution at arbitrary position $\boldsymbol{r}$ using a sum of delta-functions:

$$\rho_i(\boldsymbol{r}) = \delta(\boldsymbol{r}) + \sum_j w_{s_j} f_c(r_{ij})\, \delta(\boldsymbol{r} - \boldsymbol{r}_j)\,, \tag{3.26}$$

where $w^{(s_j)}$ is a weight assigned to the species s of each atom j at position $\boldsymbol{r}_j$, the interatomic distance r_{ij} is equal to $|\boldsymbol{r}_i - \boldsymbol{r}_j|$, and $f_c(r_{ij})$ is the same cutoff function as in Equation (3.23).

In order to construct meaningful feature vectors that accurately describe the local environment of each atom i, it is natural to expand the density ρ_i using a series of basis functions. To avoid having to use both spherical and radial basis functions with components (r, θ, ϕ), the density ρ_i from Equation (3.26) undergoes a Riemann-like projection onto a 3-sphere, which is a unit sphere in 4D. By representing this 3D density distribution on the surface of a 4D sphere, it allows the density ρ_i to be described entirely using angular basis functions with components (ϕ, θ, θ_0) [230]. The

appropriate transformation is given as

$$\boldsymbol{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{array}{l} \phi = \arctan(y/x)\,, \\ \theta = \arctan\left(\frac{\sqrt{x^2+y^2}}{z}\right)\,, \\ \theta_0 = \pi\,\frac{r}{r_0}\,, \end{array} \tag{3.27}$$

where $r_0 > r_c$ is a parameter controlling which pole of the 3-sphere that the projection is focused on. A complete natural basis can be formed using 4D spherical harmonics, also called Wigner matrices [240]. These matrices are denoted as $U^j_{m,m'}$ where j consists of half integer values and $m, m' = -j, -j+1, \ldots, j-1, j$. The full basis expansion of the density ρ_i for each atom i is given as

$$\rho_i(\phi, \theta, \theta_0) = \sum_{j=0}^{\infty} \sum_{m=-j}^{j} \sum_{m'=-j}^{j} u^j_{i,m,m'}\, U^j_{m,m'}(\phi, \theta, \theta_0)\,, \tag{3.28}$$

where $u^j_{i,m,m'}$ are expansion coefficients that used to construct the bispectrum components vectors for each atom i. These coefficients can be calculated as the inner product of matrices $U^j_{m,m'}$ and density ρ_i:

$$u^j_{i,m,m'} = \langle U^j_{m,m'} | \rho_i \rangle\,, \tag{3.29}$$

$$= U^j_{m,m'}(0,0,0) + \sum_j w_{s_j}\, f_{\mathrm{c}}(r_{ij})\, U^j_{m,m'}(\phi, \theta, \theta_0)\,. \tag{3.30}$$

These coefficients $u^j_{i,m,m'}$ contain all of the necessary information about the local environment of atom i. However, they are complex-valued and are not rotationally invariant, and are therefore unsuitable for use as a set of descriptors. This motivates the construction of the bispectrum components B_{j,j_1,j_2} which are non-complex and are invariant under rotations [230]. They are defined using a triple inner product of the expansion coefficients $u^j_{i,m,m'}$ as follows:

$$B_{j,j_1,j_2} = \sum_{m_1,m_1'=-j_1}^{j_1} \sum_{m_2,m_2'=-j_2}^{j_2} \sum_{m,m'=-j}^{j} \left(u^j_{i,m,m'}\right)^* C^{j,j_1,j_2}_{m,m_1,m_2} C^{j,j_1,j_2}_{m',m_1',m_2'} u^{j_1}_{i,m_1,m_1'} u^{j_2}_{i,m_2,m_2'}\,, \tag{3.31}$$

where j_1 and j_2 are bounded as $\|j_1 - j_2\| \leq j \leq \|j_1 + j_2\|$, and $C^{j,j_1,j_2}_{m,m_1,m_2}$ are coupling coefficients, analogous to the 2-sphere Clebsch-Gordon coefficients [240]. Equation (3.28) can be truncated with $j, j_1, j_2 \leq j_{\max}$, where the value of $j_{\max}$ [241] determines

the number of features N_f for each atom which is given as

$$N_f = \frac{(j_{\max}+1)(j_{\max}+\frac{3}{2})(j_{\max}+2)}{3}\,. \tag{3.32}$$

In conclusion, bispectrum components consider any system containing a total of N_{at} atoms, and encode the local atomic environment around each atom as a vector. These vectors form a matrix $\boldsymbol{B} \in \mathbb{R}^{N_{at}\times N_f}$ that is suitable for use in machine-learning applications. The number of features N_f corresponds to the coarseness of the spatial resolution of the representation, with this work using a default value of $2j_{\max} = 8$, which corresponds to $N_f = 55$. A diagram illustrating the construction of bispectrum components is shown in Figure 3.3. They are invariant to translations, rotations, and permutations as required, see Figure 3.1.

3.4.4 Completeness

Many-body descriptors such as the bispectrum components [75] are used to accurately represent interactions between atoms within a system. However, the intrinsic limitations of bispectrum components, and other similar representations such as symmetry functions, must be addressed. Both these representations satisfy the general requirements for use as a descriptor, such as being as descriptive, differentiable, and invariant to rotations, translations, and permutations. It was previously thought that they formed a unique one-to-one bijective mapping between atomic structures $\boldsymbol{r}$ and many-body descriptors $\boldsymbol{x}$. This was disproved by Pozdnyakov et al. [233] who showed that the same set of descriptors can be formed using different structures. Therefore, bispectrum components can be considered an incomplete representation, see Figure 3.4. Despite this lack of completeness, many-body descriptors are still a much more efficient representation than Cartesian coordinates where all translations, rotations, and permutations of $\boldsymbol{r}$ yield a different $\boldsymbol{x}$.

A many-body representation constitutes a complete basis for the atomic neighbour density distribution, as in Equation (3.26), if and only if up to n-body terms are included, where n is number of atoms in the system. Symmetry functions could potentially be adapted to include additional higher-order terms like in the 3-body case from Equation (3.25). However, each higher-order body term would need to be hand

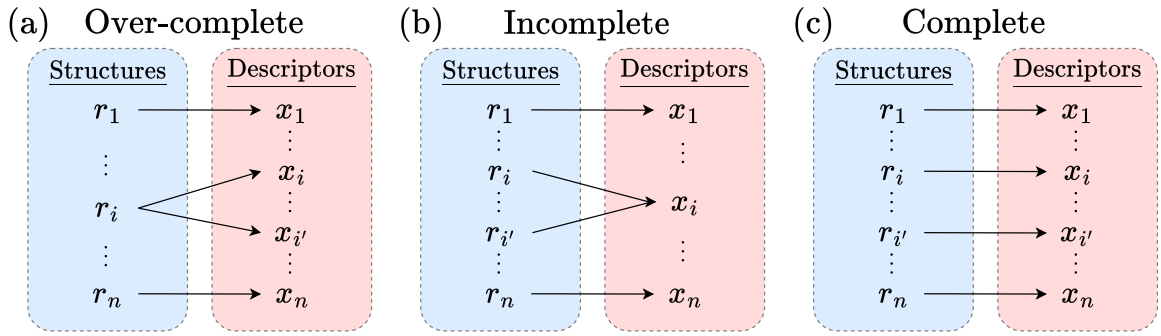

Figure 3.4: Diagram showing the three different possible mappings from atomic structures $\boldsymbol{r}$ to descriptors $\boldsymbol{x}$. Panel (a) shows an over-complete scenario where a structure can be described by multiple descriptors. Panel (b) shows the case where multiple structures can map to the same descriptor. And lastly, panel (c) shows the ideal scenario of a bijective mapping, with each structure corresponding to a unique descriptor.

crafted and would introduce a number of additional hyperparameters. Bispectrum components have no straightforward way of being reformulated to account for interactions beyond that of the 4-body. This motivates the development of more complete representations that do include higher-order terms.

The relative importance of successively higher-order body terms decays, with 2-body and 3-body terms generally being the most predictive of the quantum-mechanical properties, while higher-order terms may contribute minimally. This means that truncating the body order can yield an acceptable accuracy, with negligible difference between the full n-body case. The computational expense of calculating all these n-body terms increases exponentially with n, meaning that the inclusion of terms beyond that of the 5-body is rarely feasible.

In this work, only up to 4-body terms are considered, see Figure 3.2, but there are other more accurate representations which allow for the inclusion of an arbitrary number of body order terms. The first proposed method of including all n-body terms was that of moment tensor potentials [231]. The atomic cluster expansion [207] was later introduced, exhibiting a high degree of accuracy and versatility, making it widely used [242–245]. The Jacobi-Legendre polynomials [208] have been recently proposed as another method of including higher-order body terms.

3.5 Interatomic Potentials

There exist scenarios where the computational expense involved in performing full quantum-mechanical (QM) calculations render even the most inexpensive density functional theory (DFT) approximations infeasible. Such cases motivate the use of simpler interatomic potentials (IAP) that model the potential energy of a system as a function of the atomic coordinates. IAPs provide a much more computationally efficient alternative to DFT or coupled cluster [187,188]. This increased efficiency allows for either a larger number of atoms or for a larger number of separate calculations to be performed within the same time period. The forces acting on each atom are simply the derivatives of this potential with respect to the atomic coordinates, which is why IAPs are often called force fields. IAPs can be used to predict any quantity related to the potential energy of a system such as phonon modes, Raman spectra, formation enthalpies, heat capacities, phonon-predicted thermal conductivities, bulk and shear moduli, elastic constants, and many others [49]. There exists a hierarchy of different types of IAPs, with a trade-off between accuracy and computational expense.

The potential energy E of a system containing N_{at} atoms with coordinates $\boldsymbol{r} \in \mathbb{R}^{N_{at}\times 3}$ can be decomposed into a series of body order terms:

$$E = \sum_i \phi_1(\boldsymbol{r}_i) + \sum_{i<j} \phi_2(\boldsymbol{r}_i, \boldsymbol{r}_j) + \sum_{i<j<k} \phi_3(\boldsymbol{r}_i, \boldsymbol{r}_j, \boldsymbol{r}_k) + \dots \,, \tag{3.33}$$

where ϕ_n is a function that consider interactions between n different atoms. The first term ϕ_1 represents some external field, which is neglected in this work as no systems with external fields are considered. Many simple IAPs such the Lennard-Jones (LJ) [246] and the Morse [247] potentials can effectively predict the energy E of appropriate systems by exclusively considering pairwise interactions as captured by the ϕ_2 term. These are classified as semi-empirical potentials, consisting of a simple analytical functional form that is generally parametrised based on experimental data.

There exist more sophisticated IAPs that do consider many-body interactions as described by the ϕ_3 term in Equation (3.33). These include the Tersoff [248] and Stillinger-Weber [249] potentials, which both combine 2-body and 3-body terms, parametrised to describe covalently bonded materials. The embedded atom method [250] is used for systems with metallic bonding, as it specifically incorporates a many-

body electron density term. For modelling chemical reactions and complex dynamical systems, ReaxFF [251] is typically used as an intermediary between classical force fields and full quantum-mechanical calculations. Specific force-fields such as CHARM [252] and AMBER [253] exist for studying large biomolecules such as proteins and lipids. They consist of a summation of pairwise, angular, and torsional terms but are not classified as true many-body potentials. They also contain Coulombic terms for electrostatics and use the LJ potential for estimating Van der Waals interactions. These force-fields are very computationally efficient due to simplifications such as assuming fixed bond lengths.

A promising alternative to these traditional IAPs is to use machine-learned interatomic potentials (MLIAPs). An IAP is simply a function that maps atomic coordinates $\boldsymbol{r}$ to energies E, and a MLIAP seeks to learn this mapping. This is possible as ML algorithms act as universal approximators [172] that can learn any function, if supplied with sufficient training data. As previously mentioned in Section 3.4, the Cartesian atomic coordinates $\boldsymbol{r}$ must first be transformed into some set of descriptors $\boldsymbol{x}$ to facilitate efficient learning. Three separate components are needed for constructing a MLIAP. The first is a dataset containing reasonable, converged structures with energy values well distributed across the potential energy surface (PES). The second component are descriptors that accurately describe each atom's local chemical environment. The final component is a suitable regressor: a ML model that predicts the energy values E given the descriptors $\boldsymbol{x}$. Two such structure-to-property models are now detailed in the following sections.

3.5.1 Behler-Parrinello Neural Network

The same paper that introduced symmetry functions also introduced a new property predictor model called a Behler-Parrinello neural network (BPNN) [83]. This BPNN uses local many-body descriptors as an input. Symmetry functions are traditionally used as the many-body descriptors in BPNNs, but bispectrum components are equally valid and are used in this work. The key insight behind a BPNN is that the total energy E_T, of a molecule containing N_{at} atoms, can be decomposed into a sum of atomic contributions E_i as follows:

$$E_T = \sum_{i=1}^{N_{at}} E_i \,. \tag{3.34}$$

Using this decomposition, a BPNN predicts E_T given an input of many-body descriptors in the form $\boldsymbol{x} \in \mathbb{R}^{N_{at} \times N_f}$, where N_f is the number of features. Compositional information is encoded in BPNNs by initialising multiple separate neural networks $(f^{(Z_1)}, \ldots, f^{(Z_{N_k})})$, one for each of the N_k different chemical species contained within the dataset. Each of these N_k networks are trained to predict the atomic energies E_i of atom i if and only if the atom's chemical species Z_i matches that of the network. The total energy E_T is then calculated as a summation of all N_{at} atomic energies E_i. For example, consider benzene which contains six atoms of carbon and six atoms of hydrogen, yielding two separate networks $f^{(\mathrm{C})}$ and $f^{(\mathrm{H})}$. Assuming the ith atom is carbon, then its atomic energy E_i is calculated as $E_i = f^{(\mathrm{C})}(\boldsymbol{x}_i)$, where the model $f^{(\mathrm{C})}$ is parametrised by weights $\theta^{(\mathrm{C})}$.

There are a number of benefits to this BPNN architecture that is shown in Figure 3.5. BPNNs are compatible with configurations containing a variable number of atoms and species, due to being able to call the relevant species sub-networks as many times as needed. This allows for training on a wide range of diverse datasets. An additional benefit is that BPNNs are efficient with the number of learnable parameters required. They are still typically deep fully connected neural networks with many thousands of parameters, but the same parameters are shared for each chemical species allowing the model to be kept reasonably small and efficient. Due to being deep neural networks, they have a large learning capacity, able to generalise out to unknown data regimes. Linear network or shallower networks with less layers have a much smaller learning capacity. This learning capacity issue is one of the main reasons that BPNNs are chosen for use in Chapter 5.

3.5.2 SNAP Potential

The Spectral Neighbour Analysis Potential (SNAP) [84] is a simple MLIAP that estimates the energy of a molecule as a linear combination of bispectrum components. It assumes that the bispectrum components sufficiently capture the underlying physics of the system so as to be learned by the unsophisticated linear model. A different

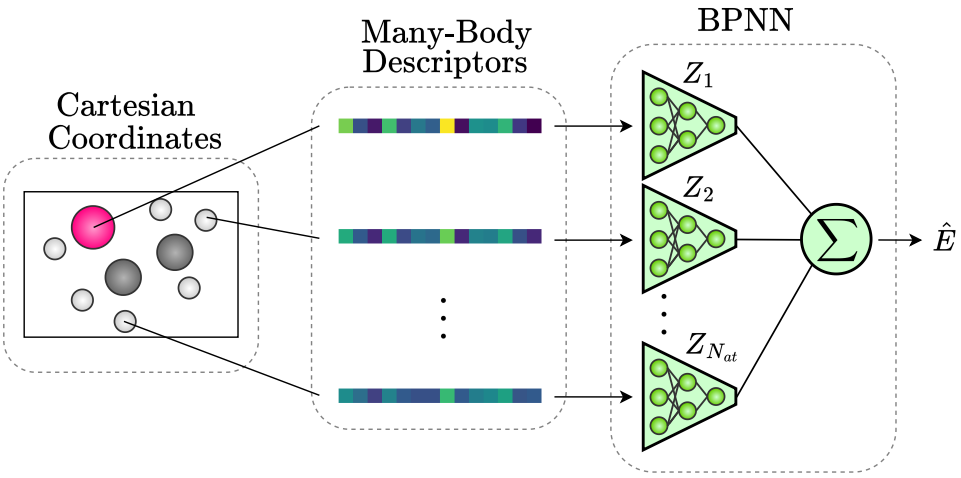

Figure 3.5: Diagram showing the workflow and architecture of a Behler-Parrinello neural network (BPNN) [83, 236]. The Cartesian coordinates of an atomic system are used to generate many-body descriptors for each atom i. These atom descriptors are fed into the BPNN sub-network corresponding to the chemical species Z_i of each of the N_{at} atoms. The atomic energies outputted by each sub-network are then summed as the predicted total energy $\hat{E}$. A Spectral Neighbour Analysis Potential (SNAP) has a very similar workflow, excepting that each sub-network is linear instead of fully connected.

many-body representation could theoretically be used to construct a similar MLIAP. SNAP has proved to be successful at predicting energies, forces, and stress tensors, despite its apparent simplicity [254, 255]. It utilises a kernel ridge regression model (KRR) [206], which assumes that the total energy of a molecule E_T, containing N_{at} atoms, can be decomposed into atomic contributions E_i similar to BPNNs in Equation (3.34). In a system with N_k different species, the atomic energy E_i is estimated using bispectrum components $\boldsymbol{x}_i$ for atom i as follows:

$$E_i = w_0^{(k_i)} + \boldsymbol{w}^{(k_i)} \cdot \boldsymbol{x}_i \ , \tag{3.35}$$

where k_i is the chemical species of atom i and $\boldsymbol{w}_s \in \mathbb{R}^{N_k \times N_f}$ are linear coefficients with additional bias parameters $w_0^{(k_i)}$. It is noted that SNAP models are extensible to systems with a variable number of atoms due to parameters $\{w_0^{(k_i)}, \boldsymbol{w}^{(k_i)}\}$ being reused for all atoms of that species k_i. The total energy E_T can be then calculated as

a sum of these atomic energy contributions E_i as

$$E_T = \sum_{i=1}^{N_{at}} \left(w_0^{(k_i)} + \boldsymbol{w}^{(k_i)} \cdot \boldsymbol{x}_i \right) . \tag{3.36}$$

The linear coefficients $\{w_0, \boldsymbol{w}\}$ are randomly initialised, and are subsequently optimised so as to minimise the mean-squared error between the predicted energies E_T and the actual DFT energies E. In a KRR model, an additional regularisation term is included to prevent overfitting, meaning the values of the parameters $\{w_0, \boldsymbol{w}\}$ are also minimised, typically using an L2 norm [256]. The strength of this regularisation is controlled using a hyperparameter α. It is also possible to predict the atomic forces and stress tensors using the derivatives of the total energy E_T with respect to the atomic positions $\boldsymbol{r}$. The parameters $\{w_0, \boldsymbol{w}\}$ are commonly optimised by minimising a combination of energy, force, and stress losses, summed together with respective coupling hyperparameters.

3.6 Summary

The objective of this thesis is to use generative-modelling techniques to create new atomic systems. The relevant machine-learning (ML) methods were previously detailed in Chapter 2. The underlying physics of atomic systems are described in this chapter, from a full quantum-mechanical (QM) perspective. The Schrödinger equation [189] was introduced, as well as the non-relativistic Hamiltonian operator for many-body systems [79]. Using the Born-Oppenheimer approximation [190], the electronic Schrödinger equation was decoupled, allowing for the exploration of the potential energy surface (PES) [49]. Density functional theory (DFT) [80–82] was introduced as a computationally feasible means of calculating the properties of systems containing many electrons and nuclei.

In this thesis, electronic structure calculations are performed at various different stages. In the creation of the initial ML training datasets, each configuration is labelled with DFT properties. The atomic systems outputted by the trained generative models are also verified using DFT. ML models are used instead of DFT inside the inner training loop of a generative model, due to being much more computationally inexpensive. Therefore, this chapter also detailed the techniques used in the construction of these ML structure-to-property models [83,84]. Of primary importance are the feature vectors used to encode many-body interactions within the local atomic environment around each atom. Bispectrum components [75] were selected for use in the accurate prediction of QM properties throughout this thesis. Different structure-to-property models that predict QM properties given these many-body descriptors were then detailed. Simple linear SNAP models were contrasted with the deeper BPNNs.

Chapter 4

Local Many-Body Inversion

A method to invert local many-body representations back into Cartesian coordinates was developed in collaboration with Matteo Cobelli, and is presented here. This chapter will first detail how the inversion algorithm works. Then it will demonstrate its efficacy by inverting real samples from molecular dynamics trajectories. The method will then be applied to its intended use case of generative modelling in Chapter 5.

4.1 Introduction

The main objective of this thesis is to apply generative-modelling techniques to chemical systems. If trained correctly, the generative model will learn the underlying dataset probability distribution, and can then be used to efficiently generate novel data samples. It has been shown that leveraging three-dimensional structural information about atomic positions can improve the accuracy of a machine-learning (ML) model when predicting quantum-mechanical properties [220–222]. This motivates the development of a generative model that is capable of learning from this structural data. Configurations could then be generated for selective values of these quantum-mechanical properties when using a conditional generative model, see Section 2.4.3.

The choice of how to encode chemical systems into suitable feature vectors is of crucial importance. A configuration containing N_{at} atoms is initially described using Cartesian coordinates $\boldsymbol{r} \in \mathbb{R}^{N_{at}\times 3}$. This would be an inefficient input into any ML model, and is therefore converted into some other representation $\boldsymbol{x} \in \mathbb{R}^{N_{at}\times N_f}$, where N_f is the number of features. The most accurate structure-based chemical represen-

tations are many-body representations [75, 207, 208, 230–232]. These representations describe the local chemical environment around each atom as a series of interactions between various neighbouring atoms, as can be seen in Figure 3.2. There exist a set of translational, rotational, and permutational transformations that leave all internal bond distances and angles the same, see Figure 3.1. These transformations change the values of $\boldsymbol{r}$ but in reality the molecule remains physically unchanged. Therefore, an effective set of many-body descriptors $\boldsymbol{x}$ should be invariant under these three transformations. Another benefit of many-body representations is that they are local, with nearby atoms being given a higher weight as compared to the distant neighbours, which do not contribute. This locality allows systems with variable sizes to be modelled. The bispectrum components [75, 84] many-body representation are used throughout this work.

A challenge with generative modelling is that whatever representation the training data is in, the generated data will also be in. This means that a representation is suitable if and only if there exists a method to invert it back into its original, human-interpretable form of Cartesian coordinates. A many-body representation must therefore be made invertible, with the Cartesian coordinates $\boldsymbol{r}$ able to be reconstructed as $\tilde{\boldsymbol{r}}$ given the many-body descriptors $\boldsymbol{x} = B(\boldsymbol{r})$. Having such an inversion method would allow a generative model to be trained using many-body descriptors as features. The generated many-body samples would then be inverted back into Cartesian coordinates for analysis and verification. In this chapter, no generative modelling is performed, only real configurations taken from a molecular dynamics trajectory [194, 257] are inverted. This is in order to verify that the proposed inversion scheme works, before applying it to its intended use case of generative modelling in Chapter 5.

This inversion problem is a difficult one to solve and only preliminary work has been taken in this direction. Many-body representations are local in nature, with the descriptors for each atom i only considering neighbouring atoms contained within a sphere of some cutoff radius r_c. Constructing a method to find the global solution to this problem, with a one-to-one mapping from many-body descriptors back to Cartesian coordinates is beyond the scope of this work. The local inversion method, proposed here, only represents a first step towards the open problem of a general inversion scheme.

The structure of this chapter is now detailed. First, the main inversion algorithm is outlined, with all relevant equations defined. The next section details an improvement made to the initialisation of the algorithm. The efficacy of the method is verified in Section 4.4 by analysing both the physical and the electronic structure of the inverted configurations. A summary of the chapter is then provided in the conclusions section.

4.2 Inversion Algorithm

This proposed method considers the local inversion of many-body descriptors as a structure optimisation problem. The initial idea and implementation for this method was developed by Dr Matteo Cobelli. My contributions include rigorous testing, assisting in the random noise term, improved coding implementation, multiple initial configuration version, and electronic structure convergence studies. An initial atomic configuration $\boldsymbol{r}^{(0)}$ is optimised by means of gradient descent [105] until its corresponding descriptors $\boldsymbol{x}^{(n)} = B\left(\boldsymbol{r}^{(n)}\right)$ at the nth iteration are equal to the target descriptors $\tilde{\boldsymbol{x}}$ within some numerical tolerance. The target descriptors $\tilde{\boldsymbol{x}}$ are those which are to be inverted. A loss function $L(\boldsymbol{r})$ is defined to compare the target descriptors with the optimised descriptors. The optimum atomic configuration $\tilde{\boldsymbol{r}}$ is that which minimises this loss function. Using the equilibrium atomic configuration as the starting point $\boldsymbol{r}^{(0)}$ for the optimisation procedure was initially thought to require the fewest iterations. The method is designed for the target descriptors $\boldsymbol{x}$ to be the outputs of a generative model. However, they can also be from known molecules in order to to verify that the mapping $B : \boldsymbol{r} \rightarrow \boldsymbol{x}$ has been successfully inverted as $B^{-1} : \boldsymbol{x} \rightarrow \tilde{\boldsymbol{r}}$ where $\tilde{\boldsymbol{r}} \approx \boldsymbol{r}$. The loss function at the nth iteration is defined as

$$L\left(\boldsymbol{r}^{(n)}\right) = \frac{1}{N_k} \sum_{k=1}^{N_k} \left(\sum_{s \in \mathcal{S}_k} \boldsymbol{x}_s^{(n)} \; - \sum_{s \in \mathcal{S}_k} \tilde{\boldsymbol{x}}_s \right)^2 , \tag{4.1}$$

where the external sum runs over all N_k distinct chemical species, and the internal sum runs over the set $\mathcal{S}_k$ of indices of atoms belonging to species k. This loss function has been constructed to be permutationally invariant, meaning that the order of the

atoms in $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ are not required to be the same. As the loss function sums up the descriptors for each chemical species, all that is required for the target descriptors is the summed descriptors $\tilde{\boldsymbol{x}} \in \mathbb{R}^{N_k \times N_f}$ instead of the previous $\tilde{\boldsymbol{x}} \in \mathbb{R}^{N_{at} \times N_f}$, where N_{at} is the number of atoms and N_f is the number of features. This reduction in size of the target descriptors means that fewer parameters are needed within the generative model.

Using gradient descent, the coordinates for each atom i in the initial atomic configuration $\boldsymbol{r}^{(0)} \in \mathbb{R}^{N_{at} \times 3}$ are updated from the nth iteration to the $(n+1)$th iteration as follows:

$$\boldsymbol{r}_i^{(n+1)} = \boldsymbol{r}_i^{(n)} - \gamma \nabla_i L\left(\boldsymbol{r}^{(n)}\right) + \eta\, e^{-\nu n} \boldsymbol{\varepsilon}_{\text{noise}} \,, \tag{4.2}$$

where γ is the learning rate. An additional noise term is also added in order to combat the tendency for the method to become trapped in local minima of the loss function. An exponential decay of the noise was found to improve the rate of convergence and the stability of the algorithm. The strength of the noise term is controlled by the coupling constant η and the lifetime of the exponential decay by ν. The noise vector $\boldsymbol{\varepsilon}_{\text{noise}} \in \mathbb{R}^3$ is drawn from the uniform distribution $\boldsymbol{\varepsilon}_{\text{noise}} \sim \mathcal{U}[-1, 1]$.

The derivative of the loss function $L\left(\boldsymbol{r}^{(n)}\right)$ with respect to each Cartesian direction $\boldsymbol{r}_x$, $\boldsymbol{r}_y$, and $\boldsymbol{r}_z$ for each atom i can be written as follows,

$$\nabla_i L\left(\boldsymbol{r}^{(n)}\right) = -\frac{2}{N_k} \sum_{k=1}^{N_k} \left[\left(\sum_{s \in \mathcal{S}_k} \boldsymbol{x}_s^{(n)} \; - \sum_{s \in \mathcal{S}_k} \tilde{\boldsymbol{x}}_s \right) \cdot \sum_{s' \in \mathcal{S}_k} \nabla_i \boldsymbol{x}_{s'}^{(n)} \right] \,, \tag{4.3}$$

where $\nabla_i \boldsymbol{x}^{(n)}$ are the derivatives of $\boldsymbol{x}^{(n)}$. The choice of many-body descriptors is limited to those which have easily calculable derivatives. There exist open-source packages [258,259] that have implemented calculations of descriptors and their derivatives.

Since the same many-body descriptors $\boldsymbol{x}$ can be calculated using atomic configurations $\boldsymbol{r}$ under all rotations or translations, it means that the mapping $B : \boldsymbol{r} \to \boldsymbol{x}$ is not globally invertible due to the fact there is not a one-to-one bijective mapping. For every possible rotation and translation of $\boldsymbol{r}$, there exists a global minima. The many-body descriptors used in this work, namely bispectrum components [75], can be considered incomplete [230] because it is possible that two distinct configurations (not just related by rotation or translation) can yield the same descriptors,

see Section 3.4.4. This is because bispectrum components only consider up to 4-body interactions, see Section 3.4.3, based on interatomic angles between triplets of atoms using a spherical harmonic basis. This can be alleviated by using more advanced, complete representations such as atomic cluster expansion [207] or Jacobi-Legendre polynomials [208] which are capable of including 4-body terms or higher. The inversion problem can be considered highly non-convex and challenging to solve with simple gradient descent. More sophisticated optimization techniques should be considered in future work.

4.3 Multiple Initial Configurations

Using a simple gradient descent algorithm to solve a non-convex optimisation problem is not ideal. The loss landscape is expected to contain many saddle points on which gradients will vanish and further optimisation will not occur. The large number of local minima provides further difficulties. Non-convex optimisation problems are common within the field of machine-learning, as optimising a neural network is just such a problem. Advanced versions of the gradient descent algorithm such as adam [106], rmsprop [107], and adagrad [108], which use adaptive learning rates and momentum are almost universally employed. Other techniques that can be used to solve non-convex optimisation problems include evolutionary algorithms [260], Bayesian optimisation [235, 261], and simulated annealing [262]. Integrating these more advanced optimisation techniques within this local inversion method raises challenges, particularly since the workflow depends on the use of external packages [238, 258, 263] to calculate the descriptors and their derivatives.

The initial atomic configuration $\boldsymbol{r}^{(0)}$ was chosen by default to be the equilibrium configuration as it seemed to make the method converge fastest. For molecules with torsional degrees of freedom, such as ethanol, it was found that both the gradient term and the noise term in the loss function of Equation (4.1) were unable to efficiently explore across these spaces. A simple and naive solution to this problem was employed: using multiple initialisations [264] with different initial atomic configurations. These initial configurations would ideally span the torsional configuration space evenly. A suitable set of initial configurations can be manually created using packages such

as Avogadro [265, 266] which allows bond lengths, angles, and torsions to be freely manipulated. For systems with a large amount of torsional degrees of freedom, a method to automatically suggest plausible initial configurations was necessary over the manual Avogadro method.

Given a dataset $\mathcal{N}$ which contains N atomic configurations, it is proposed to automatically suggest initial configurations by selecting the maximally diverse subset $\mathcal{M}$. Computing the diversity of a set of configurations is commonly achieved by first calculating descriptors $\boldsymbol{x}$, then defining a pairwise similarity metric between these descriptors, and finally, quantifying how similar each configuration is to every other configuration as a matrix $\boldsymbol{K} \in \mathbb{R}^{N\times N}$. This is the outline for one method commonly used [267, 268] to calculate diversity, but many other methods exist [269–271] such as using the root mean square deviation of Cartesian coordinates [272].

The similarity metric used in this work is based on the SOAP kernel method [230], which uses different descriptors to the bispectrum components employed in this work, but the metric should, in principle, work for any many-body descriptor. Each element K_{ij} of the matrix $\boldsymbol{K}$ compares the descriptors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ for atomic configurations i and j as follows,

$$K_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\frac{\boldsymbol{x}_i \cdot \boldsymbol{x}_j}{\sqrt{(\boldsymbol{x}_i \cdot \boldsymbol{x}_i)(\boldsymbol{x}_j \cdot \boldsymbol{x}_j)}} \right)^{\zeta} , \tag{4.4}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are both flattened from a $N\times N_{at}$ matrix into a $N*N_{at}$ vector with N_{at} being the number of atoms in each configuration. The integer exponent $\zeta \geq 2$ controls the sensitivity of the kernel with respect to the variations in atomic positions, where a default of $\zeta = 4$ is used for this work. Equation (4.4) is a positive definite function, which outputs one if $\boldsymbol{x}_i = \boldsymbol{x}_j$ and tends towards zero as the similarity decreases.

Finding the maximally diverse subset of molecules $\mathcal{M}$ given this similarity matrix $\boldsymbol{K}$ is the more challenging problem, due to the number of calculations needed for a brute force approach [267]. A greedy algorithm [273] is therefore used to address this computational inefficiency. The first atomic configuration to be added to the subset $\mathcal{M}$ is that which has the minimum value in the sum of matrix $\boldsymbol{K}$ along each row $\sum_{i=1}^{N} K_{ij}$. This can be considered the single most diverse configuration in the dataset $\mathcal{N}$. The brute force approach would be to remove this from $\mathcal{N}$, calculate another

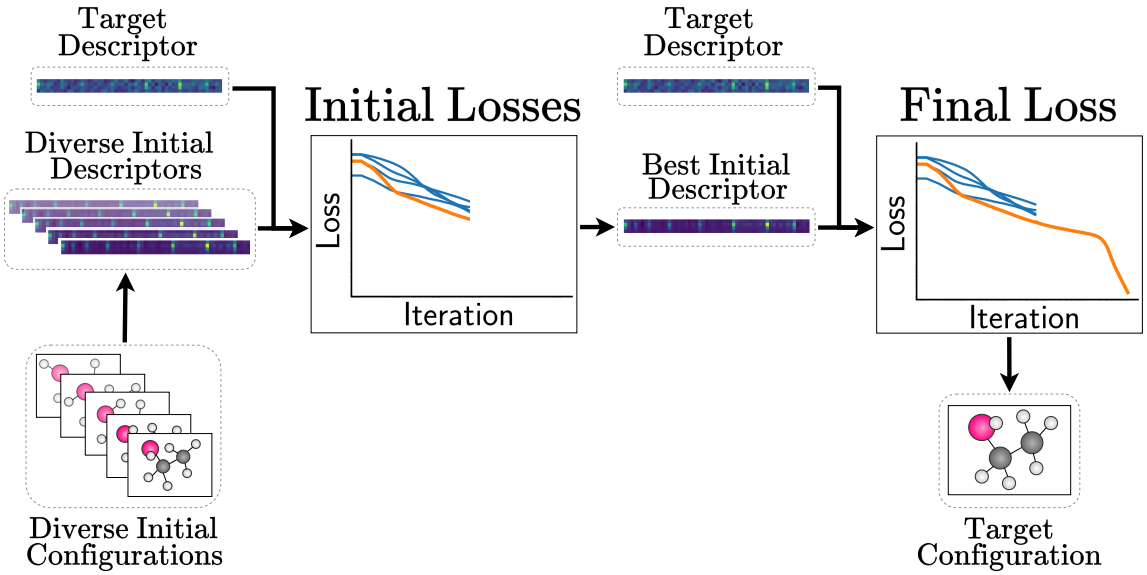

Figure 4.1: Diagram of the workflow of the proposed inversion method. It is desired to find the configuration that corresponds to some target descriptors. The maximally diverse configurations from a molecular dynamics trajectory are used as initial guesses. At each iteration, the positions of the initial configuration are updated, see Equation (4.2), so as to minimise the difference between the target descriptors and the initial descriptors, see Equation (4.1). After N_{init} iterations, the initial configuration with the lowest loss is selected, and the procedure is continued for a total of N_{iter} iterations. If convergence is reached, the resulting target configuration has been updated sufficiently that its descriptors are approximately equal to target descriptors. Therefore, the target descriptor has been inverted, with an appropriate configuration found.

similarity matrix $\boldsymbol{K} \in \mathbb{R}^{(N-1)\times(N-1)}$, and select the new most diverse configuration. This would be repeated until M configurations have been added to $\mathcal{M}$. This approach would yield the optimum set $\mathcal{M}$ but would involve calculating K_{ij} from Equation (4.4) a total of $\prod_{k=0}^{M-1}(N-k)^2$ times, making it computationally infeasible.

The more efficient, greedy algorithm [268] for determining the maximally diverse subset $\mathcal{M}$ is used in this work. It works by first determining the single most diverse configuration, adding it to $\mathcal{M}$ and removing it from $\mathcal{N}$ as before. The next configuration added to $\mathcal{M}$ is the maximally diverse one from the configurations already within $\mathcal{M}$. This means that a prospective configuration need only be compared to the subset $\mathcal{M}$ as opposed to the entirety of set $\mathcal{N}$.

Every element in the diverse subset $\mathcal{M}$ are to be used as initial atomic configurations for the loss procedure. Each iteration involves calculating Equation (4.1) and Equation (4.2) in order to update the positions of the atomic configuration from the last iteration. A total of N_{init} iterations are performed, for each of the M initial configurations. After which, the initial configuration with the lowest loss value from Equation (4.1) is chosen. It is desirous that this chosen initial configuration has torsions similar to those of the target configuration. The loss procedure is then restarted anew, using the chosen initial configuration after the N_{init} iterations already performed. A diagram of the full workflow is included in Figure 4.1.

4.4 Results

The dataset selected for use in the following sections is the revised MD-17 dataset [87]. The original MD-17 dataset [274–276] consists of long molecular dynamics (MD) trajectories for ten separate small molecules, including ethanol, benzene, and aspirin. The number of conformations in each trajectory varies from 99,999 for azobenzene to 627,983 for benzene. The number of atoms ranges from nine in ethanol and malonaldehyde to 24 in azobenzene. The MD trajectories are calculated at a resolution of 0.5fs and a temperature of 500K. Each conformation is labelled with energies and forces calculated using density-functional theory (DFT) [50,80–82] at the PBE+vdW-TS [198, 277] level of theory, see Section 3.3 for more details. This is a benchmark dataset that has been widely used, but has since been found to contain high levels of numerical noise [87]. Hence, the revised MD-17 dataset [87] was created to reduce this numerical noise, for a subset of 100,000 random samples from each trajectory. This revised dataset reperforms all DFT calculations at the PBE/def2-SVP [198, 278] level of theory using very tight convergence criteria and dense DFT integration grids.

The general workflow for analysing the results of this local inversion method is as follows. A number of atomic configurations $\boldsymbol{r}$ are sampled from the chosen dataset, the corresponding many-body descriptors $\boldsymbol{x} = B(\boldsymbol{r})$ are calculated, which are then inverted back into Cartesian coordinates $\tilde{\boldsymbol{r}}$. The efficacy of the inversion method is examined by comparing the original configurations $\boldsymbol{r}$ to the inverted ones $\tilde{\boldsymbol{r}}$. Various forms of comparisons are used in the following sections, such as through pair and an-

Molecule Type	Formula	Number of Atoms
Ethanol	C_2H_6O	9
Malonaldehyde	$C_3H_4O_2$	9
Benzene	C_6H_6	12
Uracil	$C_4H_4N_2O_2$	12
Toluene	C_7H_8	15
Salicylic Acid	$C_7H_6O_3$	16
Naphthalene	C_10H_8	18
Paracetamol	$C_8H_9NO_2$	20
Aspirin	$C_9H_8O_4$	21
Azobenzene	$C_{12}H_{10}N_2$	24

Table 4.1: A table showing the ten different molecule types contained within the revised MD-17 dataset [87].

gular distribution functions. The electronic structure of the original and the inverted configurations are also compared using DFT calculations.

The exact methodology used to perform all the local inversion calculations is now detailed. Bispectrum components [75] are used as the many-body descriptors with $2j_{\max}=8$ and $r_c=3.6$Å. All calculations of bispectrum components are performed in LAMMPS [238, 239]. The parameters in the gradient descent algorithm in Equation (4.2) are set as $\gamma=3\times10^{-7}$Å^2, $\eta=1\times10^{-2}$Å, and $\nu=1\times10^{-3}$. The loss procedure is ran for a total of $N_{iter}=1\times10^6$ iterations, with $M=50$ different initial configurations, each of which undergoes $N_{init}=500$ iterations.

4.4.1 Pair and Angle Distribution Functions

We have two sets of atomic configurations, and need to tell if they are equivalent. Each pair of corresponding original and inverted configurations should be identical, if the inversion method works as intended. Therefore, the most straightforward way to compare them would be to align the configurations by performing the necessary translations and rotations so as to minimise the root-mean-square deviation (RMSD) between them. This can be easily achieved using the rmsd package [279, 280] or the RDKit package [281]. After alignment, the remaining RMSD deviation quantifies how close the two configurations match. This analysis is not performed here in this chapter, due to the following methods being considered sufficient.

Examining the pair and angular distribution functions are other ways to verify the

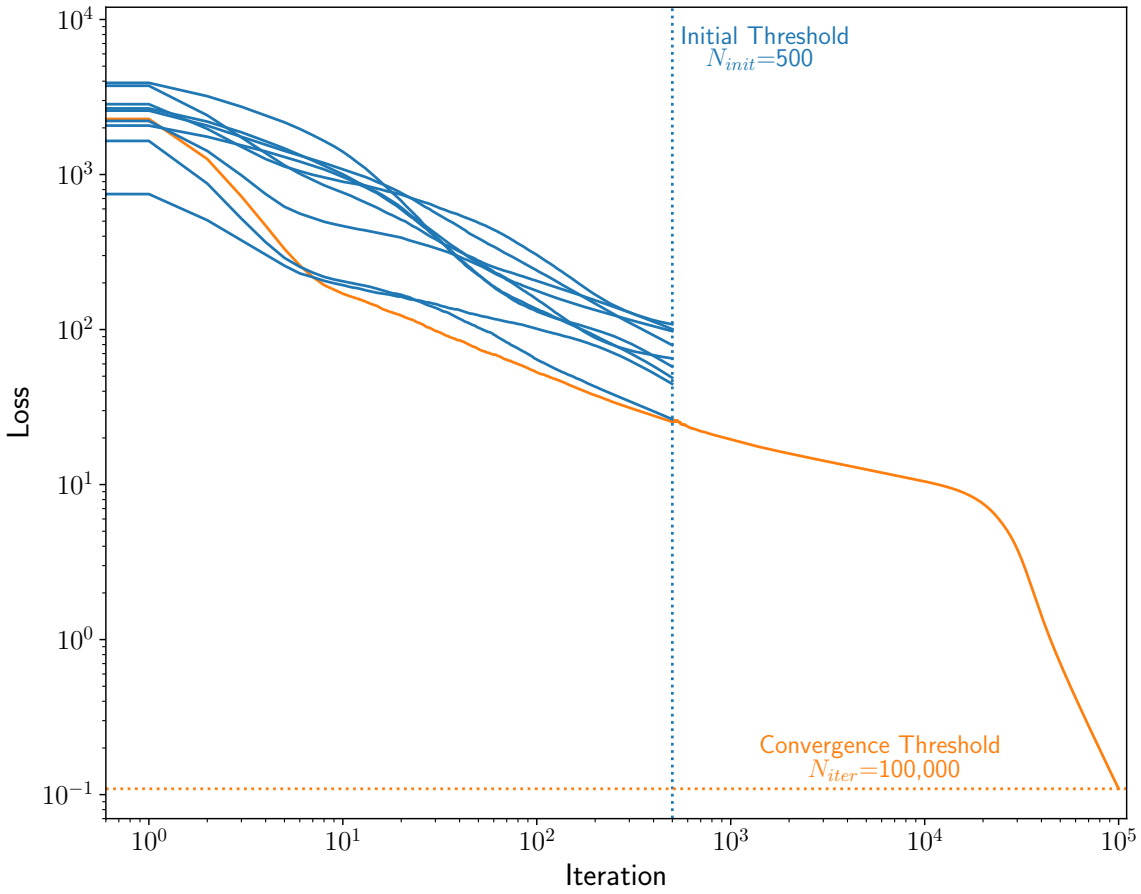

Figure 4.2: Plot of the loss function of Equation (4.1) at each iteration throughout the local inversion method. The loss procedure is rerun multiple times using different, diverse initial configurations for $N_{init} = 500$ iterations. The configuration with the lowest loss is then used to continue the loss procedure until it reaches either convergence or the maximum number of iterations N_{iter} of 100,000.

validity of the inverted configurations. The pair distribution function considers the distances between each pair of atoms in a set of configurations, and quantifies how often they occur. The angular distribution function measures the distribution of the set of angles formed by all possible atom triplets within a configuration. The package auto-FOX [282] is used to calculate the angular distributions. An infinite cutoff radius is used with all chemical species being given an equal weighting. As both pair and angular distributions are internal functions, invariant to translations and rotations, there is no need to align the configurations by minimising the RMSD as before. It is hypothesised that, if the pair and angular distribution functions for the original and the inverted configurations closely align, then the local inversion method can be

considered successful.

The pair distribution function can be visualised either as the partial function, where each chemical species combination (e.g., H-H, C-H, C-C) is individually displayed or as the total pair distribution function summed over all species combinations. The total pair distribution functions for all ten molecules types included in the revised MD-17 dataset are shown in Figure 4.3. Within this plot, a large sample of $10,000$ real configurations are used for each molecule type in order to best estimate the ground truth distributions. In comparison, only a small sample of 200 inverted configurations are used. From this plot it is clear than the inversion method is capable of producing configurations with a very similar set of pair distances as the real configurations.

There exist various molecular descriptors that are constructed only using interatomic distances [283]. These representations can be termed as 2-body descriptors. They are not ideal for use in learning potential energy surfaces due to being incomplete. Incompleteness refers to the possibility for multiple different distinct configurations to have the same set of internal interatomic distances [233]. Therefore, in order to distinguish between many-body descriptors, one must also examine internal angles or higher-order body terms. For this reason, the angular distribution functions are here examined. All possible triplet angles are considered, centered on each species (e.g., X-C-X, X-H-X). The total angular distribution function is the summation of all these angles over every combination of species.

The total angular distribution functions for the same ten molecule type datasets [87] are shown in Figure 4.4. From this figure, it can be observed that the distributions for real and inverted configurations are very similar for all molecule types. There exist minor differences, particularly in the cases of aspirin and azobenzene, the most complex of the molecule types. However, it can be concluded that the local inversion method has the ability to produce configurations containing realistic internal angles.

The local inversion method struggles to perform when the torsional angles of the initial configuration are different to that of the target configuration. This motivated the development of the method to use multiple initial configurations with different torsional angles. The molecule ethanol provides a simple test case to study the inversion method's ability to resolve torsion angles. The partial pair and partial angular distribution functions for original and inverted configurations of ethanol are shown

in Figure 4.5. From the figure, it can be seen that the inverted configurations of ethanol are similar to the original ones from the training set for all pair and angle combinations. The C-C bond displays the most deviation which can be explained due to a smaller sample set size; each ethanol configuration has only one C-C bond as compared to, for example, 12 C-H bonds. Despite the tendency of the inversion method to struggle converging across torsional space, it is capable of doing so in the simple test case of ethanol.

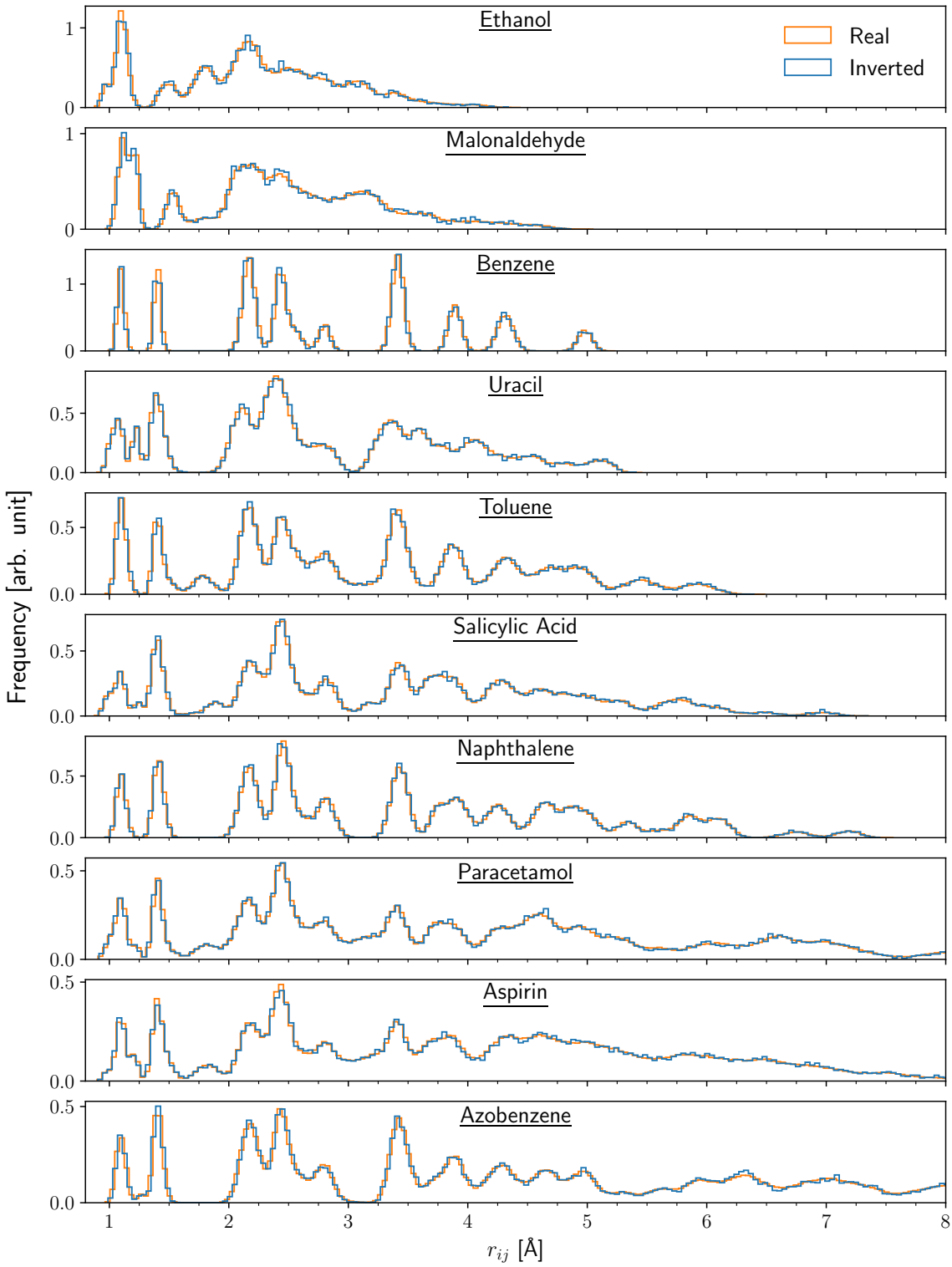

Figure 4.3: Plot of the total pair distribution functions for all ten molecules types included in the revised MD-17 dataset [87]. This function considers every possible distance between atoms i and j in a configuration. The real configurations are taken directly from this dataset. The inverted configurations are attempts to reconstruct these real configurations given their corresponding many-body descriptors. The real distribution contains $10,000$ samples while the inverted one contains 200.

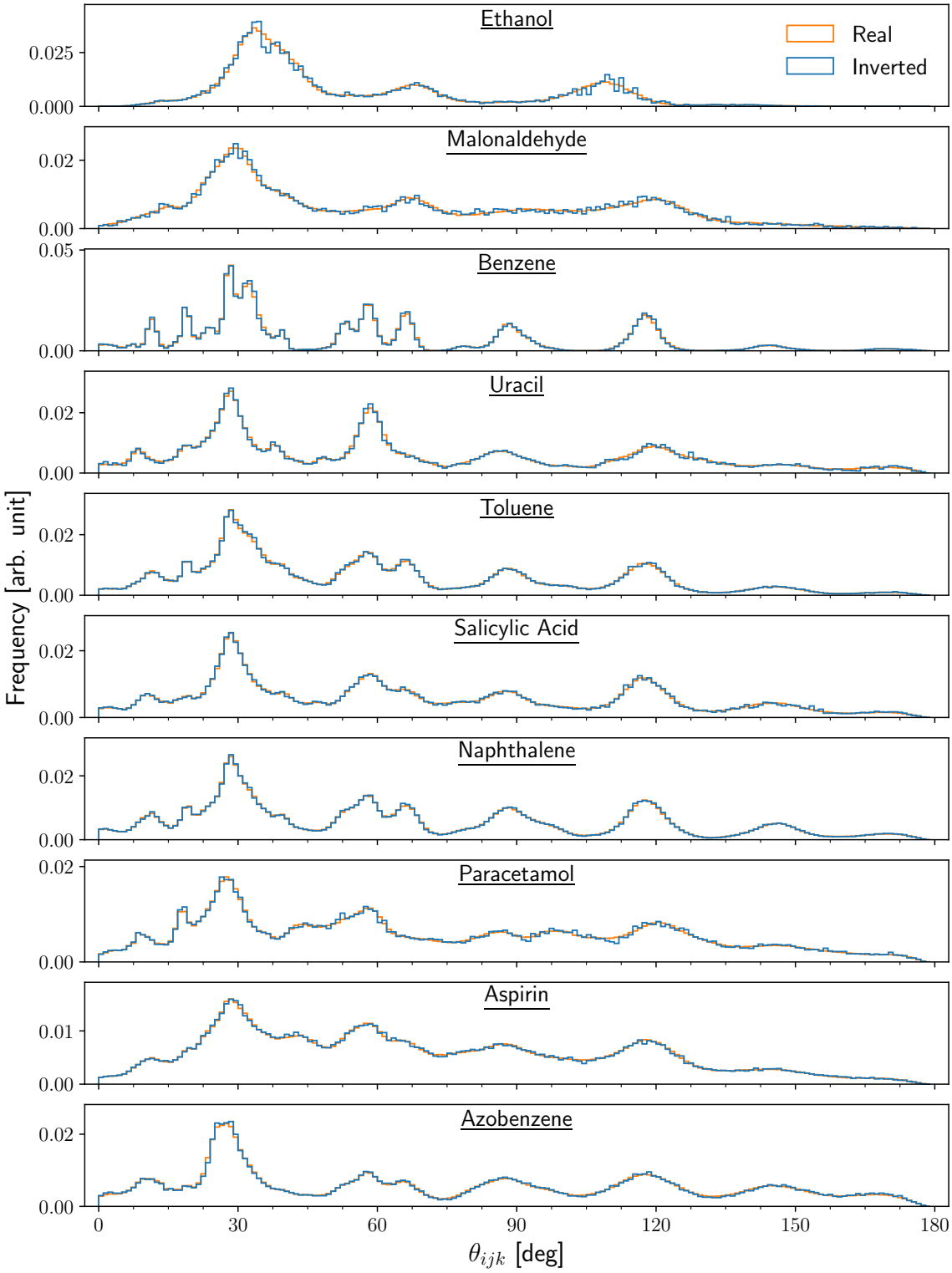

Figure 4.4: Plot of the total angular distribution functions for all ten molecules types included in the revised MD-17 dataset [87]. This function is over all possible triplet angles θ_{ijk} between atoms i, j, and k. The real configurations are taken directly from this dataset. The inverted configurations are attempts to reconstruct these real configurations given their corresponding many-body descriptors. The real distribution contains $10,000$ samples while the inverted one contains 200.

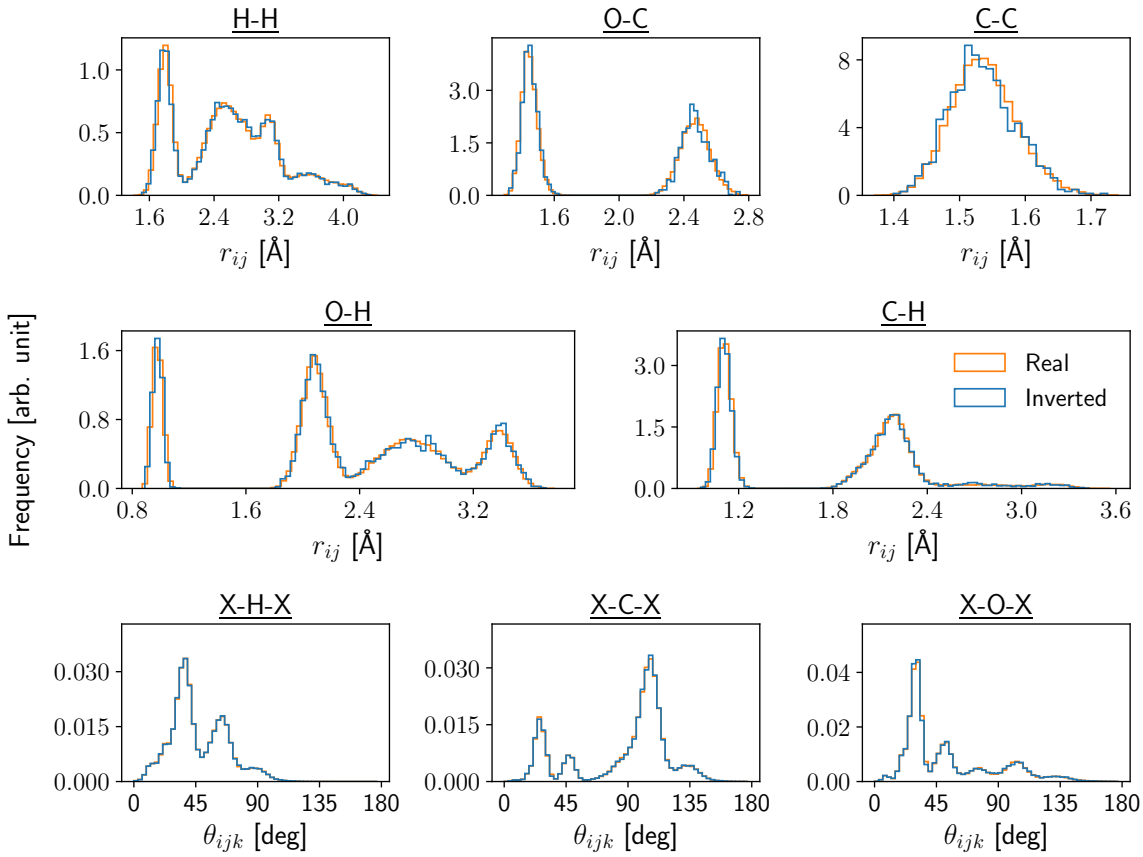

Figure 4.5: Plot of the partial pair and partial angular distributions functions for ethanol configurations from the revised MD-17 dataset [87]. The partial pair distribution function measures the distances r_{ij} between atoms i and j over all possible combinations of chemical species. The partial angular distribution function considers every triplet angle θ_{ijk} between atoms i, j, and k centered on all three species present in ethanol. The real configurations are taken directly from this MD-17 dataset. The inverted configurations are attempts to reconstruct these real configurations given their corresponding many-body descriptors. The real distribution contains $10,000$ samples while the inverted one contains 800.

4.4.2 Electronic Structure

The electronic structures of the original configurations and of the inverted configurations are also compared to verify the efficacy of the inversion method. In order to do this, a DFT calculator is required. The PySCF [42–44] package was chosen for its ease of use. The version of PYSCF used was 2.1.1. PySCF uses Gaussian basis sets [44] instead of plane-waves and is primarily used for molecular systems, with little functionality for periodic boundary conditions [42]. The same PBE functional [198] and def2-SVP basis set [278] were used as in the revised MD-17 dataset [87]. All DFT calculations were performed using atom-centered Gaussian basis sets, with no corrections for van der Waals forces or basis superposition error, and no periodic boundary conditions. The default convergence criteria were used: an energy convergence of 10^{-10} Ha and a maximum of 50 SCF cycles. However, the DFT calculations in the revised MD-17 dataset were performed using ORCA 4.0.1 [284,285]. Therefore, the energies of all of the original configurations included in the analyses here, are recalculated using PySCF. This is done to eliminate any discrepancies that may arise from comparing the results of two different DFT implementations, even if the same functionals and basis sets were used. Both ORCA and PySCF have different default convergence criteria leading to potentially different results. These recalculations using PySCF allow for more accurate analysis of the energies of the inverted molecules.

The convergence of the inverted configurations' energies to the energies of the original configurations is examined. The inversion method has no knowledge of the energies but, if it converges to the correct configuration, then it follows that it should also converge to the correct target energy of the original configuration. The energies of the updated configurations throughout the entire inversion process are shown in Figure 4.6 for a sample of malonaldehyde. This plot tracks the energy convergence of the loss procedure using five different, diverse, initial configurations. It can be seen that some initial configurations closely converge to the target energy, while some become trapped in local minima, unable to correctly converge. These initial configurations that do not converge are those with torsions, bond lengths, and angles very different from those in the target configuration. It is noted that the initial configuration that most closely converges to the target energy is that which has the furthest energy at the start of the loss procedure. This demonstrates the ability of the inversion method to traverse

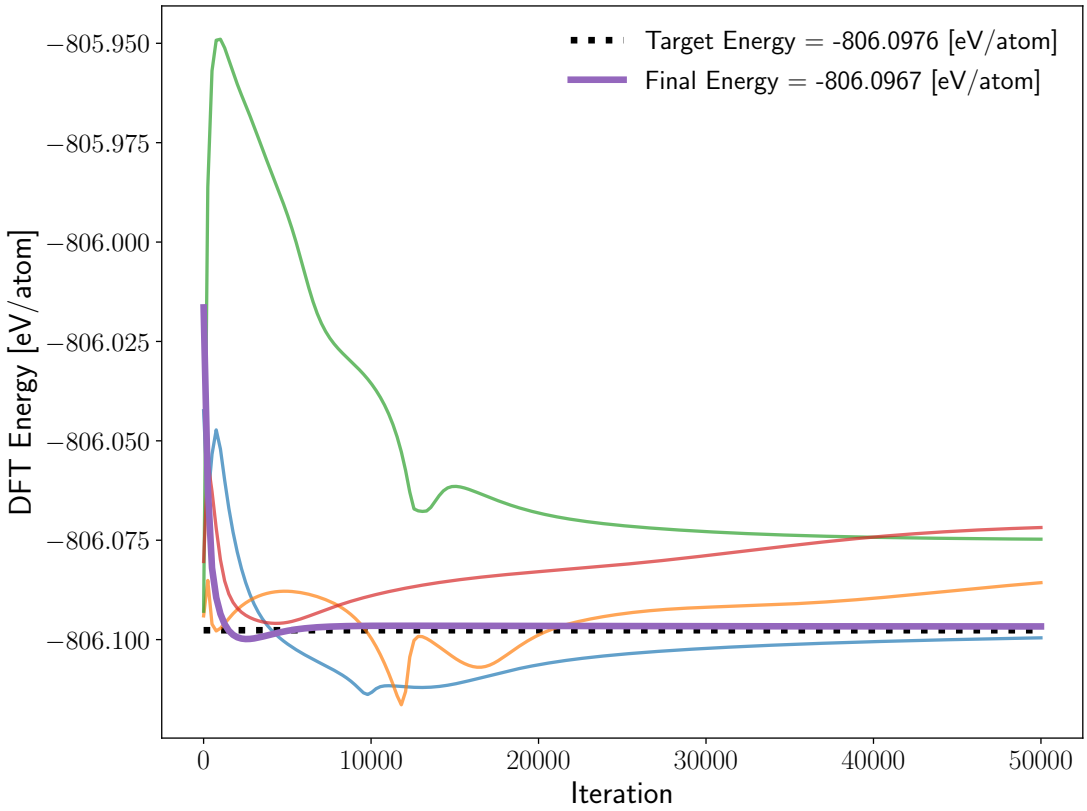

Figure 4.6: Plot of the convergence of the energies of multiple different initial atomic configurations to the target energy throughout the entirety of the loss procedure. All energies are calculated using the DFT package PySCF [42–44]. The target energy is the energy of a configuration of malonaldehyde from the revised MD-17 dataset [87]. The difference between the target energy and the best converged energy at the end of the $50,000$ iterations is $\Delta E = 0.0009$ eV/atom.

across energy landscapes to the correct configuration. Figure 4.6 also showcases the efficacy of using multiple initial configurations. It is clear that the initial configurations with a very different geometry to the target one, have energies that are still diverging even after $50,000$ iterations, compared to the best initial configuration which closely converges in less than $10,000$ iterations.

The ability of the inversion method to correctly converge to the target energy is demonstrated in Figure 4.6, but only for one sample of malonaldehyde. In order to visualise the convergence of multiple samples, parity plots are used. A parity plot is a common tool within the field of machine learning for analysing the accuracy of some set of predictions. The target values are plotted on the x-axis versus the

predictions on the y-axis. If the predictions are perfect, then all points will lie on the trend line. A parity plot of the energy convergence of 200 configurations of ethanol is shown in Figure 4.7. The relevant distributions are included within the parity plot. The residual differences between the original and inverted energies are shown in the bottom section of the plot. Ethanol was chosen due to having among the largest energy ranges of $\sim$ 0.11 eV/atom of all ten molecules types. Ethanol also contains some torsional degrees of freedom, and has only nine atoms present which aides in the speed of computation. From the goodness of fit shown in Figure 4.7 it can be concluded that the inversion method is able to reproduce the original DFT energies, in the case of ethanol, to a remarkable degree of accuracy. The error in the worst prediction (MAX score) is off by only 4.638×10^{-5} eV/atom.

The accuracy of the inversion method in reproducing configurations with the correct energy for the remaining nine molecule types is examined in Figure 4.8. The high level of accuracy in the case of ethanol as seen in Figure 4.7, applies to a lesser degree to the remaining molecules. This is primarily due to the increased structural complexity which affects the ability of the method to reach convergence. For example, after running the inversion method for $100,000$ iterations, the loss $L(\boldsymbol{r})$ from Equation (4.1) for a configuration of ethanol would typically be $\sim2\times10^{-6}$ compared to an azobenzene configuration of $\sim$1. It is expected that the final loss would be further reduced by increasing the number of iterations and the number of initial configurations. It was also expected that molecule types with more torsional degrees of freedom would have a lower accuracy in reproducing target energies. However, since there are no torsional degrees of freedom in benzene, uracil, and naphthalene, it means that there other contributing factors to error than just torsions.

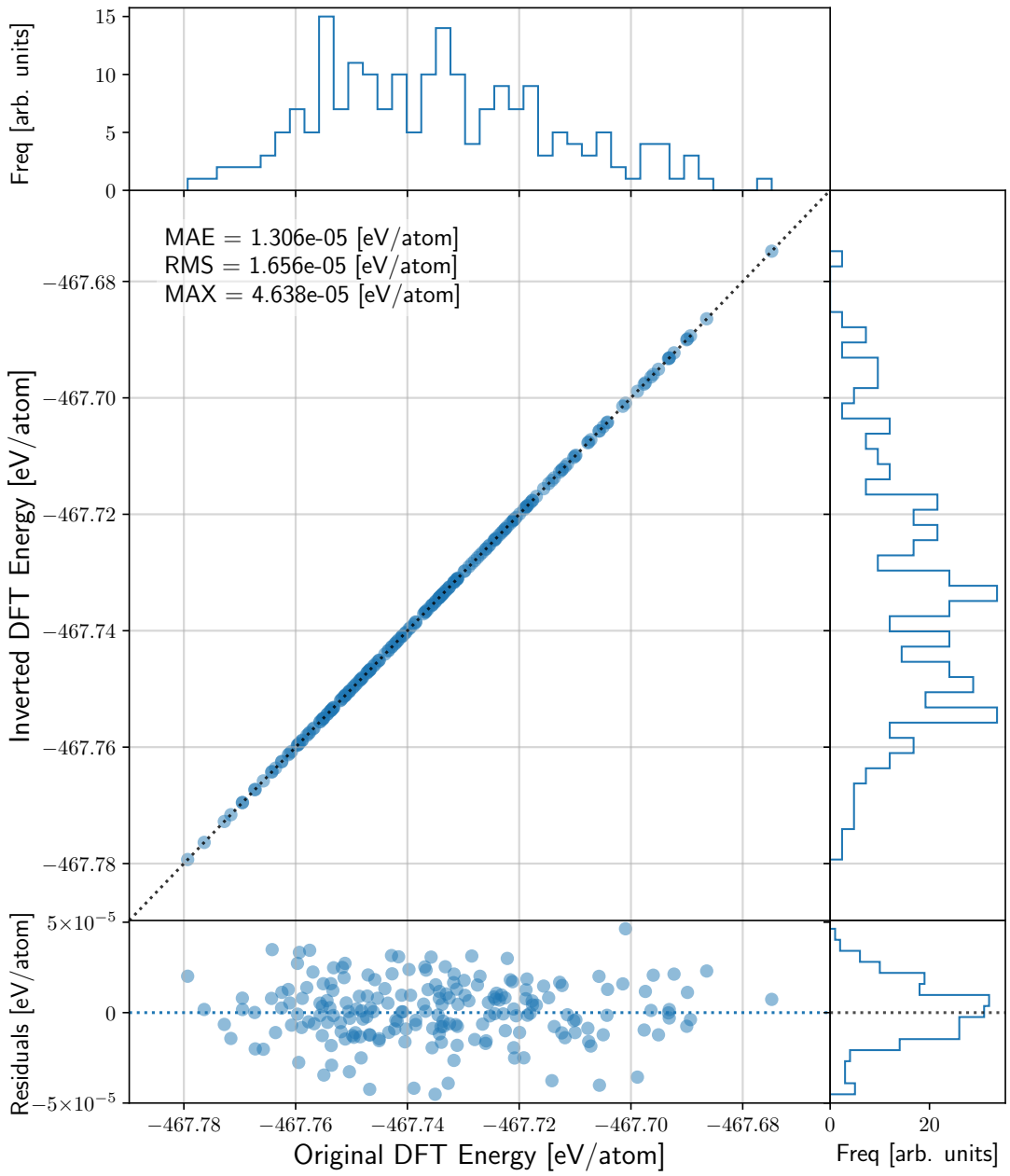

Figure 4.7: A parity plot for ethanol configurations from the revised MD-17 dataset [87]. On the x-axis are shown the DFT energies of these MD-17 ethanol configurations and on the y-axis are the DFT energies of the corresponding configurations produced by the inversion process. The central dotted line represents the perfect fit. A number of metrics are included to quantify the goodness of fit, the Mean Absolute Error (MAE), the Root-Mean-Square error (RMS), and the maximum error of the worst prediction (MAX). The distributions of original and inverted energies are included on the top and right respectively. The difference between the original and inverted energies is included in the residual subplot at the bottom. A total of 200 ethanol configurations are used.

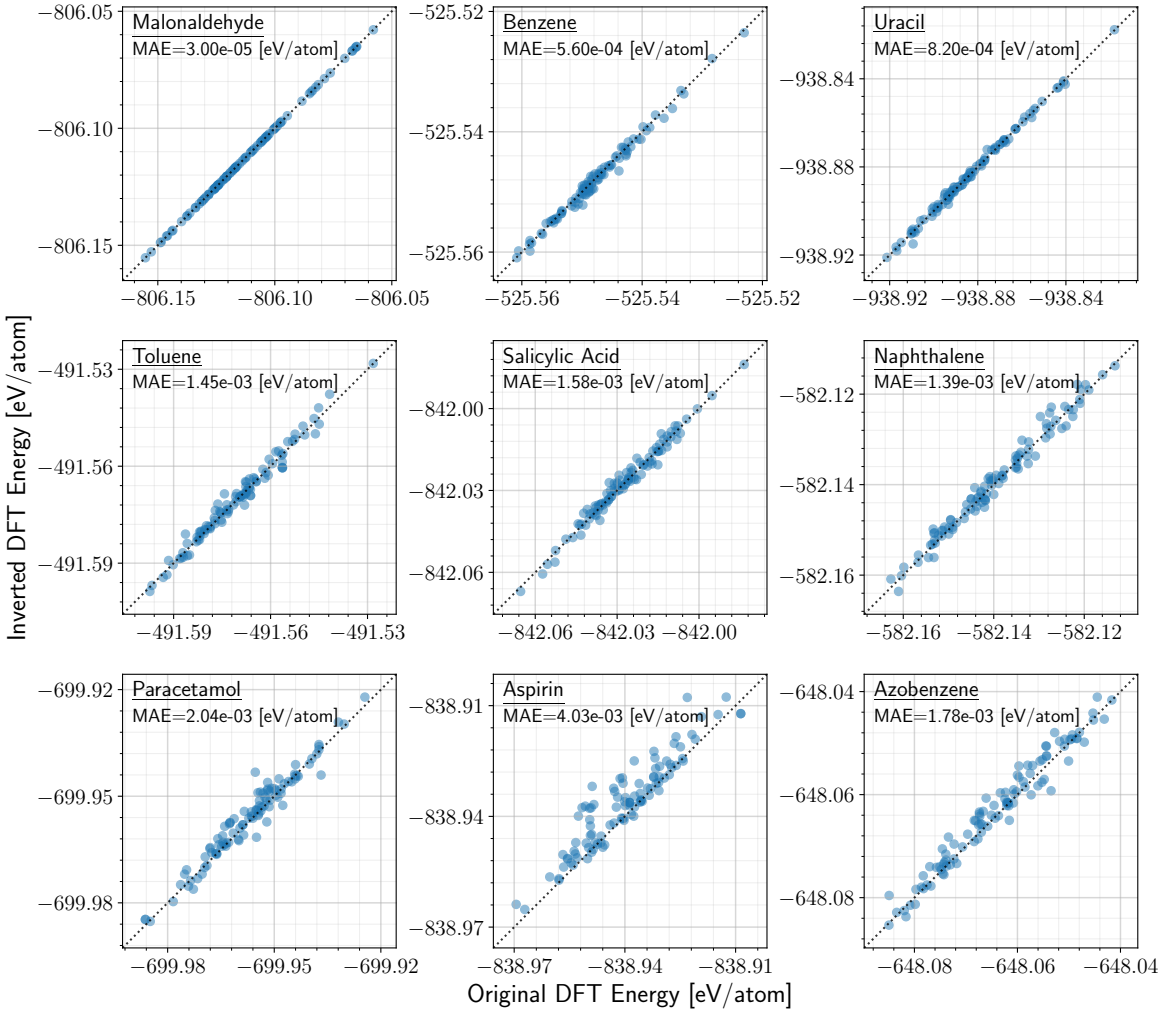

Figure 4.8: Parity plots for nine out of ten of the molecule types in the revised MD-17 dataset [87] excepting ethanol. In each subplot, the x-axis shows the DFT energies of these MD-17 configurations and on the y-axis are the DFT energies of the corresponding configurations produced by the inversion process. The central dotted line represents the perfect fit. The goodness of fit is measured using the Mean Absolute Error (MAE) metric. The molecule types are arranged in order of increasing number of atoms, ranging from nine atoms in malonaldehyde to 24 atoms in azobenzene. A total of 100 samples are shown in each plot.

4.5 Summary

In this chapter, a newly developed method was presented that takes a many-body representation, and inverts it back into a human-interpretable atomic configuration in Cartesian coordinates. It is an iterative, gradient-based method which requires an initial starting configuration. The calculator [238, 239] used for generating the many-body descriptors is called at each iteration, which can make it a somewhat computationally expensive procedure for inverting large numbers of samples. The target descriptors to be inverted and the descriptors of the initial configuration, are compared at each iteration using a loss function, Equation (4.1). The gradients of this loss function are used to update the initial configurations' positions until the method converges. The ability of the method to converge across torsional space was found to improve by determining the most appropriate initial configuration from a diverse subset of configurations. Only one many-body representation was tested, the bispectrum components [75, 84], but the method will theoretically work for all many-body representation which have access to the relevant derivatives.

The efficacy of the method was verified by inverting real configurations, sampled across a MD trajectory dataset [87] for various molecule types. It was determined by examining pair and angular distribution functions that the inverted configurations closely aligned with the original ones, validating the method. The ability of the inversion method to reproduce DFT energies was also analysed. It was found that the energies of ethanol configurations could be replicated with near perfect accuracy. The remaining molecule types also exhibited a high, but lesser, degree of accuracy.

Chapter 5

Generating Molecule Conformations

This chapter is the natural progression of Chapter 4, applying the local many-body inversion method to the descriptors outputted by a generative adversarial network. The motivation for using many-body descriptors is detailed, based on preliminary work. Versions including single and multiple molecule types in the training dataset are analysed. Lastly, a method is developed to allow for the selective generation of atomic configurations with some target property value.

5.1 Introduction

This chapter is concerned with the creation of molecule configurations using generative-modelling techniques. Only conformations or conformers are considered, where bond lengths, and bond rotations are varied, but the chemical composition and overall bonding structure remains the same. It is essential that any atomic configuration is realistic, since they are a basic starting point for understanding and predicting a molecule's (or material's) behaviour. Electronic, thermal, and mechanical properties are all intrinsically based on the spatial arrangement of atoms within any system [50, 194, 257]. Structure-based simulations use these atomic configurations for an initial, in-silico assessment of the performance of a novel compound, without the need for costly experiments [286, 287]. In the field of drug design, the ability of a

candidate drug to successfully interact with the intended target binding site depends on the 3D shape of the molecule [16, 70, 71]. The use of experimental techniques such as X-ray crystallography [288], nuclear magnetic resonance [289], and scanning tunnelling microscopy [290] for determining atomic structure forms the foundation for all subsequent computational work. Traditional methods for determining atomic structure can be very computationally demanding [49]. This motivates the use of generative modelling, which can learn the underlying statistical distribution from a large pre-existing dataset, and can then generate new samples at a fraction of the cost [77]. They are even capable of selectively generating configurations with desired values for some specified property like the total energy [30, 218, 224].

There exist numerous ways to create realistic atomic configurations using tools such as Monte-Carlo sampling [194]. One particularly noteworthy method, that is relevant to this work, is molecular dynamics (MD) [194, 257], which is briefly described here. An MD trajectory consists of some initial configuration that has been evolved over time. Some interatomic potential or force field [248, 251–253] is defined, and is then used to update the positions and the velocities of the atoms at each timestep. The level of accuracy of the chosen force field has a large impact on both the computational cost and the ability of the method to describe the system. If more accuracy is desired, then ab-initio MD (AIMD) [291] can be used, where the interatomic interactions are calculated using first principles quantum-mechanical calculations, typically density-functional theory (DFT) [50, 80–82]. Instead of using an empirical force field to calculate the forces on each atom, AIMD directly solves the electronic Schrödinger equation under the Born-Oppenheimer approximation [190, 292]. Since this work is interested in analysing the electric structure of atomic configurations, AIMD is exclusively used.

The main dataset used in this chapter is the same as in the last, namely the revised MD-17 dataset [87]. The original MD-17 dataset [274–276] is used in Section 5.2 which contains preliminary work. The subsequent sections all utilise the revised MD-17 dataset [87] instead, due to the high level of numerical noise contained within the original dataset. The MD-17 dataset consists of long molecular dynamics trajectories calculated at a resolution of 0.5fs and a temperature of 500K. Every single configuration within each trajectory is labelled with energies and forces calculated with

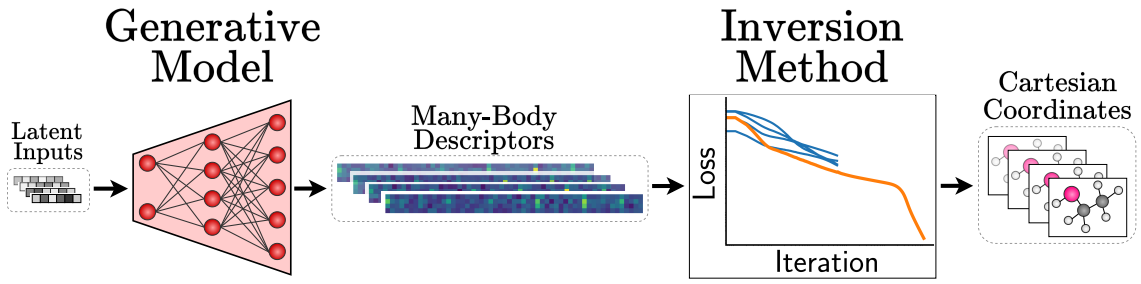

Figure 5.1: Diagram showing the workflow to be used throughout this chapter. The generative model, in this case a GAN, produces molecules in the form of high-dimensional many-body descriptors by randomly sampling from a latent space. These many-body descriptors are then transformed into human-interpretable atomic configurations in Cartesian coordinates using the developed local many-body representation inversion method.

density-functional theory (DFT), see Section 3.3. There are ten separate molecule types included, ranging from ethanol with nine atoms to azobenzene with 24. A full list of all the ten molecule types, and additional relevant information, is shown in Table 4.1. The revised MD-17 dataset randomly selects 100,000 configurations from the original MD-17 dataset for each molecule type. The DFT energies and forces for all configurations are calculated using the PBE functional [198] and the def2-SVP basis set [278], combined with strict convergence criteria and dense integration grids [87].

The type of generative model that is used in all sections of this chapter is a Generative Adversarial Network (GAN) [77], see Section 2.4.2 for more details. A GAN consists of two neural networks which compete against each other. The generator G takes random noise latent vectors $\boldsymbol{z} \sim p_z$ as input and outputs synthetic data $\boldsymbol{x}_g \sim p_g$. The discriminator D takes either real data $\boldsymbol{x} \sim p_{\text{data}}$ or the generated data $\boldsymbol{x}_g = G(\boldsymbol{z})$ and must distinguish between the two. The weights of G and D are updated so as to minimize the distance between the real data distribution p_{data} and the generated data distribution p_g. The Jensen-Shannon divergence [149] was originally used [77] to measure the distance between these two distributions, but it was found to contain sub-optimal properties leading to vanishing gradients. In the now ubiquitous Wasserstein

GAN (WGAN) formulation [27], the Wasserstein-1 distance is used to measure the distance between p_{data} and p_g as $W(p_{\text{data}}, p_g)$. This distance $W(p_{\text{data}}, p_g)$ is continuous everywhere and differentiable almost everywhere [27], a feature that reduces the risk of vanishing gradients and mode collapse. More formally, a WGAN plays a zero-sum game [146] with a Nash equilibrium [293, 294] found using the following minimax objective:

$$\min_{G} \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\text{data}}} [D(\boldsymbol{x})] - \mathop{\mathbb{E}}_{\boldsymbol{x}_g \sim p_g} [D(\boldsymbol{x}_g)] , \tag{5.1}$$

which is constructed using the Kantorovich-Rubinstein duality [151] where $\mathcal{D}$ is the set of 1-Lipschitz functions, which are strictly continuous functions with an absolute value for the maximum slope of one [152]. Throughout the training of a WGAN, the discriminator D must remain a 1-Lipschitz function. Two different methods can be used to enforce this, (1) clipping all weights of D to within $[-c, c]$ where c is some small constant $c \approx 0.01$, and (2) where the gradients of D are penalized if their norm deviates from one, as described in Equation (2.25). This second method of a gradient penalty is used in this work due it improving the quality of the generated samples.

There are two main components used in this workflow, as can be seen in Figure 5.1: the generative model and the many-body inversion method. Both contribute to the total error of the workflow in their own way. The error inherent in the inversion method can be estimated from the results shown in Chapter 4, where real descriptors from the revised MD-17 [87] training set are inverted, instead of descriptors obtained from a generative model. This effectively isolates the inherent error in the inversion method. Therefore, the results shown in this chapter should be compared and contrasted to those shown in the last chapter in order to see the error caused by the generative modelling part of the workflow.

Details on the various libraries used throughout this chapter are now provided. The WGAN is implemented in PyTorch [295], a widely used deep learning framework known for its flexibility and support GPU acceleration. The models were trained using a Nvidia RTX 3060 Ti GPU, the funding for which was provided by the Irish Research Council. The many-body descriptors of the bispectrum components were calculated using LAMMPS [238, 239]. The Atomic Simulation Environment (ASE) was used for data reading and writing, manipulating molecule trajectories, and for visualisation

[296, 297]. All DFT calculations were performed with PySCF [42–44]. The feature scaling used throughout this chapter is to standardise many-body descriptors and to normalise energies with the scikit-learn package [298], see Section 2.2 for more details. SciPy [299] is used for the kernel density estimation.

The organisation of this chapter is as follows. First presented is some preliminary work on generating configurations using GANs, primary credit for which is due to Dr James Nelson. In Section 5.3, many-body descriptors are implemented as the new chemical representation to be used within the GAN, facilitated by the local inversion method developed in Chapter 4. The architecture of the GAN must be correspondingly changed to accommodate this new representation. This improved model undergoes two further modifications in the succeeding sections. In Section 5.4 multiple different molecule types are included within the training dataset of the GAN. The second modification takes place in Section 5.5, where the GAN is adapted to allow for the selective generation of configurations which have a specified value for some observable property. The final section includes a summary of the presented results.

5.2 Preliminary Work

The idea for my initial PhD project came from preliminary work by a former PhD student Dr James Nelson, who graduated from the Computational Spintronics group here in Trinity College Dublin in 2020. This preliminary work consisted of the creation of a Generative Adversarial Network for the Discovery of Atomic Landscapes and the Formation of new molecules (GANdalf). Primary credit for the work presented here in Section 5.2 is therefore due to Dr Nelson. My contributions to his work include helping to adapt the original GAN into a more sophisticated Wasserstein GAN, and in the inclusion of an additional energy prediction model to allow for conditional generation. A brief overview of the findings of this preliminary work is given here as it is pertinent to the choices that were subsequently made. This preliminary work motivated the direction of this chapter, and inspired the use of more advanced many-body descriptors and the subsequent inversion method as described in Chapter 4.

The objective of the work presented in this section is create a generative model

capable of learning how to output atomic configurations directly in Cartesian coordinates. A Wasserstein GAN, see Section 2.4.2, is used as the generative model. A simple representation based on various interatomic distances between neighbouring atoms, see Figure 5.2, is used as the descriptor for each atom in order to enforce locality. The original, un-revised version of the MD-17 dataset was used [274–276]. This preliminary version of GANdalf is trained on the configurations contained within the molecular dynamics trajectory for a single molecule type such as benzene.

5.2.1 Representation

The choice of which molecular representation is to be used has a critical importance on the achievable accuracy within any machine-learning framework [85]. The most naive way to represent an atomic configuration containing N_{at} atoms is with Cartesian coordinates $\boldsymbol{r} \in \mathbb{R}^{N_{at}\times 3}$. However, directly learning Cartesian coordinates is very inefficient because every translation, rotation, and permutation will alter the descriptors, even though the configuration is essentially unchanged, see Figure 3.1. This means that identical configurations, related by translation or rotation will yield differing descriptors, confusing the model and hindering learning. Cartesian coordinates are therefore said to be an over-complete representation [233], with the same configuration yielding differing descriptors for all translations, rotations, and permutations, see Figure 3.4. The way around this problem is to construct the descriptors using many-body terms which only consider internal distance and angles, invariant to translations and rotations. 2-body terms such as interatomic distances r_{ij} between atoms i and j are calculated as

$$r_{ij} = |\boldsymbol{r}_i - \boldsymbol{r}_j| \,. \tag{5.2}$$

Interatomic angles (triplet angles) θ_{ijk} between atoms i and k, centered on atom j are classified as being 3-body terms and are calculated as

$$\theta_{ijk} = \arccos\left(\frac{(\boldsymbol{r}_i - \boldsymbol{r}_j)\cdot(\boldsymbol{r}_k - \boldsymbol{r}_j)}{r_{ij}\, r_{kj}}\right) . \tag{5.3}$$

In this preliminary version of GANdalf, the descriptors were constructed as a simple concatenation of these interatomic distances and angles. It was found that in-

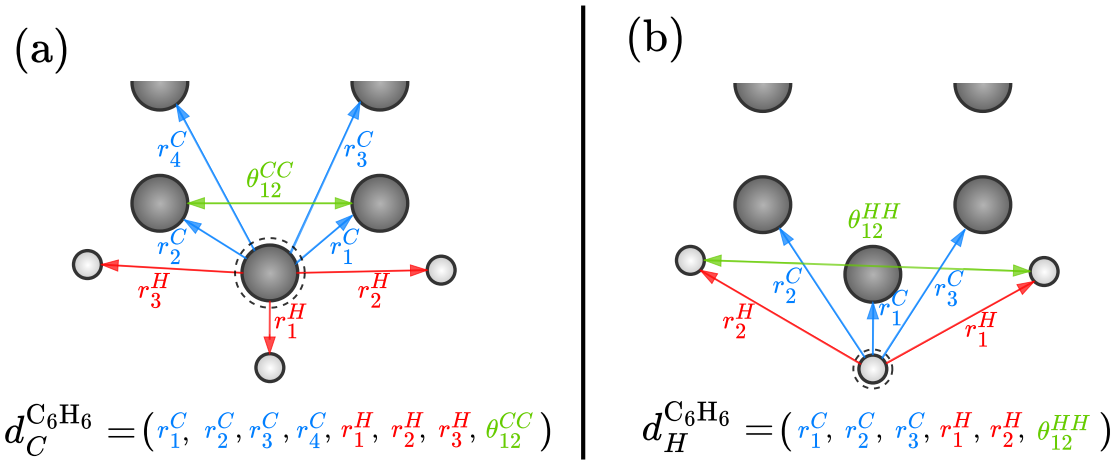

Figure 5.2: Diagram showing the descriptors for benzene (C_6H_6) used in the preliminary version of GANdalf. These simple descriptors are entirely based on the interatomic distances to the nearest neighbouring atoms. Panel (a) centered on a carbon atom and panel (b) centered on a hydrogen atom, both show the relevant interatomic distances used.

cluding information about the chemical species present in the molecule was essential to allowing the GAN to learn. A disadvantage of this method is that the appropriate descriptors must be hand-crafted for each molecule type being considered. For example, the descriptors for each carbon and hydrogen atom in the molecule type of benzene (C_6H_6) are respectively defined as

$$\boldsymbol{d}_C^{C_6H_6} = (r_1^C, r_2^C, r_3^C, r_4^C, r_1^H, r_2^H, r_3^H, \theta_{12}^{CC}), \tag{5.4}$$

$$\boldsymbol{d}_H^{C_6H_6} = (r_1^C, r_2^C, r_3^C, r_1^H, r_2^H, \theta_{12}^{HH}), \tag{5.5}$$

where r_i^σ is the distance to the ith nearest neighbouring atom of species α and $\theta_{ij}^{\alpha\beta}$ is the distance between the ith nearest neighbouring atom of species α and the jth nearest neighbouring of species β. The descriptors used here are local in nature, with all features being based on nearest neighbours.

5.2.2 Architecture

This preliminary version of GANdalf was trained using the architecture as shown in Figure 5.3. It is a Wasserstein GAN [27, 153] like all the GANs in this chapter, with the default hyperparameters taken from literature [27, 134, 153]. The generator G is a fully connected neural network which takes normally distributed latent vectors $\boldsymbol{z}$ as an input and outputs configurations containing N_{at} atoms in Cartesian coordinates $\boldsymbol{r} \in \mathbb{R}^{N_{at}\times 3}$. In order to enforce translational invariance, the center of mass $\boldsymbol{m} = \frac{1}{N_{at}} \sum_{i=1}^{N_{at}} \boldsymbol{r}^{(i)}$ is subtracted from each atom $\boldsymbol{r}^{(i)}$ in a generated configuration. Both the real and the generated atomic configurations are then converted into descriptors $\boldsymbol{d}$ as described in Equation (5.4) and Equation (5.5) for the case of benzene.

The architecture of the discriminator D was inspired by a Behler-Parrinello neural network (BPNN) [83, 232] as described in Section 3.5.1. The reasoning for this architecture choice was that the descriptor for each atom i only considered its nearby local environment, without having a full global view. Using a BPNN-style model will then allow it to be adaptable to different systems, which contain similar chemistries but different structures. In this BPNN paradigm, a given target property y, is calculated as a sum of atomic contributions y_i as follows:

$$\begin{aligned} y &= \sum_{i=1}^{N_{at}} y_i \\ &= \sum_{i=1}^{N_{at}} f_{Z_i}(\boldsymbol{d}_i)\,, \end{aligned} \tag{5.6}$$

where f_{Z_i} is neural network which takes the local descriptor $\boldsymbol{d}_i$ as input for every chemical species Z_i. There is a separate neural network f_{Z_i} for all of the N_k chemical species present. For example, in the case of benzene (C_6H_6), there are two neural networks: f_C and f_H. Each one is called six separate times, once for each of the six atoms of the corresponding species.

The specifics of the model architecture and training are now detailed. Both G and D were trained using mini-batch stochastic gradient descent [110] with a batch size of 128. The rmsprop optimizer [107] was also used in both models, with a learning rate of 5×10^{-5}. The random noise latent vectors $\boldsymbol{z}$ were drawn from a normally distributed latent space of dimension 100. The non-linear activation function ReLU [119] was used

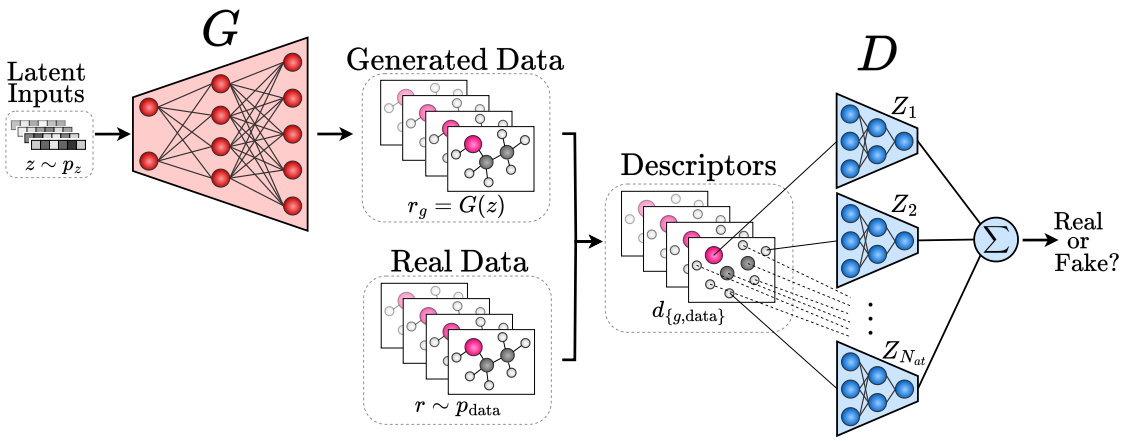

Figure 5.3: Diagram showing the architecture of the preliminary version of the GAN model used to generate conformations for a single molecule type. The generator G takes random noise $\boldsymbol{z}$ from the latent space p_z as an input and outputs a molecule $\boldsymbol{r}_g$ in Cartesian coordinates. Both the real molecules $\boldsymbol{r}$ that are drawn from the training set distribution p_{data}, and the generated molecules $\boldsymbol{r}_g$ are converted into descriptors $\boldsymbol{d}_{\{g,\text{data}\}}$ based on interatomic distances between the nearest neighbours. The species-wise discriminator D takes as inputs these descriptors $\boldsymbol{d}$ and outputs a score quantifying its belief in whether the inputs are real or generated.

in G, while LeakyReLU [125] was used in D with a negative slope of 0.1, see Section 2.2 for more details. The original, un-revised version of the MD-17 dataset [274–276] was used, which contains a total of $627,233$ configurations for the molecule type of benzene. The benzene GANdalf model which is analysed in Figure 5.4, was trained for a total of 1000 epochs on this full dataset. There are a total of four layers in G, with $\{100, N_G, N_G, N_G, N_{at}\times 3\}$ number of neurons in each corresponding layer. A BPNN-style model is used for D with N_k separate models, one for each chemical species present. The first layer of each species-wise model takes the relevant descriptor as input, with the number of input neurons matching the number of features in the descriptor vector. In the case of benzene, there are eight features for carbon and six features for hydrogen. The shape of the carbon model is $\{8, N_D, N_D, N_D, 1\}$ and the hydrogen model shape is $\{6, N_D, N_D, N_D, 1\}$. The size of the models are controlled with the hyperparameters N_G and N_D with a default of 70 used being for both.

5.2.3 Results

The validity of the atomic configurations outputted from the generator of this preliminary version of GANdalf is analysed here. The partial pair distribution functions are examined. These measure the number occurrences of each interatomic distance value as a histogram, for each combination of species. It is hypothesised that, if the pair distribution functions of real and generated configurations closely align, then GANdalf has successfully learned the underlying distribution of the training data p_{data}. Due to the fact that these preliminary descriptors need to be hand-crafted for each molecule type, only benzene is considered here.

The partial pair distribution functions for benzene are shown in Figure 5.4. It can be seen that all of the generated configurations contain realistic interatomic distances; the generated distributions are fully bounded by the real distributions. However, there exists an obvious bias, with the mean values of all the generated distributions, i.e., the positions of the peaks, being consistently underestimated. The variance of the generated distributions, the distribution widths, is also underestimated compared to those of the real distributions. The generated configurations of benzene look reasonable when viewed as a 3D molecule [266, 296, 297], with no atoms out of place and all chemical bonds looking correct. They may look reasonable but in reality, all interatomic distances are slightly smaller than their actual ground-state value. Even a small change in the position of the atoms can have a drastic effect on the configuration's electronic structure [42, 50]. Therefore, these generated configurations are not of any real use due to their unphysical interatomic distances, and can be effectively discarded. However, this preliminary method is still capable of generating configurations in 3D, with some level of accuracy, and is therefore considered worthy of further investigation.

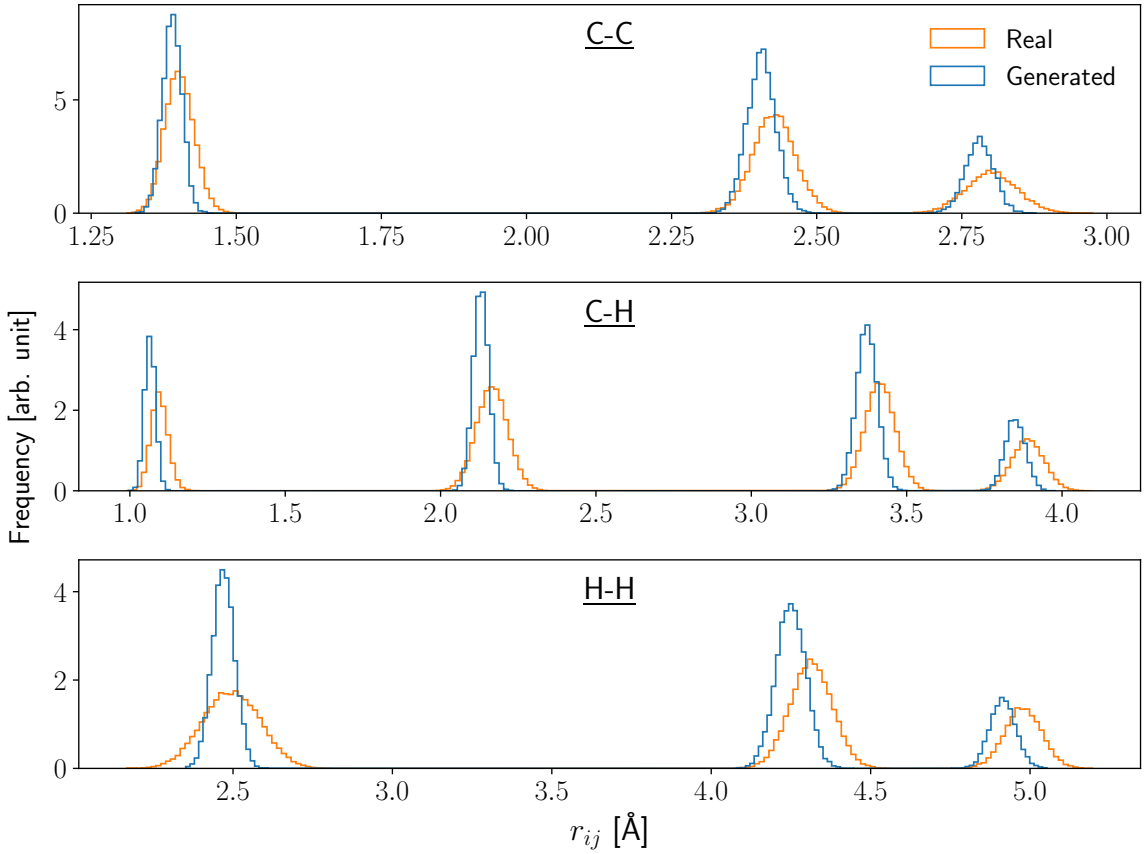

Figure 5.4: A plot of the partial pair distribution functions for atomic configurations of benzene. The partial pair distribution function measures the distances r_{ij} between atoms i and j over all possible combinations of chemical species. The real configurations are those taken from the original MD-17 training dataset [274–276]. The generated configurations are those outputted from the preliminary version of GANdalf as shown in Figure 5.3. The results are considered promising but the generated configurations are far from indistinguishable from those in the training set.

5.3 Single Molecule Type

The preliminary version of GANdalf, from the previous section, has demonstrated sufficient accuracy to warrant further investigation, but it has also exhibited a tendency to output biased atomic configurations. This section seeks to improve upon these preliminary results and to develop an improved version of GANdalf[1] which will not output biased configurations. Numerous different modifications were made, combined, removed, and adapted, throughout a relatively long development process, in order to find a reliable architecture. The architecture developed in this section forms the foundation from which further major adaptions can be made, as detailed in Section 5.4 and Section 5.5. The title of this section refers to the fact that this developed architecture is only suitable for use with a training set consisting of different configurations of the same molecule type. For example, a set of atomic configurations of benzene can be used to train a GANdalf model, but an entirely new model will need to be trained on any other molecule type such as toluene or aspirin.

5.3.1 Representation

In order to improve the preliminary version of GANdalf so as to be able to generate atomic configurations that are actually realistic, the most obvious thing to change is the representation. The preliminary descriptors used, see Figure 5.2, were constructed entirely using 2-body terms that only considered interatomic distances. It was hypothesised that, using more sophisticated many-body representations [75, 207, 208, 230–232], which also consider 3-body interactions between triplets of atoms, would increase the generation abilities of the model. Both the preliminary descriptors and the proposed many-body descriptors, are local in nature, which means that the descriptor (i.e., the feature vector) for each atom i is constructed entirely based on its nearby environment. In the preliminary descriptors, the nearest neighbours are explicitly defined, which means they must be hand-crafted for each molecule type such as ethanol, benzene, or aspirin. Whereas, in many-body descriptors, locality is enforced by each atom i only considering the neighbouring atoms within some cutoff distance r_c. They use a cosine cutoff function f_c that smoothly goes to zero at r_c. This

[1] GANdalf the White ...

has the effect of giving the closer neighbouring atoms a higher relative importance, while also not introducing any discontinuities that may hinder learning. Bispectrum components were selected for use as the many-body representation due to their high degree of accuracy [85] and their ease of computation in the open-source package LAMMPS [238, 239]. This also provides the relevant descriptor derivatives.

Changing GANdalf to incorporate many-body descriptors promised to increase the accuracy of the model, but it also introduced some challenges. It was first attempted to use the same architecture as that of previous preliminary work, see Figure 5.3, where the generator G outputs molecules in Cartesian coordinates, then the descriptors are calculated, before finally being inputted into the discriminator D. This has the major benefit that it is possible for G to generate a large number of configurations, that are already in the human-interpretable format of Cartesian coordinates. After training has concluded, the inference stage begins, where only G, and not D, is of importance. Training G to directly output in Cartesian coordinates makes this inference stage very computationally efficient, with no need for the previously proposed local inversion method.

Training G in the ideal scenario of directly outputting in Cartesian coordinates also introduces a major drawback, which is that during the training stage, the descriptors of each generated batch must be calculated at every training step. In the case of the very simple preliminary descriptors, this additional computational expense is trivial. Many-body descriptors, however, require a non-trivial computational expense to calculate. This involves calling the calculator used to determine the many-body descriptors (e.g., LAMMPS) at each training step. It was determined that this was too much of an additional computational expense, and that the more promising line of research was to develop an inversion method to find the Cartesian coordinates that correspond to the generated many-body descriptors. When this decision was made, the collaboration with Matteo Cobelli on the inversion method had already begun. However, a large number of calls to the many-body calculator LAMMPS must also be made in the inversion method, as it works by iteratively updating some initial configuration until its many-body descriptors match those of the target descriptor which is to be inverted. This shifts the computational expense of calling the many-body calculator from the training stage to the inference stage.

It is noted that despite the efficacy of the local many-body inversion method, it does slow down the inference stage. Therefore, the preliminary method of directly generating configurations in Cartesian coordinates underwent further investigation, but this time using a many-body representation in the discriminator D. This is in spite of the aforementioned additional computational expense of calling the many-body calculator at each training step. The benefit of being able to directly generate in Cartesian coordinates was deemed too great to ignore. However, in order to properly train a GAN, the gradients of the loss function from Equation (2.25) are backpropagated [101] from the output layer of D, all the way back through to the input of G. An external package such as LAMMPS, being in between G and D, acts as a non-differentiable layer through which gradients cannot flow. However, if the derivatives of the descriptors with respect to the Cartesian coordinates are known, then it should be possible to manually define the gradients to be used for backpropagation. This was briefly tried but was not successfully achieved due to problems with the feature scaling of the descriptor derivatives.

5.3.2 Architecture

Once it was decided to use a generator G that outputs many-body descriptors, facilitated by the newly-developed local inversion method, the next major design choice was the architecture of the discriminator D. In the preliminary version of GANdalf, the design of D was inspired by a Behler-Parrinello neural network (BPNN) [83, 236], see Section 3.5.1 and Figure 5.3, which has been widely used for prediction of properties of molecules and solids to great success [300–302]. Initial attempts at an updated version of GANdalf that uses both many-body descriptors and a BPNN-style discriminator were unsuccessful. In order to diagnose why, a simple test case was studied: a dataset containing random distortions of the hydrogen molecule (H_2). Being a diatomic molecule, the local environment for each atom is the exact same, resulting in two identical descriptors for each configuration. Training this proposed version of GANdalf on this hydrogen dataset also produced unusable outputs, but allowed for an easy diagnosis of the problem. It was found that G would generate realistic descriptors for each of the two atoms, but there were not identical as they should be. Since the BPNN-style D quantifies the realism of each atom's descriptor independently, with

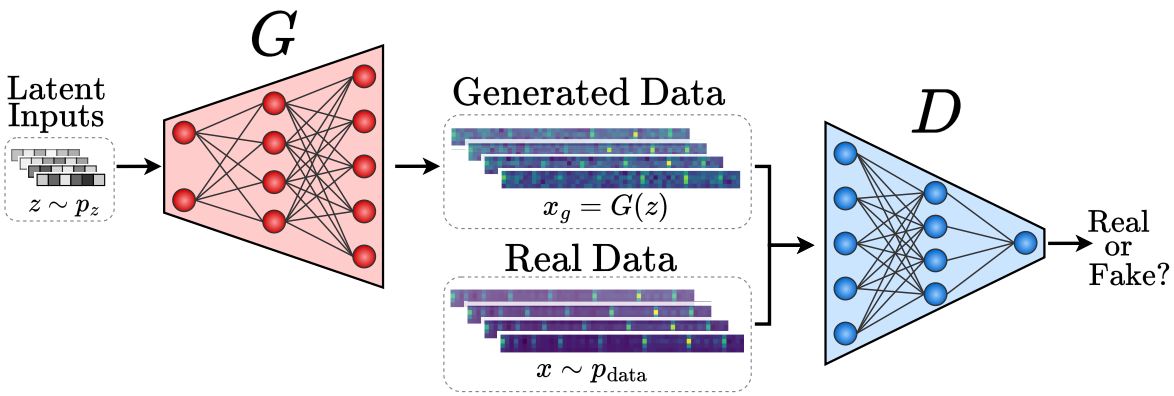

Figure 5.5: Diagram showing the architecture of the GAN model used to generate conformations for a single molecule type. The generator G takes random noise $\boldsymbol{z}$ from the latent space p_z as an input and outputs many-body descriptors $\boldsymbol{x}_g$. The input of the discriminator D alternates between real descriptors $\boldsymbol{x}$ and generated descriptors $\boldsymbol{x}_g$, where $\boldsymbol{x}$ is drawn from the real data distribution p_{data}. The score outputted by D signifies its belief in whether the inputted descriptors are real or generated.

no global view, it cannot distinguish between generated molecules with non-identical descriptors and real molecules which do have identical descriptors. It is thought that the reason that the BPNN-style D worked successfully in the preliminary version of GANdalf is due to the fact that the descriptors were calculated at each training step, enforcing the atoms of the generated molecules to be appropriately correlated. This motivated the change from using a BPNN-style D to a fully connected D, which utilises a global view of the molecule when quantifying its realism score.

The exact methodology used for training this version of GANdalf which generates many-body descriptors for a single molecule type, is now detailed. Bispectrum components [75, 238, 239] were used as the many-body representation, with the number of features N_f determined by the value of $2j_{\text{max}}$, where $2j_{\text{max}} = 8$ is used to yield $N_f = 55$. All the parameters used by the local many-body inversion method are kept the same as those from Section 4.4. The architecture of the GAN employed is a Wasserstein GAN [27, 153], which has become effectively ubiquitous, with no real scenarios where the original GAN formulation is superior.

The generator G is a fully connected neural network which takes latent vectors $\boldsymbol{z}$ as input and outputs molecules $\boldsymbol{x}_g$ expressed in many-body descriptors, namely the bispectrum components. Random noise latent vectors $\boldsymbol{z}$ were sampled from the latent space $\boldsymbol{z} \in \mathbb{R}^d$, which is normally distributed $\boldsymbol{z} \sim \mathcal{N}_d(0,1)$ with the dimension d defaulting to 100. The generator G contains five fully connected layers with $\{d, N_G, 2N_G, 4N_G, N_k{\times}N_f\}$ neurons in each corresponding layer, where $N_G = 60$ is a hyperparameter controlling the number of neurons, N_k is the number of chemical species present and N_f is the number of features. The ReLU activation function [119] from Equation (2.12) is used in all layers except the last, which is left linear with no output activation function.

The discriminator D is a fully connected neural network that takes many-body descriptors as inputs, alternating between generated descriptors $\boldsymbol{x}_g$ and real descriptors $\boldsymbol{x}$, both in bispectrum components. The model outputs a score quantifying its belief in whether the input is real or generated. It contains five fully connected layers with $\{N_k{\times}N_f, 4N_D, 2N_D, N_D, 1\}$ neurons in each corresponding layer, where $N_D{=}65$. The LeakyReLU activation [125] from Equation (2.13), is used with a negative slope set to 0.2. Both gradient clipping [27] and gradient penalty versions of a WGAN are implemented [153], see Section 2.4, with the gradient penalty version being chosen for the superior quality of its outputs. However, it is noted that the training time for a gradient penalty WGAN is much longer than that of one with gradient clipping. As is standard procedure in WGANs, for every time the weights of G are updated, the weights of D are updated five times.

Both models were trained using mini-batch stochastic gradient descent [110] with a mini-batch size of 128. The learning rates for the rmsprop gradient descent optimizers [107] in G and D were both set to $5{\times}10^{-5}$. The size of the models G and D are controlled by the hyperparameters N_G and N_D, respectively. These hyperparameters are set such that D is $\sim 5-10\,\%$ larger than G with the rationale being than D is the model driving the learning as it is the only one which sees the training data. However, the capacity of G and D must be balanced, otherwise mode collapse may occur, where the model with the larger capacity learns to always trick the other, causing gradients to vanish and learning to halt. The risk of mode collapse occurring can be minimised by using a WGAN architecture [27, 153], matching capacities in both models, and by

carefully choosing the values of the hyperparameters.

A total of 100,000 configurations are contained within the revised MD-17 dataset [87] for the ten molecules types[2]. Each of these ten were randomly split into three separate sets, the training set with 85% of the data, the test set with 10%, and the validation set with 5%. The test set and the validation set are typically used for overfitting prevention and for hyperparameter tuning, respectively. Within the paradigm of generative modelling, the problem of overfitting is not of primary importance as in the usual regression or classification tasks. Default hyperparameters from the literature are used [27,134,153]. Ordinarily a systematic hyperparameter search is performed, using either grid search [234] or Bayesian optimisation [261]. This step was not performed due to the relatively high expense of retraining a GAN from scratch multiple times, and then applying the inversion method to a large number of configurations. Additionally, no one performance metric exists to choose the best model. One could have been tailor made based on comparing the real and the generated pair and angular distribution functions, but this was left undone due to the acceptable level of performance already achieved with the default hyperparameters.

5.3.3 Results

This section deals with a version of GANdalf that is trained on many-body descriptors of atomic configurations from a single molecular dynamics run. It is now desired to analyse the realism of the generated atomic configurations by this version of GANdalf. The section title of 'Single Molecule Type' refers to the fact that a different GAN must be trained for each different molecule type (e.g., ethanol, benzene, aspirin, etc) This yields a total of 10 different GANdalfs to be analysed, one for each molecule type contained in the revised MD-17 dataset [87], see Table 4.1. The workflow to be used, in all sections of this chapter, for generating atomic configurations is shown in Figure 5.1. The generator G takes random noise latent vectors as input, and outputs molecules expressed in many-body descriptors $\boldsymbol{x}_g \in \mathbb{R}^{N_k \times N_f}$, where N_k is the number of chemical species and N_f is the number of features. The newly-developed local many-body inversion method from Chapter 4, is then used to calculate the

[2]Nine of the ten molecule datasets have $100,000$ configurations, with azobenzene having only 99988 due to 11 failed DFT calculations, and the original MD-17 dataset [274–276] only containing 99999 configurations.

appropriate atomic configuration $\boldsymbol{r}_g \in \mathbb{R}^{N_{at}\times 3}$ in Cartesian coordinates, where N_{at} is the number of atoms contained in the molecule type.

In order to verify the validity of the generated atomic configurations $\boldsymbol{r}_g$, as compared to the real atomic configurations $\boldsymbol{r}$ from the training set, the pair and the angular distribution functions are analysed. The pair distribution function measures the frequency of interatomic distances r_{ij} between all atoms i and j within a configuration. The angular distribution function similarly quantifies the frequency of angles θ_{ijk} for all possible atom triplets between atoms i, j, and k. Each of these two functions can also be expressed as the partial pair or partial angular distribution functions, where different combinations of chemical species are displayed individually. For example, the partial pair distribution function for benzene (C_6H_6), would be separated out into C-C, C-H, and H-H. The angular distribution function for benzene can be separated out various different ways (such as C-C-C, H-H-H, C-H-C, H-H-C, etc) but in this work all angles centered on a given species are summed over, irrespective of the species of the other two atoms in the triplet (e.g., X-C-X, X-H-X). For more details on the pair and angular distribution functions, see Section 4.4.1.

The pair distribution functions for all ten molecule types are shown in Figure 5.6. It can be clearly seen that the generated atomic configurations have internal bond distances that closely align to those contained within the real configurations. This is especially evident when compared to the results obtained by using the preliminary version of GANdalf shown in Figure 5.4. There do exist minor differences between the real and the generated distributions, such as in the case of benzene. These discrepancies are thought to mainly arise from the inherent problems in the many-body inversion method. However, there is still some contribution to the error of the method from the GAN. The final loss values obtained in the inversion method, from Equation (4.1), tend to be higher for the generated descriptors $\boldsymbol{x}_g$ than for real descriptors $\boldsymbol{x}$ from the training set. This implies that there is some inherent difference between the generated $\boldsymbol{x}_g$ and the real $\boldsymbol{x}$ that the pair and angular distribution functions cannot identify.

The angular distribution functions for both the real and the generated configurations are shown in Figure 5.7. This figure shows that the internal angles between any arbitrary triplet of atoms are very similar for real configurations and for gener-

ated configurations. This similarity holds for all ten molecule types included in the revised MD-17 dataset [87]. The minor deviations evident are no more than those shown in Figure 4.4, which represents the minimum error possible in the inversion method, with no generative model being used. As detailed in Section 4.3, the inversion method struggles for systems with many torsional degrees of freedom. Therefore, the partial pair and partial angular distribution functions are analysed in Figure 5.8 for the case of malonaldehyde, which consists of a chain of carbon and oxygen atoms, containing many possible torsional angles. It can be seen in this figure that the triplet angles are very realistic in the generated configurations. The interatomic distances do exhibit some differences in species combinations which include carbon, in particular with the peak height of the distributions. However, the peak locations and widths are all accurate, with the relatively minor additional variance still being at an acceptable level of deviation from the real configurations. It can therefore be concluded that GANdalf is capable of generating descriptors that, when inverted, are similar to those configurations included in the training dataset.

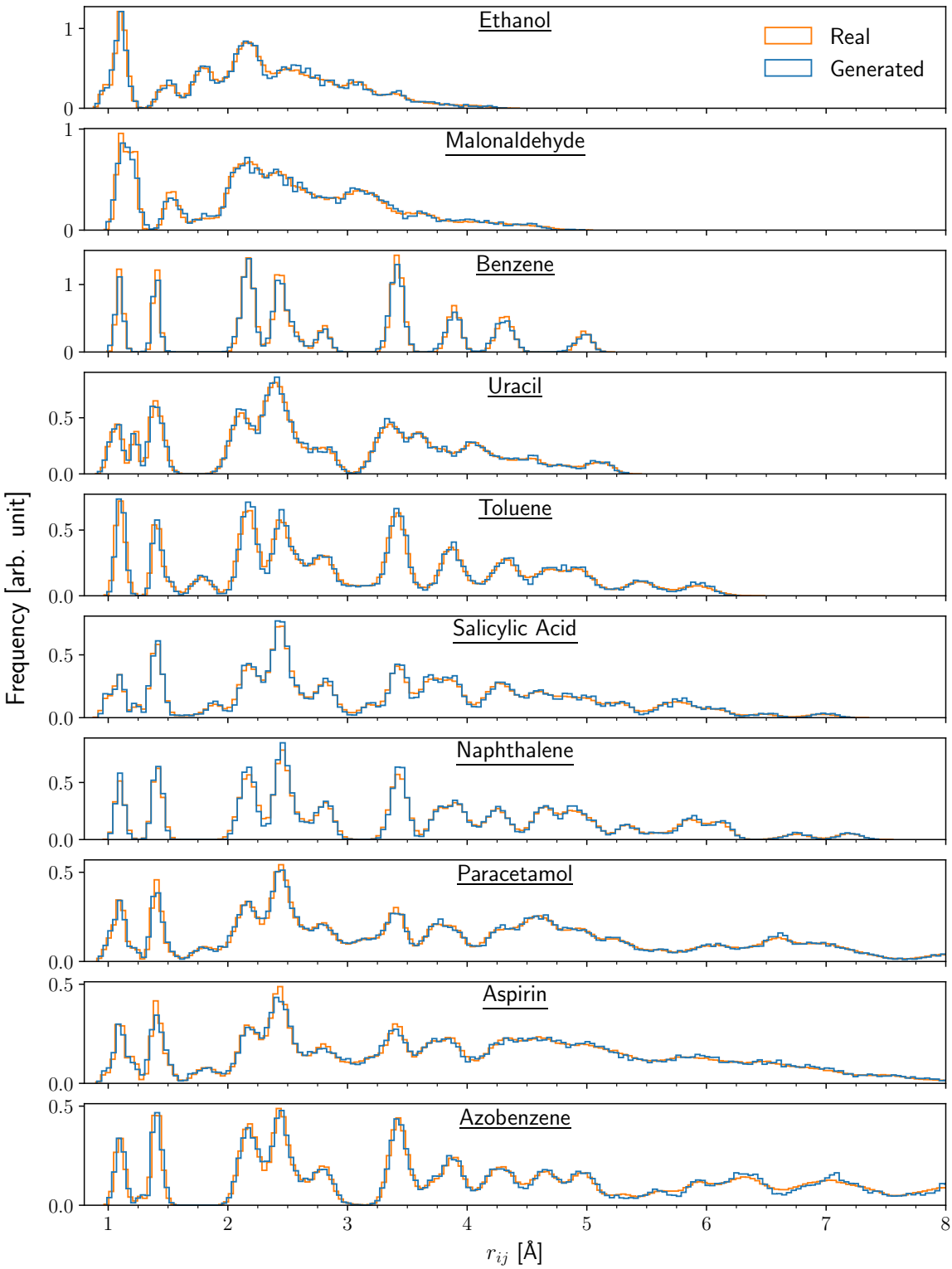

Figure 5.6: Plot of the total pair distribution functions for all ten molecules types included in the revised MD-17 dataset [87]. This function considers every possible distance between atoms i and j in a configuration. The real configurations are taken directly from this dataset. The generated configurations are the result of the local inversion method being applied to the many-body descriptors generated by GANdalf. The real distribution contains $10,000$ samples, while the generated one contains 200.

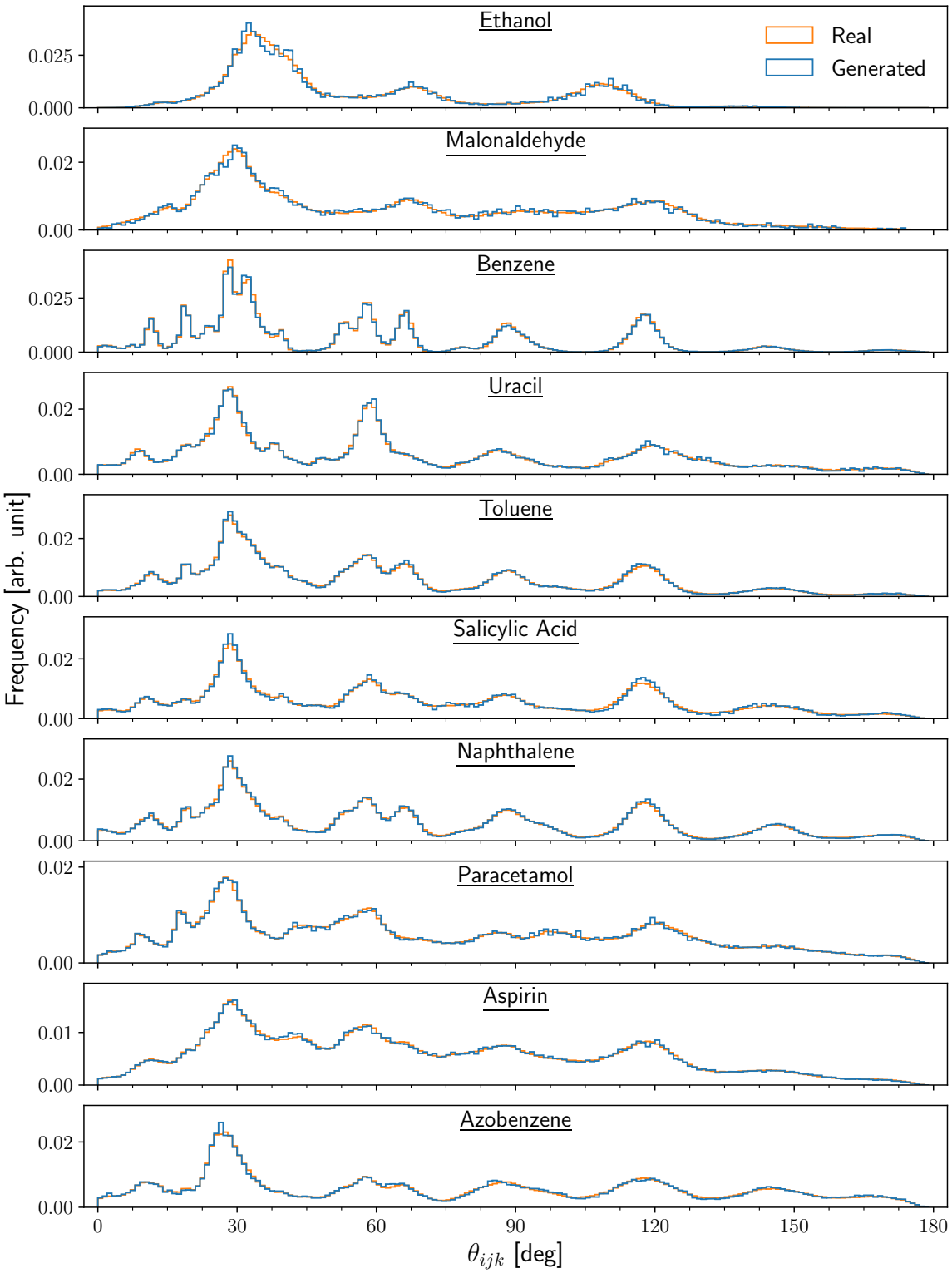

Figure 5.7: Plot of the total angular distribution functions for all ten molecules types included in the revised MD-17 dataset [87]. This function is over all possible triplet angles θ_{ijk} between atoms i, j, and k. The real configurations are taken directly from this dataset. The generated configurations are the result of the local inversion method being applied to the many-body descriptors generated by GANdalf. The real distribution contains $10,000$ samples, while the generated one contains 200.

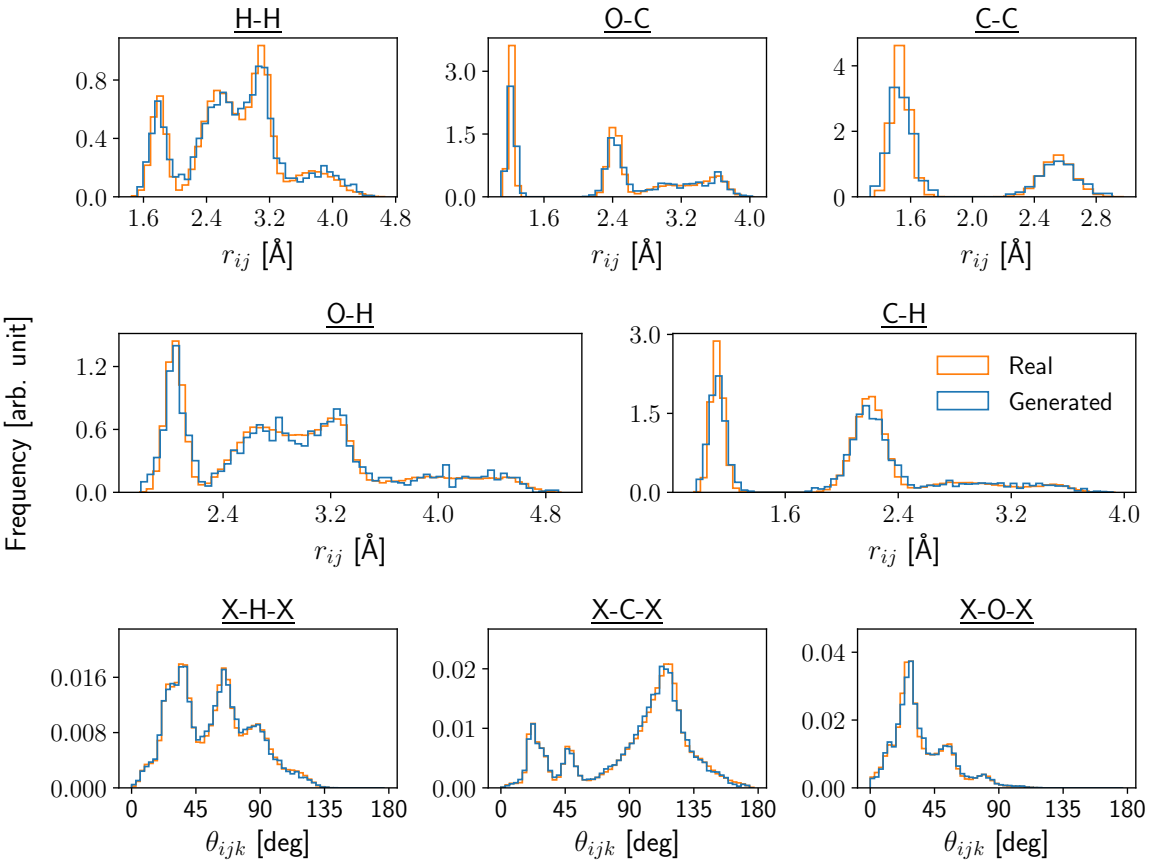

Figure 5.8: Plot of the partial pair and partial angular distributions functions for malonaldehyde configurations from the revised MD-17 dataset [87]. The partial pair distribution function measures the distances r_{ij} between atoms i and j over all possible combinations of chemical species. The partial angular distribution function considers every triplet angle θ_{ijk} between atoms i, j, and k centered on all three species present in malonaldehyde. The real configurations are taken directly from this MD-17 dataset. The generated configurations are the result of the local inversion method being applied to the many-body descriptors generated by GANdalf. The real distribution contains $10,000$ samples, while the generated one contains 200.

5.4 Multiple Molecules Types

After it was shown that this many-body descriptor version of GANdalf could successfully reproduce conformations of a single molecule type, the next step taken was to include multiple different molecule types in the training set. In order to do this, a conditional GAN, see Section 2.4.3, was used, where an additional input $\boldsymbol{l}$, denoting the molecule type, is integrated into both the generator G and the discriminator D. This additional input is required so that the generated molecule descriptors $\boldsymbol{x}_g$ can undergo the local many-body inversion method to obtain configurations in Cartesian coordinates $\boldsymbol{r}$. If each generated configuration was not labelled with a specific molecule type label $\boldsymbol{l}$, then the inversion method would not know the initial configuration $\boldsymbol{r}^{(0)}$ with which to begin the optimisation procedure. The ultimate goal for training on multiple molecule types in this fashion is to have the ability to generate new molecule types. However, this is infeasible due to the inversion method requiring an initial configuration that is reasonably close to the target configuration to be inverted. This inability to generate new molecule types means that the benefit of training on multiple molecule types is that it is efficient; one model is capable of performing the job of many models. It also serves to demonstrate the versatility of the GANdalf framework.

5.4.1 Label Embedding

A method to encode categorical information such as molecule type into a vector $\boldsymbol{l}$ is now required. Two ways to do this are considered: (1) one-hot encoding and (2) embedding layers [179]. Both of these methods are borrowed from the field of natural language processing, where words in a text must be represented as numbers. The one-hot encoded molecule type vectors for a training dataset containing ethanol, malonaldehyde, and aspirin may be constructed as $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$ respectively. One-hot encoding is the simplest method, but is only suitable for cases when the total number of words (or molecule types) is small as it is a sparse representation. Another disadvantage of this method is that each vector is linearly independent so no notion of similarity is introduced.

The second possible method used to represent molecule types as vectors are embedding layers [179] which seek to represent each word as a point in a vector space

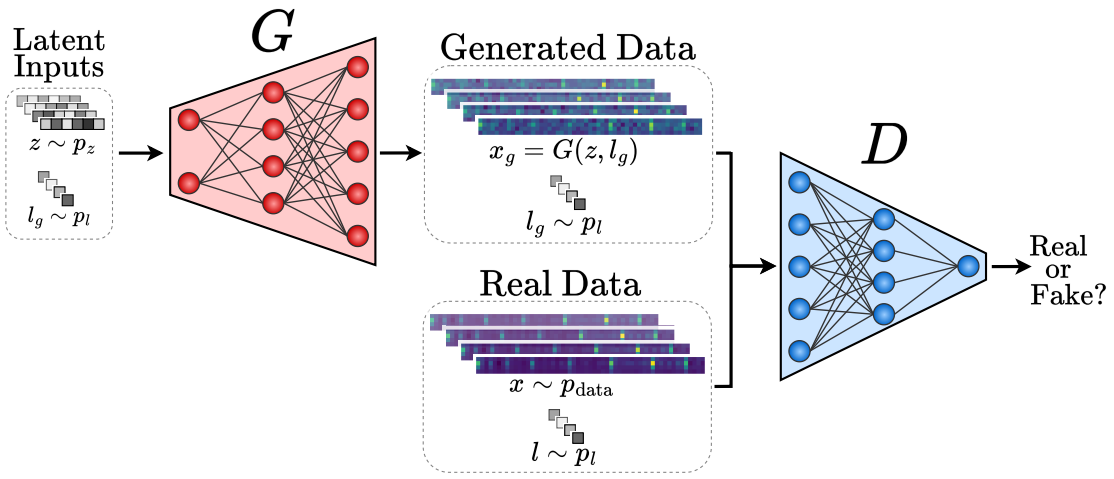

Figure 5.9: Diagram showing the architecture of the GAN model used to generate conformations for multiple different molecule types. The generator G is inputted random noise $\boldsymbol{z}$ from the latent space p_z and molecule type labels $\boldsymbol{l}_g$ from the set of all possible labels p_l. The outputs of G are many-body descriptors $\boldsymbol{x}_g$. The inputs of the discriminator D alternate between real pairs $(\boldsymbol{x}, \boldsymbol{l})$ and generated pairs $(\boldsymbol{x}_g, \boldsymbol{l}_g)$, where $\boldsymbol{x}$ are real descriptors drawn from p_{data} and $\boldsymbol{l}$ are real labels drawn from p_l. The score outputted by D signifies its belief in whether the inputs are real or generated.

where similar words are grouped closely together. This is achieved by using a set of learnable weights that are updated by backpropagating gradients. A major benefit of this method is that, if properly trained, it learns some notion of the semantic meaning of the labels [303]. This allows vector arithmetic to be performed on the labels projected into this vector space. For example, type vectors $\boldsymbol{l}_X$ for some arbitrary molecule type X could be constructed as follows: $\boldsymbol{l}_{\text{benzene}} - \boldsymbol{l}_{\text{hydrogen}} + \boldsymbol{l}_{\text{methyl}} = \boldsymbol{l}_{\text{toluene}}$.

In both of these methods, the molecule type vector $\boldsymbol{l}$ is retrieved using indices L. The one-hot encoded vectors are constructed by indexing the desired row out of the identity matrix $\mathbb{I}_n$ where n is the number of labels. For example, given a label index $L = 2$ out of three possibilities $\{0, 1, 2\}$, the corresponding row of the identity matrix $\mathbb{I}_3$ would be $[0, 0, 1]$. Embedding layers can be thought of as a simple lookup table, storing the learnable embeddings for each molecule type index. Therefore, the actual input into each model in both cases is just the label index L, not the type vector $\boldsymbol{l}$. This can be seen in Figure 5.9 which shows the architecture of GANdalf for multiple molecule types as described. The constructed type vector $\boldsymbol{l}$ is then simply concatenated onto the original model inputs.

5.4.2 Results

A version of GANdalf that can be trained on a dataset containing multiple different molecule types is now outlined. Consider that GANdalf is trained on descriptors $\boldsymbol{x} \in \mathbb{R}^{N_k \times N_f}$, where N_k is the number of chemical species present and N_f is the number of features. It is therefore straightforward to include multiple molecule types with the same species in the training set even if they have a variable number of atoms. However, since the output layer of the generator G has $N_k * N_f$ nodes, it cannot accommodate molecule types which contain differing species. For example, a molecule type such as ethanol which contains hydrogen, carbon, and oxygen atoms could not be included in the same training dataset as benzene, which only contains hydrogen and carbon. It may be possible to pad the descriptors $\boldsymbol{x}$ with appropriately shaped vectors, full of zeroes for species which are not present in that particular molecule type. This would allow GANdalf to work for both a variable number of atoms and a variable number of species. The zero paddings would then be removed before the local many-body inversion method. However, this possibility has not yet been investigated.

This multiple molecule version of GANdalf is trained in practically the same way as the previous single molecule version. It uses the same hyperparameters as detailed in Section 5.3.2 and the same inversion method parameters as in Section 4.4. The major difference in this proposed architecture is that both G and D now require the molecule type vector $\boldsymbol{l}$ as an additional input. This is simply concatenated onto the end of the original input, which requires the input layer of the fully connected neural network to be expanded to include an extra $\|\boldsymbol{l}\|$ neurons. In the case of G, the original input of latent vector $\boldsymbol{z}$, is concatenated with $\boldsymbol{l}$ to become $\boldsymbol{z} \oplus \boldsymbol{l}$, with a total of $\|\boldsymbol{z}\|+\|\boldsymbol{l}\|$ neurons in the input layer. In D, which takes many-body descriptors $\boldsymbol{x}$ as input, the new multiple molecule input becomes $\boldsymbol{x} \oplus \boldsymbol{l}$, with $(N_k \times N_f)+\|\boldsymbol{l}\|$ neurons in the input layer. Care must be taken with feature scaling, see Section 2.2, in this multiple molecule version of GANdalf. This is achieved by applying feature scaling to the descriptors $\boldsymbol{x}$ of each molecule type separately, only using statistics from that particular molecule type. Since the descriptors $\boldsymbol{x}$ for each molecule type have different ranges, mean values, and standard deviations, it is not appropriate to calculate the feature scaling statistics using all molecule types.

A multiple molecule version of GANdalf was trained on a dataset consisting of

benzene, toluene, and naphthalene. These three molecule types have 12, 15, and 18 atoms, respectively, and only contain hydrogen and carbon atoms. After training has concluded, the inference stage begins where the generated many-body descriptors are inverted back into the human-interpretable Cartesian coordinates, as shown in Figure 5.1. In order to generate descriptors of the desired molecule type, only the vector $\boldsymbol{l}$ must be user defined, the latent vector $\boldsymbol{z}$ will be randomly sampled as usual. The version of GANdalf analysed here was trained using one-hot encoded molecule type vectors instead of embedding vectors. It is expected that using embedding vectors will yield better results, when there are a large number of molecule types included. One-hot encoded vectors are deemed sufficient for the current three molecule types.

To verify that GANdalf has learned the underlying data distribution of the real descriptors, a large number of atomic configurations are generated by the means of the local inversion method. The partial pair and partial angular distribution functions for benzene, toluene, and naphthalene are shown in Figure 5.10, Figure 5.11, and Figure 5.12, respectively. From these plots, it can be clearly seen that GANdalf is fully capable to generate configurations of multiple molecule types, particularly when compared to the preliminary results shown in Figure 5.4. It can therefore be said that the atomic configurations generated by this version of GANdalf have internal interatomic distances and angles which are similar to those of the configurations contained within the training set. Some minor deviations can be noted, particularly in the hydrogen distribution functions. This physically corresponds to the H atoms being too rigid, with not enough variation. This is believed to be a problem with the inversion method. A possible solution, not yet investigated, would be to weight H atoms more heavily than C atoms in calculating the bispectrum components in LAMMPS [75, 238], or in the inversion loss function from Equation (4.1).

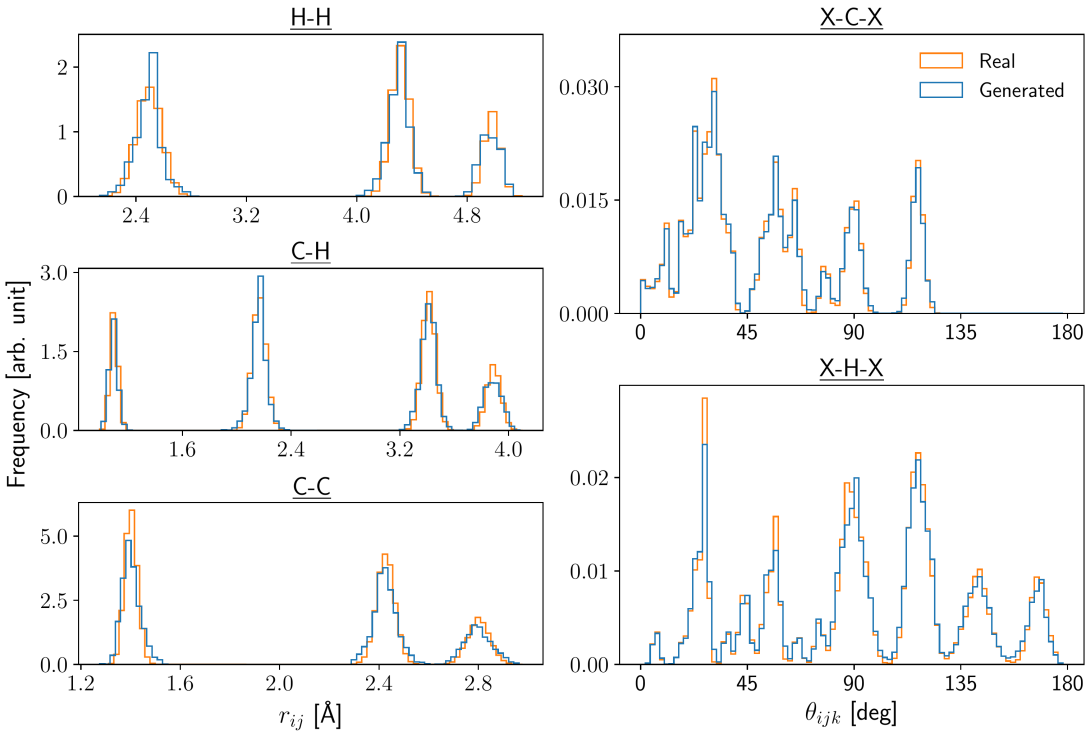

Figure 5.10: Plot of benzene's partial pair and partial angular distributions functions for configurations from the revised MD-17 dataset [87]. The partial pair distribution function measures the distances r_{ij} between atoms i and j over all possible combinations of chemical species. The partial angular distribution function considers every triplet angle θ_{ijk} between atoms i, j, and k centered on all three species present in benzene. The real configurations are taken directly from this MD-17 dataset. The generated configurations are the result of the local inversion method being applied to the many-body descriptors generated by GANdalf. The real distribution contains $10,000$ samples while the inverted one contains 400.

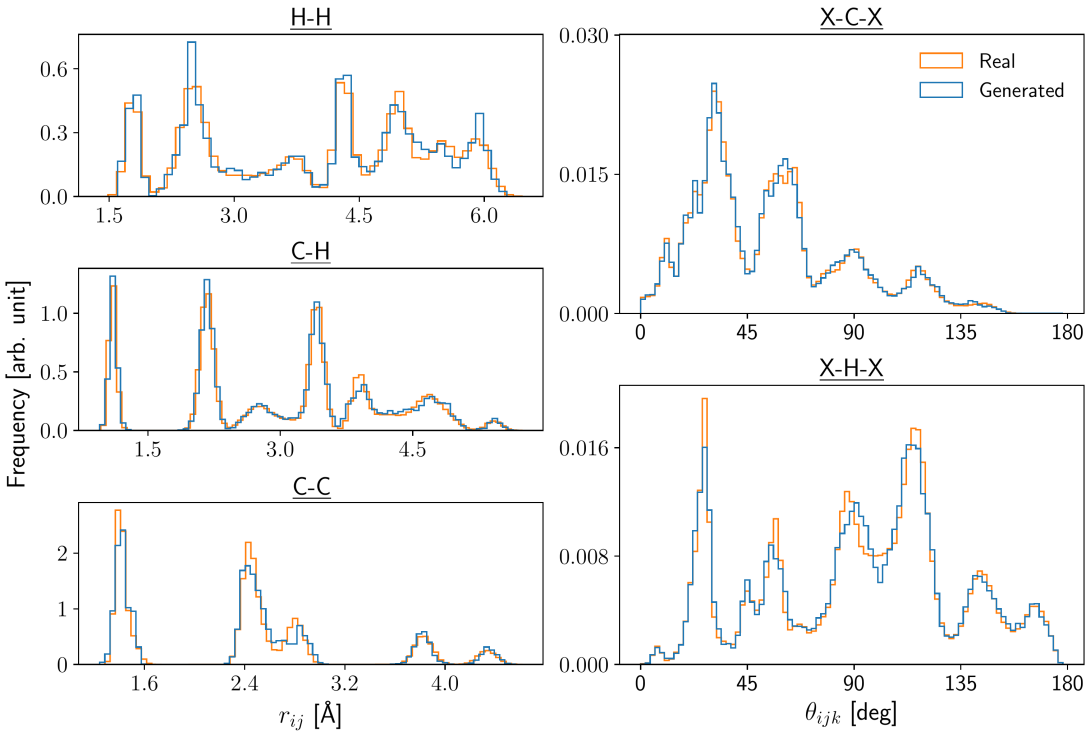

Figure 5.11: Plot of toluene's partial pair and partial angular distributions functions for configurations from the revised MD-17 dataset [87]. The partial pair distribution function measures the distances r_{ij} between atoms i and j over all possible combinations of chemical species. The partial angular distribution function considers every triplet angle θ_{ijk} between atoms i, j, and k centered on all three species present in toluene. The real configurations are taken directly from this MD-17 dataset. The generated configurations are the result of the local inversion method being applied to the many-body descriptors generated by GANdalf. The real distribution contains $10,000$ samples while the inverted one contains 400.

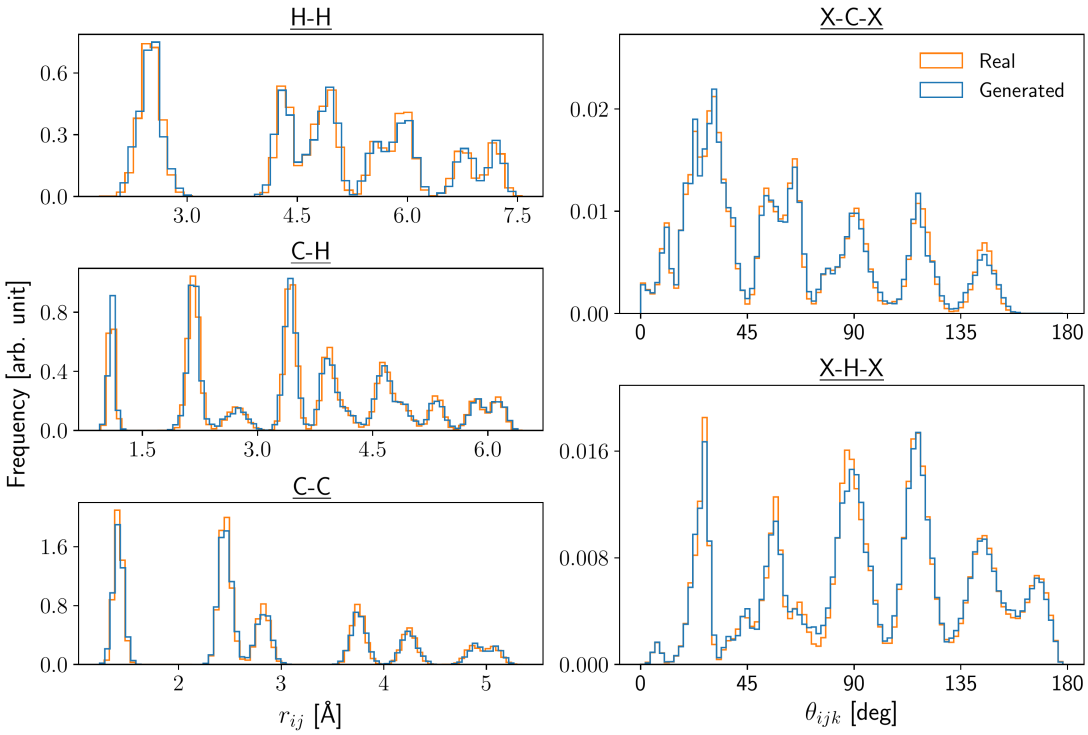

Figure 5.12: Plot of naphthalene's partial pair and partial angular distributions functions for configurations from the revised MD-17 dataset [87]. The partial pair distribution function measures the distances r_{ij} between atoms i and j over all possible combinations of chemical species. The partial angular distribution function considers every triplet angle θ_{ijk} between atoms i, j, and k centered on all three species present in naphthalene. The real configurations are taken directly from this MD-17 dataset. The generated configurations are the result of the local inversion method being applied to the many-body descriptors generated by GANdalf. The real distribution contains $10,000$ samples while the inverted one contains 400.

5.5 Property Conditioning

It would be desirable to have a GAN that is conditioned upon some observable molecular property in order to allow for the selective generation of molecules with certain values of this property. For example, it might be desirable to generate only stable molecules with a low total energy. Additionally, it may be useful to increase the number of configurations occupying certain regions of the property distribution that is difficult to sample using traditional methods [194]. In this approach, both the generator G and the discriminator D would then take this property label as an additional input [218]. A structure-to-property model, see Section 3.5, can be included in the inner training loop in order to force the desired properties to align with the predicted ones. Rigorous methods of calculating property information, such as DFT, would be prohibitively slow for inclusion in the GAN training loop. Therefore, machine learned interatomic potentials or force fields are used. Specifically, a Behler-Parrinello style neural network (BPNN) is used due to its accuracy, large learning capacity, and inexpensive predictions [83,85]. A similar method was successfully used in the past on the Ising model [162,163], allowing for the generation of 2D lattices with desired magnetisation, energy, and temperature values [88]. In theory, BPNN models are suitable for predicting many different quantum-chemistry properties [300–302]. In this case, however, they are only applied to DFT total energy values [50]. This is due to the revised MD-17 dataset [87] only containing energy values and no other molecular properties. A diagram illustrating the architecture of this proposed version of GANdalf is shown in Figure 5.13.

5.5.1 Property Predictor

There exist a number of criteria that a structure-to-property model must satisfy in order to be valid for inclusion in the inner training loop of this property-conditioned version of GANdalf. It needs to take configurations in the form of many-body descriptors as an input and to output predictions for the target property values. The model must also have a sufficiently large learning capacity in order to generalise to the generated descriptors outputted by G. During the initial stages of training, these generated descriptors $\boldsymbol{x}_g$ are extremely noisy, with little resemblance to the real de-

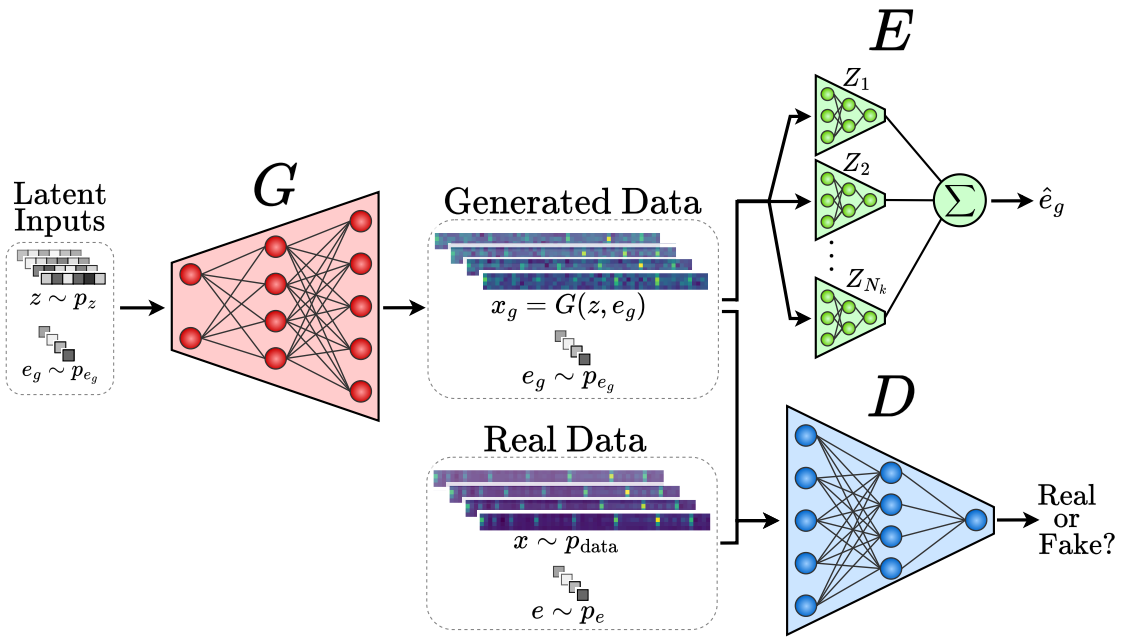

Figure 5.13: Diagram showing the architecture of the GAN model used to generate conformations conditioned on property values. The generator G is inputted random noise $\boldsymbol{z}$ from the latent space p_z and desired property values e_g, drawn from the approximate distribution p_{e_g}. The outputs of G are many-body descriptors x_g. The inputs of the discriminator D alternate between real pairs $(\boldsymbol{x}, e)$ and generated pairs $(\boldsymbol{x}_g, e_g)$, where $\boldsymbol{x}$ are real descriptors drawn from p_{data} and e are real properties drawn from p_e. The score outputted by D signifies its belief in whether the inputs are real or generated. The GAN is conditioned using a property predictor E, consisting of N_k separate sub-models, one for each species Z. It predicts the property values of the generated $\boldsymbol{x}_g$ as $\hat{\boldsymbol{e}}_g$ where G is trained to minimise the difference between the predicted $\hat{\boldsymbol{e}}_g$ and the desired $\boldsymbol{e}_g$.

scriptors $\boldsymbol{x}$. If the learning capacity of the property predictor model is too low, then these initial noisy descriptors will result in predictions that do not allow learning to occur in G. Two different architectures were considered for use as the property predictor model E: a spectral neighbour analysis potential (SNAP) model [84], and a Behler-Parrinello neural network (BPNN) [83, 236].

A SNAP model is a ridge regression model [103] that uses bispectrum components [75] as the input representation, see Section 3.5.2. Ridge regression is just simple linear regression with an additional regularization term that penalises the learnable parameters that grow too large, see Section 2.2. It is an effective widely used model,

particularly in scenarios with few data points. However, since it is only a linear model, with few learnable parameters, it was found to be incapable of being successfully integrated into the GANdalf training loop. The SNAP model was never able to provide reasonable initial predictions for the properties of the noisy generated descriptors, meaning that no learning could occur.

The next candidate architecture for use as the property predictor model E was a BPNN, as detailed in Section 3.5.1. This style of model was first developed for use with a representation of symmetry functions [232], but it was found to work equally well with bispectrum components [75]. This is the model architecture that the discriminator of Section 5.2 was modelled after. The architecture of a traditional BPNN was modified somewhat for use in this work. The local many-body inversion method only requires descriptors $\boldsymbol{x} \in \mathbb{R}^{N_k \times N_f}$ instead of the typical $\boldsymbol{x} \in \mathbb{R}^{N_{at} \times N_f}$, where N_k is the number of species, N_f is the number of features, and N_{at} is the number of atoms. Therefore, this proposed version of BPNN still requires a different constituent network for each of the N_k species, but each one is only called once compared to the multiple times in the traditional formulation. Only a very slight reduction in accuracy was found to occur after this modification.

This BPNN structure-to-property model is pre-trained on real many-body descriptors $\boldsymbol{x}$ and energies $\boldsymbol{e}$ from the training dataset. This pre-training is done in order to prevent the model from learning the initial noisy generated descriptors $\boldsymbol{x}_g$ [88]. It uses LeakyReLU [125] from Equation (2.13) as the activation function in the internal layers, with a negative slope α of 0.1. A sigmoid output activation function [126] is used to match the normalised feature scaling that it applied to the property values. Each of the N_k sub-models in the BPNN consists of 5 fully connected linear layers with $\{N_f, N_E, N_E, N_E, 1\}$ neurons in each corresponding layer, where N_E is a hyperparameter used to control the model size with a default of 80. The training objective of the model was to minimise the mean-squared error (MSE) loss function $\frac{1}{m_b}\sum_{i=1}^{m_b}(\boldsymbol{e}_i - \hat{\boldsymbol{e}}_i)^2$, where m_b is the mini-batch size and $\hat{\boldsymbol{e}}$ are the predicted energies. An Adam gradient descent optimiser [106] is used with a learning rate of $8{\times}10^{-4}$. Overfitting is prevented using early stopping, where the MSE on the test set is monitored, with the best performing model being chosen if no improvement occurs after 400 epochs. The maximum number of training epochs was 1000.

The accuracies of the BPNN model for each of the ten molecule types can be seen in Figure 5.14. It is clear that all model predictions lie close to the dotted line of best fit, and hence the models can be said to be reasonably accurate. The most accurate results can be seen in the case of benzene, which is unsurprising considering it has the least torsional degrees of freedom and only has 12 atoms. The worst accuracies are exhibited in the case of aspirin, which has the most torsional degrees of freedom.

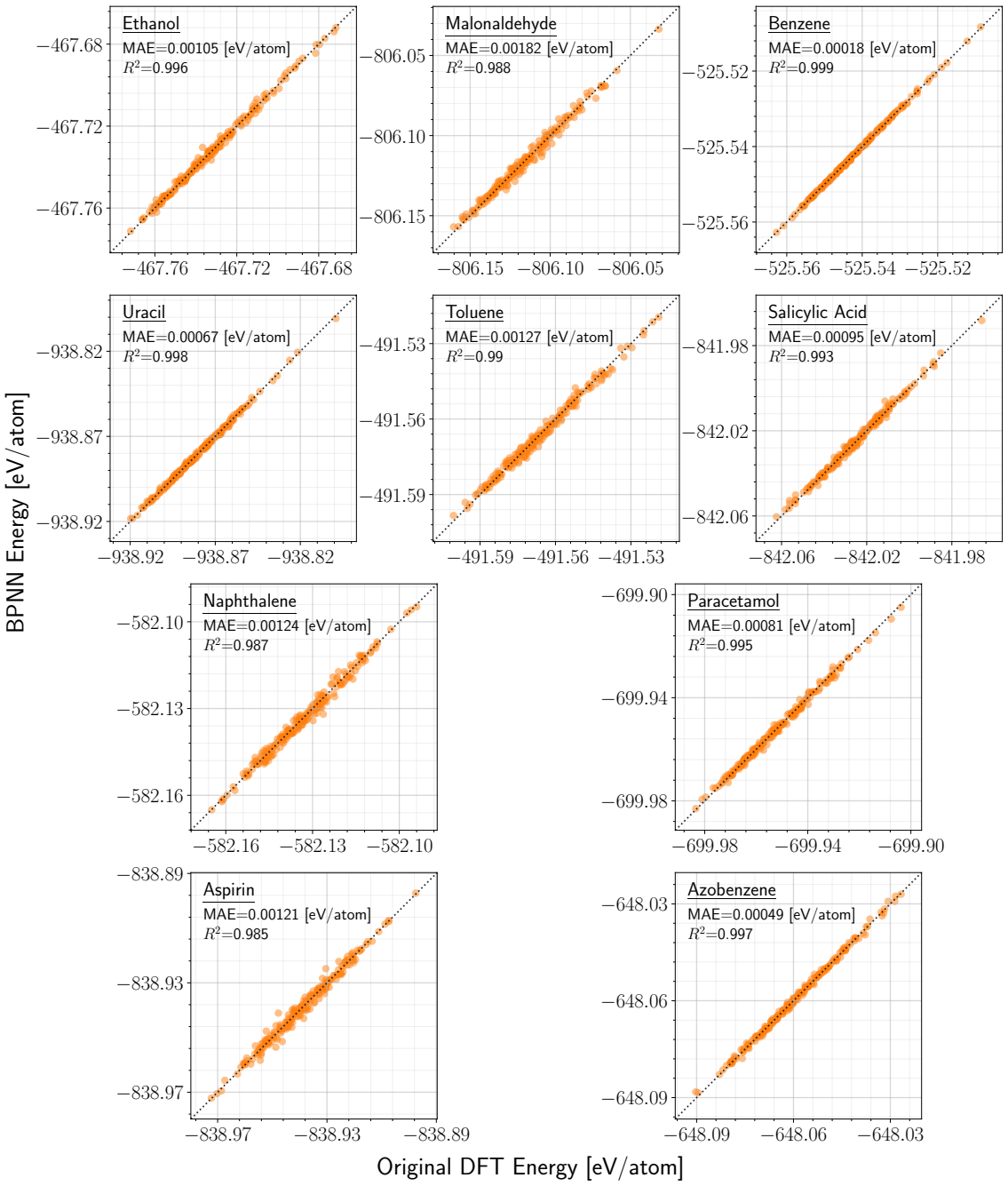

Figure 5.14: Figure showing the parity plot accuracies of a Behler-Parrinello neural network (BPNN) structure-to-property model for all ten molecule types in the revised MD-17 dataset [87]. The x-axis shows the original DFT energy labels e from the MD-17 dataset. The y-axis shows the BPNN predicted energies $\hat{e}$ for the given many-body descriptors $\boldsymbol{x}$. Included are the mean-absolute errors (MAE) in units of eV/atom, coefficients of determination R^2 [304], and line of best fits for each of the ten molecule types.

5.5.2 Property Latent Space

Most generative models work by learning the mapping from a latent space to the training data space. The latent space is simply a user-defined vector space of arbitrary dimension d. Latent vectors $\boldsymbol{z} \in \mathbb{R}^d$, represent points in this vector space, and can be randomly sampled using a multivariate probability distribution p_z. The probability distribution is commonly chosen as either a uniform distribution $\boldsymbol{z} \sim \mathcal{U}_d(-1, 1)$ or as a normal distribution $\boldsymbol{z} \sim \mathcal{N}_d(0, 1)$. A normal distribution is used in this work, as it matches the distributions of both the neural network weight initialisations and the scaling of the feature vectors.

The objective of this work is to have the ability to selectively generate molecules for a specified target property value. This is relevant in the scenario where the training dataset with samples $\boldsymbol{x}$ also contains additional class labels $\boldsymbol{y}$. These labels $\boldsymbol{y}$ might represent energy values, HOMO-LUMO gap values, see Section 3.3.2, or any number of physically observable quantities. In this case, the labels $\boldsymbol{y}$ are not discrete categories but are continuous values instead. Therefore, some adaptation to the standard formulation of a conditional GAN is required, as detailed in Section 2.4.3. In this chapter, DFT total energy values [50] are the only properties that are used. But the method should, in principle, extend to the use of other, more advanced properties. Consider a dataset such as the revised MD-17 dataset [87], where each configuration $\boldsymbol{r} \in \mathbb{R}^{N_{at} \times 3}$ contains N_{at} atoms, with associated labels of scalar DFT energy values $e \in \mathbb{R}$. The training dataset consists of m pairs of real samples $\{(\boldsymbol{r}^{(i)}, e^{(i)})\}_{i=1}^{m}$. In comparison, an arbitrary number of generated configurations and target property pairs $\{\boldsymbol{r}_g, e_g\}$ can be obtained. The set of real energies $\{e^{(i)}\}_{i=1}^{m}$ forms a distribution p_e.

In the standard formulation of a GAN, with no conditioning, generated samples $G(\boldsymbol{z})$ form a distribution p_g, which will approximate the distribution of real samples p_{data} as $p_g \approx p_{\text{data}}$ after training successfully completes. In order to have control over the energies of the generated configurations, it is necessary to use a conditional GAN, which takes a secondary input e_g drawn from an additional energy latent space. These latent energies e_g represent the desired energies of the generated configurations $\boldsymbol{r}_g$. How this energy latent space is defined is of importance. The standard GAN latent space p_z is rarely ever interacted with by the user, latent vectors $\boldsymbol{z}$ are simply

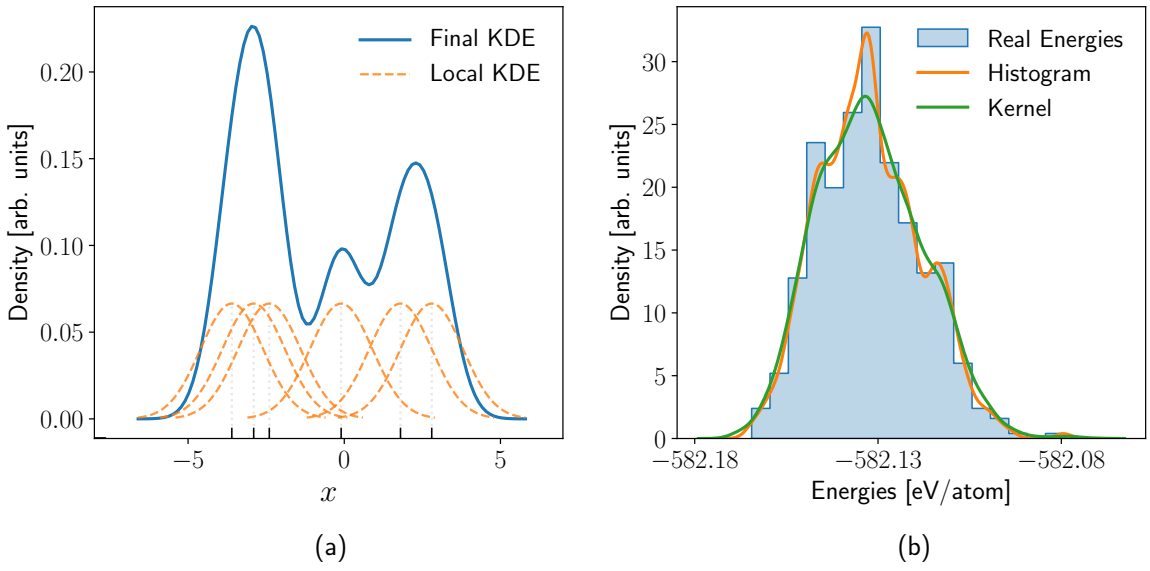

Figure 5.15: Figure showing how the underlying property distribution p_e is approximated as p_{e_g}. Panel (a) shows how a kernel density estimation (KDE) is formed from a summation of local density estimations around each random data point x. Panel (b) displays a KDE fit and a histogram fit to a set of 500 naphthalene energy values taken from the revised MD-17 dataset [87]. It can be seen that KDE is a more appropriate choice as the histogram fit tends to overfit to the data.

randomly sampled. To selectively generate with specified energies, it is necessary to enforce that the latent energies e_g correspond, as close as possible, to the actual DFT energies e_g^{DFT} of the generated configurations $\boldsymbol{r}_g$. The most naive approach to defining this energy latent space would be to just randomly sample real energies from the training set $\{e^{(i)}\}_{i=1}^{m}$, which is the approach taken in Fung et al. [225]. However, the objective is to have a model that can extrapolate out to new data regimes and to interpolate between known samples. It was thought that the model may do these optimally, if it only uses real energies e throughout the training process. This leads to the problem of how to approximate the real distribution of energies p_e.

Methods from the field of probability density estimation [305–307] are used to create an approximate distribution p_{e_g} given the distribution of real energies p_e. A simple and intuitive method of using a histogram was initially explored. This approach worked by placing all energies into bins of fixed width, ranging from the minimum

to the maximum values[3], then calculating the cumulative distribution function. It is then a matter of sampling, where the naive approach is to simply choose a bin with the relevant probability from the cumulative distribution function, and then sample a random latent energy e_g from a uniform distribution between the upper and lower bounds of that bin. It would have been better to sample from the cumulative distribution function using inverse transform sampling [308]. This simple histogram method was later changed to a more sophisticated, and easier to implement, kernel density estimation [305, 306].

Kernel density estimation (KDE) is a widely used method for probability density estimation. In histogram plots, for example, a smooth KDE curve is often included for ease of visualisation. KDE is a non-parametric method that approximates a distribution as a smooth continuous function, which can then be sampled from at will [309]. A visualisation of how the KDE is constructed using a sum of local density estimations is included in of Figure 5.15(a). More formally, given a set of m samples, $(x^{(1)}, x^{(2)}, \ldots, x^{(m)})$, independently drawn from the same distribution p_x, the density $f(x)$ is estimated. The local probability density is modelled around each sample x, typically as a normalised Gaussian of the form:

$$\mathrm{K}(x) = \frac{1}{\sqrt{2\pi}}\, e^{\left(-\frac{1}{2}x^2\right)}, \tag{5.7}$$

where K is said to be the kernel function. The final density estimation is taken as the sum of all these local kernel densities for all samples x:

$$f(x) = \frac{1}{m\,h} \sum_{i=1}^{m} \mathrm{K}\left(\frac{x - x_i}{h}\right), \tag{5.8}$$

where h is a smoothing parameter and m is the number of samples. The smoothing parameter h is set according to Scott's rule [307]: $h = m^{\left(-\frac{1}{d+4}\right)}$ where d is the dimension of the distribution p_x, being equal to one in this work as only univariate property distributions are considered (e.g., just energies).

In summary, kernel density estimation is used to form an approximate distribution p_{e_g} given the real distribution of properties p_e of the training set. During the training

[3]Increasing these maximum and minimum energy bounds by an additional $5-10\,\%$ was experimented with. This allowed the model to extrapolate out into data regimes beyond that contained within the training set.

of this conditional version of GANdalf, the target or desired properties are drawn from this approximated distribution p_{e_g}. This allows the model to correctly learn the underlying data distribution p_{data}, while still retaining the ability to generalise out to new data regimes.

5.5.3 Results

The efficacy of this proposed method of property conditioning is now examined. Proper analysis requires more than just the pair and angular distribution functions already used. However, it is noted than the configurations generated using this conditioning method do have similar distribution functions to those previously shown in Figure 5.6 and Figure 5.7. It is briefly confirmed in Figure 5.16 that the generated configurations are still structurally similar to the real ones. Then it must be verified that the actual property values of the generated configurations correspond to the desired values that were inputted into the GANdalf model. This can be thought of as checking the tuning of a speaker; making sure that the outputted decibel level actually matches what is specified by the speaker control dial.

The training details of this property-conditioned version of GANdalf are much the same as described in Section 5.3.2. The additional latent energies e_g are simply concatenated onto the random noise vectors $\boldsymbol{z}$, before input into both G and D. In order to assign higher importance to the latent energies e_g, they were concatenated a total of ten times. This means that the number of neurons in the input layer of G is therefore $10 + \|\boldsymbol{z}\|$. An additional loss function term is added to G as described in Section 2.4.3. This secondary training objective of G is to minimise the mean squared error (MSE) difference between the latent energies e_g and the BPNN predicted energies $\hat{e}_g$. This MSE difference is added to the standard Wasserstein G loss function as

$$L_G = -\mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_z}[D(G(\boldsymbol{z}))] + \lambda_{\text{e}} \mathop{\mathbb{E}}_{e_g\sim p_{e_g}}[(e_g - \hat{e}_g)^2], \tag{5.9}$$

where λ_{e} is a coupling hyperparameter that defaults to 10 and p_{e_g} is a approximation of the real energy distribution.

A method to verify that the generated configurations are similar to the real configurations is called Principal Component Analysis (PCA) [304, 310]. This is a linear-

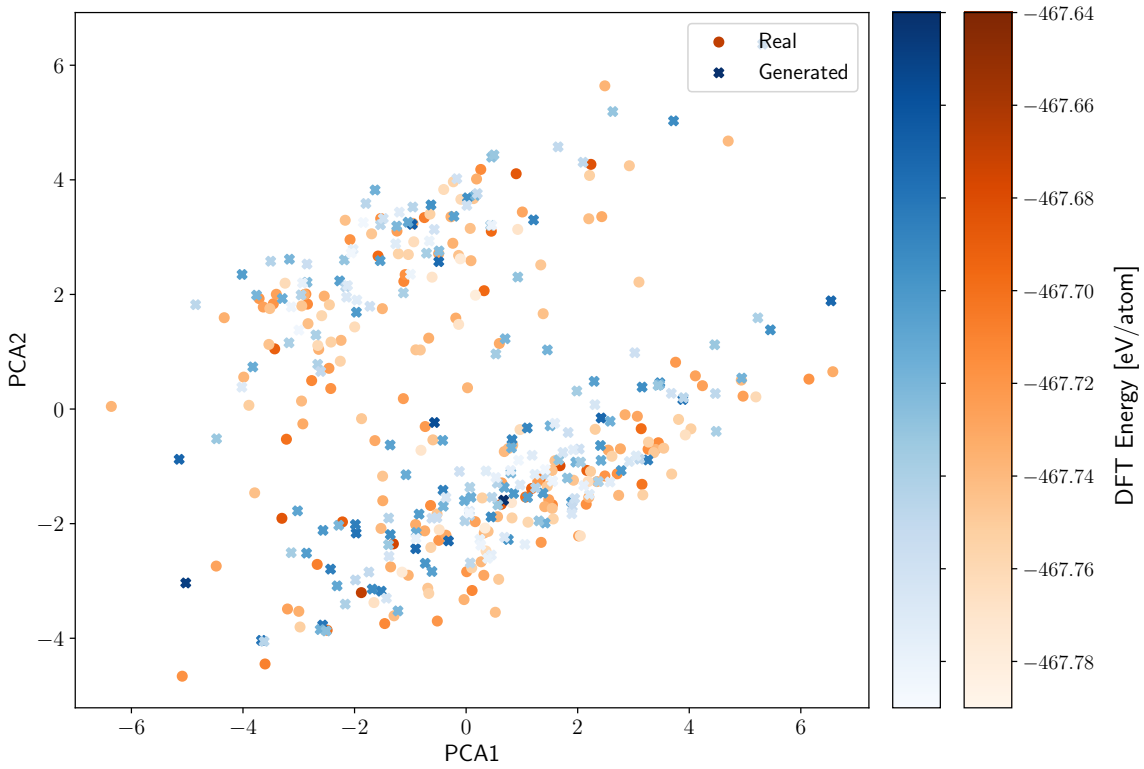

Figure 5.16: Figure showing the principal component analysis (PCA) of the Cartesian coordinates of the real and the generated atomic configurations of ethanol. PCA is used to visualise high-dimensional data in a 2D space. The real configurations are projected into this 2D space using PCA. The generated configurations are also included using the same projection as the real data. The DFT energies of both are included as colour bars. A total of 200 configurations are shown for both the real and generated cases.

dimensionality reduction technique that projects high-dimensional data into a lower-dimensional space using a set of orthogonal components. These components are chosen along principal axes such that the variance of the data is maximised. PCA is commonly used to visualise multivariate datasets in a 2D or 3D space. Similar samples will be grouped close together in this reduced space. In order to compare two different datasets, the orthogonal components are determined using the first dataset and then both datasets undergo the same transformation. This ensures that both datasets are transformed into the same space. It is crucial that all data is properly feature-scaled beforehand in order for the method to work, usually by standardising to a mean zero and a standard deviation of one. In Figure 5.16, 200 real configurations $\boldsymbol{r}$ and 200 generated configurations $\boldsymbol{r}_g$ of ethanol are compared using PCA. All configurations

are first aligned by performing rotations and translations such that the root mean square deviation between each configuration is minimised [279, 280]. Two distinct bands can be seen in this 2D figure, with both sets of Cartesian coordinates $\boldsymbol{r}$ and $\boldsymbol{r}_g$ occupying them with a similar distribution. Energy values of every configuration are included as colour bars. No groupings of similar energy values were found, with real and generated energies seemingly randomly distributed. From this figure it can be concluded that the sets of real and generated configurations are broadly similar. Similar plots can be seen for the other nine molecule types.

The workflow used to verify the accuracy of this proposed conditioning method is now detailed. The latent energies e_g are inputted into the generator G, alongside the usual random noise $\boldsymbol{z}$. The generated many-body descriptors $\boldsymbol{x}_g = G(\boldsymbol{z}, e_g)$ are then inputted into the new regression model E, outputting the predicted energies $\hat{e}_g = E(\boldsymbol{x}_g)$. Measuring the alignment of the desired energies e_g to the predicted energies $\hat{e}_g$ serves as the first method of verification. The second, more rigorous method of verification involves the use of DFT. Applying the local many-body inversion method to the descriptors $\boldsymbol{x}_g$ results in atomic configurations $\boldsymbol{r}_g$. The DFT energies e_g^{DFT} of the generated configurations $\boldsymbol{r}_g$ can then be calculated using PySCF [42–44]. The alignment of the desired energies e_g to the actual generated DFT energies e_g^{DFT} serves as the fundamental test of this proposed conditioning method. Both of these verification methods are shown in Figure 5.17.

The primary results of this chapter are shown in Figure 5.17 for all ten molecule types. It shows parity plots with the desired energies e_g on the x-axis and the BPNN predicted energies $\hat{e}_g$ and the DFT calculated energies e_g^{DFT} on the y-axis. A number of insights can be gained from examining these parity plots. Firstly, the property conditioning method can be deemed to be successful. There exists a clear linear relationship between the desired energies e_g and the DFT calculated energies e_g^{DFT}. It is noted that eight out of ten of the GANdalf models have an R^2 value [304] greater than 0.8. As previously stated, the method of sampling the desired energies e_g is of importance. The method that is used throughout the training of GANdalf is to sample from p_{g_e}, which is the kernel-density estimate [309] of the real energy distribution p_e. This method of sampling e_g was used to make the BPNN predictions shown in orange. This shows the regions where the majority of training samples occur, and hence where

the majority of learning occurs. It can be seen that the most accurate DFT energies e_g^{DFT} tend to lie in the regions containing the majority of the training samples.

The desired energies e_g corresponding to the DFT energies e_g^{DFT} that are shown in blue in Figure 5.17, were uniformly sampled between the minimum and maximum energies contained in the real distribution p_e. This was done to showcase the full capabilities of the GANdalf model, whereby energetically reasonable configurations can be generated even in those regions where few training samples occur. The accuracy in these low-data regions is correspondingly lower but still obeys the correct linear relationship in most cases. As the desired energies e_g approach the maximum permissible value, the DFT energies become less and less accurate. There are a number of prominent outliers in the high energy regions, particularly in the larger molecules of paracetamol, aspirin, and azobenzene. It seems that the conditioning performance is affected more by the number of species and the number of torsional degrees of freedom than it is by the number of atoms. Azobenzene performs quite well despite being the largest molecule with 24 atoms. Aspirin is once again the most difficult case, exhibiting a very weak relationship between the desired and the DFT energies.

The accuracy of the property prediction model E, which is included in the GANdalf inner training loop, is of critical importance. It is a direct source of error during training, contributing to the uncertainty in the calculated DFT energies. In the best case scenario, the error in the predicted energies $\hat{e}_g$ of the generated descriptors $\boldsymbol{x}_g$ is comparable to the error of the real descriptors $\boldsymbol{x}$ as shown in Figure 5.14. Other significant sources of error include the local many-body inversion method, with the minimum possible error able to be estimated using Figure 4.7 and Figure 4.8. It is noted error that the inherent error in the inversion method is particularly large in the cases of aspirin and paracetamol. In conclusion, the inaccuracies shown by the calculated DFT energies in Figure 5.17 are due to a number of factors including: (1) the error of the predictor E, (2) the error in the inversion method, and (3) generating in low-data regimes.

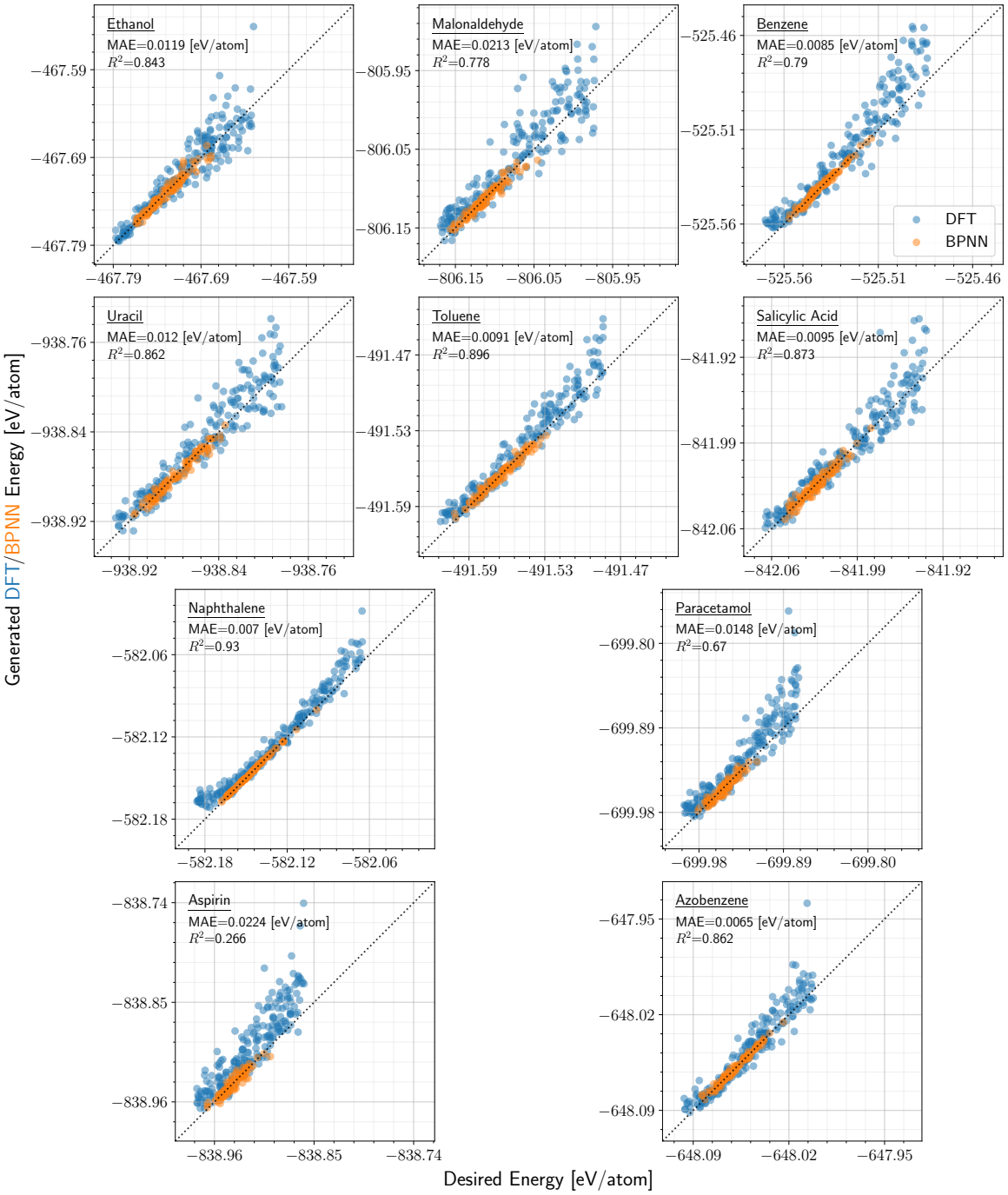

Figure 5.17: Figure showing parity plots of property-conditioned GAN models for all ten molecule types in the revised MD-17 dataset [87]. Each models' accuracy is assessed by comparing desired energies to the generated energies. The x-axis shows the desired latent energies e_g inputted into the generator G. The y-axis shows the energies of the configurations outputted by the model. In blue, are the DFT energies e_g^{DFT} of the generated configurations $\boldsymbol{r}_g$ after inversion, with their corresponding MAE and R^2 accuracies. In orange, are the BPNN predicted energies $\hat{e}_g$ of the generated many-body descriptors $\boldsymbol{x}_g$. The BPNN desired energies are sampled from the approximated energy distribution p_{e_g} used during training. The DFT desired energies are uniformly sampled between the minimum and maximum values of the real distribution p_e.

5.6 Summary

This chapter uses generative modelling to create new atomic configurations that are similar to those contained within the training dataset. The only training dataset considered is one consisting of molecular dynamics trajectories [194] for ten different small molecules [87]. Generative adversarial networks (GANs) [77] are used as the generative model. The so-called GANdalf framework outputs configurations in the format of many-body descriptors, such as bispectrum components [75]. This means that a high degree of structural accuracy is learned by the model. These generated descriptors are then converted into human-interpretable Cartesian coordinates using the local many-body inversion method developed in Chapter 4. The use of this inversion method places restrictions on GANdalf; the model cannot extrapolate out to new molecule types that were not included in the training dataset. GANdalf can only generate conformations of the molecule types such as ethanol, aspirin, and paracetamol, which were included in the training dataset.

The developmental history of GANdalf was detailed. Preliminary work was included to illustrate the reasons for using many-body descriptors. The basic version of the developed GAN architecture was then described. This first version works with only one molecule type in the training dataset. It was shown that the pair and angular distribution functions for both the real and generated configurations were similar. A conditional GAN model [32] was successfully implemented to allow for the inclusion of multiple molecule types in the training set. Finally, a machine learned interatomic potential [83] was included in the GAN training loop, in order to selectively generate configurations with desired property values. It was confirmed that the desired energy values align, to a reasonable degree of accuracy, with the density-functional theory energies of the generated configurations. It can therefore be concluded, that the proposed GANdalf framework is capable of generating realistic atomic configurations for desired property values. This corresponds to an inverse design workflow where structures are directly outputted for a given input property value.

Chapter 6

Language Modelling of Molecular Blocks

This chapter uses language modelling to generate diverse molecules for specified property values. A novel fragment-based textual representation called MolBlox is developed. An inversion scheme to reconstruct the generated MolBlox sequences into valid molecules is detailed. Molecules generated by pre-trained and fine-tuned models are verified. Both physicochemical and quantum-mechanical properties are successfully used for targeted generation.

6.1 Introduction

The objective of this chapter is to create a generative model that is capable of outputting a diverse range of novel molecules. In the previous Chapter 5, only conformations of known molecule types were able to be generated. This lack of diversity is considered a significant disadvantage of the GANdalf method and this chapter seeks to address it. The design choices here taken are motivated by this requirement to increase diversity in the generated molecules. The two major changes from Chapter 5 are (1) the generative-modelling architecture and (2) the chemical representation. These two choices are required to be complementary; the model architecture must be compatible with the representation. A text-based representation is adopted to allow for increased diversity. This includes no three-dimensional (3D) structural informa-

tion. The standard textual chemical representation is called SMILES [72], see Section 6.2.1. A molecule of paracetamol would have a SMILES of CC(=O)Nc1ccc(O)cc1. A generative pre-trained transformer (GPT) is consequentially employed, as the most appropriate generative model for a text-based representation, see Section 2.5.2. These are notable differences, changing the intended use case of the workflow significantly.

The previous Chapter 5 used generative adversarial networks (GANs) [77], as they were compatible with bispectrum components [75], the chosen chemical representation. Bispectrum components are essentially in the format of a 2D image (see Figure 3.3), which is what GANs are typically applied to [32,77,134,139,155]. The developed GANdalf framework can only generate conformations of the molecule types included in the training dataset. This is a major restriction, which is caused by the local many-body inversion method from Chapter 4. Despite the fact that the inversion method was instrumental in GANdalf's ability to generate realistic 3D conformations, it is now removed from the workflow. There are too many restrictions introduced by its requirement for an initial structure to begin the inversion procedure from. In order to fulfil the objective of increasing the diversity of the generated molecules, a less sophisticated representation than many-body descriptors was adopted, allowing for the removal of the inversion method.

Numerous different generative-modelling frameworks have been applied to chemical systems in recent times. These include various combinations of model architectures and chemical representations. The landmark paper that first demonstrated the potential of generative models was Gómez-Bombarelli et al. [30]. They generated novel molecules in a textual representation called SMILES, see Section 6.2.1, by randomly sampling from the latent space of a variational autoencoder [113]. They also showed that an additional property predictor model can be used to guide the generation process, similar to our method presented in Chapter 5. Various other early attempts have succeeded in generating SMILES strings [311, 312] but lack any means of property-conditioning, which is required for effective inverse chemical design. Some success has been found at generating textual molecules with generative adversarial networks [34,218] and with recurrent neural networks [313]. More modern models are typically based on the transformer architecture [29], due to the success of large language models [76, 78, 97, 144, 145, 164–169, 171]. Transformers are deemed

highly compatible with the textual chemical representations considered here, as their training objective is to learn the next symbol in any given sequence of text. Prominent in this work is MolGPT [33], see Section 6.3, whose basic architecture is modified and used as the generative model in this chapter.

As has been previously stated, a major objective of this thesis is to train generative models that can create molecules for selective property values. As described in Chapter 1, there are numerous different types of properties which are suitable depending on the nature of the training data. There exist a number of so-called two-dimensional (2D) properties that are commonly used in the field of cheminformatics [30, 33–36]. No 3D structural information is used in calculating these properties, only the SMILES structure is, hence the name of 2D properties. Four different 2D properties are used in this work for the conditioning of generative models. These include hydrophobicity (logP) as measured by the logarithm of the octanol-water partition coefficient [37], and the topological polar surface area (TPSA) [38]. Both of these can be considered physicochemical properties [314], which respectively affect biological absorption and barrier permeability [41]. A synthetic accessibility score (SAS) [39] is used to heuristically determine how easily a molecule can be synthesised, based on structural complexity and fragment rarity. SAS values range from zero which is easily synthesisable to ten which is considered difficult. A final 2D property is the quantitative estimate of druglikeness (QED), which consists of a weighted sum of eight different molecular properties[1]. QED evaluates a molecule's druglikeness as a score between zero and one, with one being a high druglikeness.

Another aim of this chapter is to condition the generative model on properties derived from quantum-mechanical electronic structure calculations [42–44]. These can be classified as full 3D properties, requiring 3D atomic coordinates to be known, as opposed to the previous 2D physicochemical properties [37–40] which only requires the SMILES. Three such quantum-mechanical properties are considered as targets in this work. These are the total energy at 0K, the HOMO-LUMO gap energy, see Section 3.3.2, and the dipole moment. The total energy includes kinetic energy, potential, and internal energy contributions, see Section 3.3.1. It is crucial in assessing the stability

[1]These eight constituent properties of QED are: molecular weight, hydrophobicity, number of hydrogen bond donors, number of hydrogen bond acceptors, topological polar surface area, number of rotatable bonds, number of aromatic rings, and number of structural alerts.

of a molecular system. The HOMO-LUMO gap is the energy difference between the highest occupied molecular orbital (HOMO) and the lowest unoccupied molecular orbital (LUMO) [49]. High values of this property may imply chemical stability, while lower values may indicate chemical reactivity. Lastly, the dipole moment is a vector quantity that measures the difference between positive and negative partial charges within a molecule. Polar versus non-polar is a fundamental concept in chemistry, affecting solubility and general behaviour in electric fields. Having the ability to selectively generate molecules for desired values of these 3D properties improves the versatility and usefulness of the workflow.

The open-source cheminformatics package RDKit [281] is used throughout this chapter. This package constructs a molecule object from a SMILES, which can then be used to determine neighbouring atoms, physicochemical properties [37–40], and much more. The ETKDG [91,92] method of conformer generation is used to convert SMILES into 3D structures, as implemented by RDKit. The HuggingFace package [315, 316] is used for tokenisation techniques. The GPT model is implemented using a modified version of the open-source code NanoGPT [178]. PyTorch [295] is used as the deep learning framework, for its ease of use and GPU acceleration. The density-functional theory (DFT) [50] package PySCF [42–44] is used to perform electronic structure calculations and geometry optimisations.

There are two different training datasets used in this chapter, one for pre-training and one for fine-tuning. The pre-training dataset is called GuacaMol [89] and is widely used as a benchmark for drug discovery tasks within the field of cheminformatics [33,317,318]. It provides numerous standardised metrics that can be used to compare to existing generative models. This dataset consists of around 1.5 million SMILES, with the largest molecule containing 250 atoms. It is formed as a subset of the ChEMBL database [12], which is manually curated to only contain bioactive molecules with drug-like properties. This GuacaMol subset is created by filtering out salts, ions, and SMILES of greater than 100 characters. Molecules which contain chemical species other than H, B, C, N, O, F, Si, P, S, Cl, Se, Br, and I are not included. Furthermore, molecules with a high degree of similarity [319] to a set of 10 marketed drugs[2] are excluded, ensuring that these 10 drugs can be safely used as a holdout validation set.

[2]This set of known drugs is: aelecoxib, aripiprazole, cobimetinib, osimertinib, troglitazone, ranolazine, thiothixene, albuterol, fexofenadine, and mestranol.

The QM9 dataset [14, 90] is used to fine-tune a GPT model already pre-trained on the GuacaMol dataset. QM9 is a smaller, more domain-specific dataset containing 134,000 molecules. It is formed as a subset of the GDB17 database [14], containing only those molecules with fewer than nine atoms of the species (C,O,N,F), not including hydrogens. The underlying GDB17 database contains synthetic molecules, constructed using combinatorial techniques [19]. This means that QM9 allows for a larger chemical space to be explored. The most significant feature of the QM9 dataset is that every molecule is labelled with 12 different DFT properties. These DFT calculations are performed using a B3LYP functional [199–202] and 6-31G(2df,p) basis set [320,321], see Section 3.3.1. A series of increasingly strict electronic and geometric convergence thresholds were employed. Despite QM9 containing both SMILES and optimised 3D structures, the GPT model is only trained using SMILES. A number of the 12 different 3D properties are used to condition the generative model. These include the total energies, the dipole moments, and the HOMO-LUMO gaps.

The layout of this chapter is as follows. The newly-adopted text-based chemical representation will first be introduced. Two different textual representation are used, the standard SMILES as well as MolBlox, a new representation that is proposed in this thesis. An inversion method to reconstruct the original SMILES from a given MolBlox sequence is then described. Section 6.3 gives the specifics architecture of the GPT model used in this chapter. The results of the workflow are then analysed, on both the pre-training [89] and the fine-tuning datasets [14, 90]. The accuracy of the property conditioning is also assessed.

6.2 Textual Representation

An objective of this chapter is to create a generative-modelling framework that addresses the limitations and challenges encountered during Chapter 5. It was therefore decided to replace quantum-mechanically accurate many-body descriptors with a simpler representation, more appropriate for generating diverse molecules. The most obvious candidate would be a two-dimensional (2D) representation such as SMILES (Simplified Molecular Input Line Entry System) [72]. This is a method of encoding molecules as a text-based string. It is compact and machine-readable, making it a

widely used representation within the field of cheminformatics. A visual example of a molecule and its corresponding SMILES is shown in Figure 6.1. Molecular training datasets [13, 89, 90] are commonly just a list of SMILES. There also exist large online chemical databases [10, 11] which can be searched using SMILES. There already exist multiple generative models trained to output molecules in a SMILES representation [30,33,218,322,323]. One way that our proposed method differentiates itself from these existing methods is in the novel MolBlox representation. In this section SMILES is first described, before progressing onto the proposed MolBlox representation.

6.2.1 SMILES

The SMILES representation is 2D, meaning that it only includes information such as chemical species, bond types, and atom connectivities. No 3D information about atomic positions is encoded, rendering SMILES unable to distinguish between different conformations of the same molecule. However, there exist packages [91,324] which can be used to generate estimates for the 3D atomic coordinates given a molecule in SMILES. The small number of rules used to construct SMILES are now briefly detailed. Each atom is denoted with its associated chemical symbol (C for carbon, N for nitrogen, etc), except hydrogen atoms which are left implicit based on the remaining valences. Adjacent atoms are either single (−), double (=), or triple (#) bonded, with single bonds often being left unspecified. Parentheses are used to specify branches off the proceeding sequence e.g., C(C=O)CO. In aromatic rings, where bonds alternate between single and double, the atom symbols are generally made lowercase, with special aromatic bonds used to dynamically specify single or double bonds. Aromatic rings are properly closed by appending a number to adjacent atoms in the ring e.g., Oc1nccc(O)n1. Ionic charges (and explicit hydrogens) can be added using square brackets e.g., CO[N+](=O)[O−].

The aim of generative language modelling is to predict the next token in a given sequence of tokens. These tokens can represent words in a sentence, notes in a song [166], or visual patches in an image [97]. It is often necessary to split the raw dataset samples into some sequence of suitable tokens. In a natural language scenario, a balance between word-level and character-level tokenisation is often considered optimal. The set of all unique tokens, or the vocabulary, would therefore consist of subwords such as

'chem', 'istry', and 'bio', 'logy'. A similar tokenisation strategy must be performed on SMILES to optimise them for use with language models. A SMILES tokeniser was developed by Schwaller et al. [325], and is widely used [33,322,323] as it is tailored for the specific semantic structure of SMILES. It converts SMILES into a sequence of tokens, for example, C(Cl)O would become [C,(,Cl ,),O]. Atom symbols, parentheses, ions, and bond types all are independently converted to tokens. In all tokenisation schemes, it is standard practice to prepend a start-of-sequence token '<sos>' and to append an end-of-sequence token '<eos>' to every sequence. Additionally, a padding token '<pad>' is iteratively appended until the maximum sequence length T is reached. All sequences must have the same dimension to allow for parallel training [120]. Various other special tokens can be used depending on the training objective. The final step in any tokeniser is to convert the tokens into integers that correspond to indexes in the vocabulary, for example, the sequence [<sos>,C,(,Cl ,),O,<eos>,<pad>,… ,<pad>] might become [587,11,12,23,13,14,588,590,… ,590].

6.2.2 MolBlox

The text-based representation proposed in this work is called MolBlox, and is motivated by the observation that all molecules essentially consist of a series of constituent fragments or molecular blocks. The set of all possible molecular blocks is small, with the vastness of the space of possible molecules [1] coming from the number of ways these blocks can be combined. An analogy is that of LEGO toy blocks, where a small set of blocks, with unique shapes and colours, can be used to construct an almost endless variety of objects. One advantage of such a molecule fragmentation strategy would be that it allows for more physical meaning to be encoded into the representation. Each fragment token would correspond to something physical, unlike in SMILES where specific syntax tokens are required e.g., '(', and '['. It would also eliminate the need for the model to learn the semantic structure of SMILES where, for example, parentheses must be properly closed. The MolBlox vocabulary consists of the set of all possible fragments.

MolBlox is an invertible representation that consists of text-based molecular blocks. Being invertible means that a MolBlox sequence outputted by a generative language model can be reconstructed back into a valid molecule. Considering that cheminformatics training datasets [13, 89] are typically just long lists of different SMILES, MolBlox is designed to be SMILES-based. MolBlox takes a SMILES and decomposes it into a sequence of molecular blocks, which can then be reconstructed to yield the original SMILES. Each individual molecular block explicitly encodes the neighbouring atoms around each non-hydrogen atom. Hydrogen atoms, with their valence of one, do not affect molecular scaffolds. In SMILES, they are often left implicit, and can be added based on remaining valencies. MolBlox does not require blocks to be centered on hydrogen atoms in order to be fully descriptive and invertible. However, hydrogens are explicitly included as neighbours in each molecular block. Only the nearest neighbours are considered: those atoms that are directly bonded to the central one. This was a specific design choice in order to simplify the method. It is expected that if larger fragments which included secondary or tertiary neighbours were also included in the vocabulary, then the accuracy of the representation would increase. As MolBlox are derived from SMILES, the same atom ordering is used, as well as the same branching structure. If a SMILES contains N non-hydrogen atoms, then the

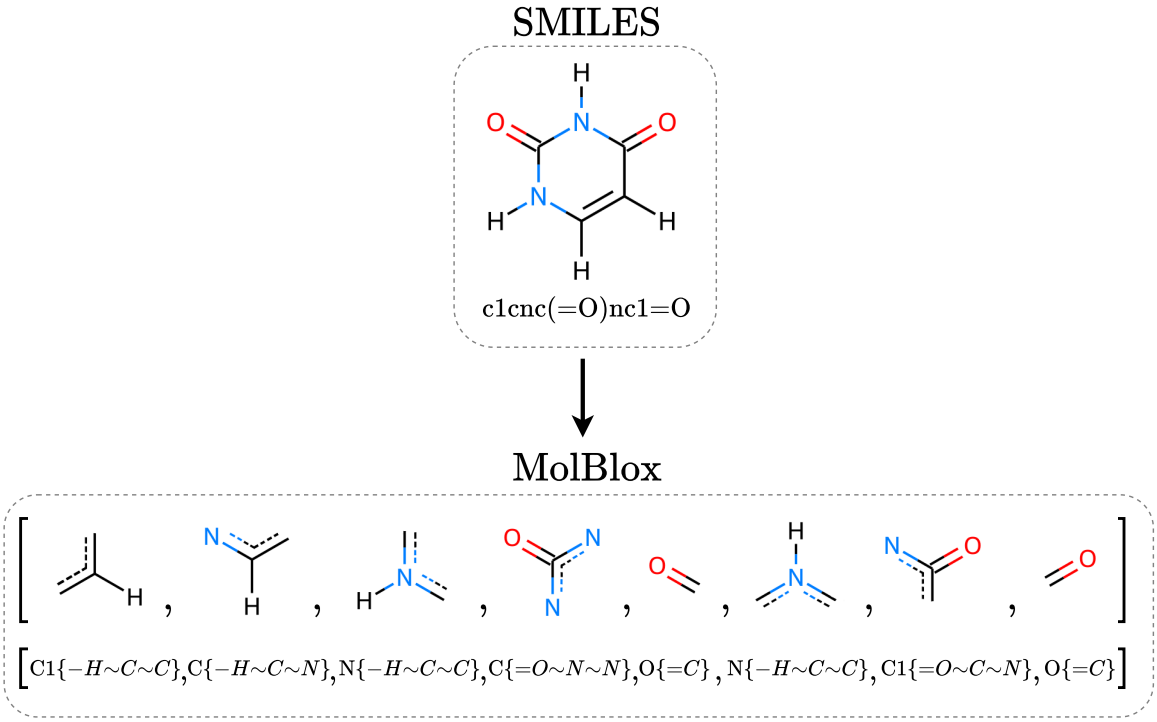

Figure 6.1: Diagram showing the construction of MolBlox from SMILES. A uracil molecule and its SMILES are shown at the top. The corresponding MolBlox sequence is shown on the bottom, in both graphical form and in the actual text-based form. In the text-based MolBlox representation, the central atom of each molecular block is first specified, followed, in braces, by all nearest neighbours with their corresponding bond types.

MolBlox sequence will be of length N. This means that MolBlox sequences are more compact that tokenised SMILES sequences due to the latter requiring syntax tokens for parenthesis and aromatics ring numbers.

In the MolBlox representation, each molecular block is essentially just a list of nearest neighbouring atoms that are bonded to some central non-hydrogen atom, with the bond types being explicitly specified. For example, consider a molecule of formic acid with SMILES of O=CO; the corresponding MolBlox sequence would be [O{=C}, C{−H−O=O}, O{−C−H}]. In each molecular block, the central atom comes first, followed by the neighbouring atoms in braces. The same bond type notation is used as in SMILES, with one addition: ∼ is used to denote the bonds in an aromatic ring. The valencies of aromatic bonds are unspecified, allowing for atom valencies to be balanced dynamically, as implemented in the RDKit cheminformatics package [281].

Rings are closed similarly to SMILES, by explicitly numbering two adjacent atoms in the ring. Figure 6.1 illustrates the formation of a MolBlox sequence of separate molecular blocks from a given SMILES. It can be seen from this figure, that adjacent central carbons atoms are numbered as C1, allowing for the ring to be closed. Also evident is that hydrogen atoms are included as neighbours within the braces, but not as central atoms with their own molecular blocks. Similar to the SMILES tokenisation scheme previously described, once a sequence of molecular blocks is obtained, it is must then be converted into a sequence of integer indexes in the MolBlox vocabulary $\mathcal{V}$. Special tokens such start-of-sequence, end-of-sequence, and padding tokens are also added as before.

6.2.3 MolBlox Inversion

In any generative model, the outputted samples will be in the same format as the training data. This presents challenges in a scenario where the training data has been transformed into some particular representation that is less easily interpretable, but has desirable characteristics (such as MolBlox). An inverse transformation must therefore be applied to the generated samples in order to interpret them. MolBlox are constructed from SMILES, with there existing an inverse transformation from MolBlox to SMILES. The MolBlox inversion method applies this inverse transformation, reconstructing SMILES from a MolBlox sequence. The inversion method is simple in principle but somewhat complicated in practice. A diagram showing an overview of the inversion method is shown in Figure 6.2. The raw sequences outputted by the generative model must first undergo some post-processing, where the special tokens such as padding tokens are removed, leaving only the token indexes. These token indexes are then converted back into their textual form using the vocabulary $\mathcal{V}$. The inversion method can then begin.

The inversion method works by combining all molecular blocks in the sequence together. This combination process begins with the construction of a so-called prototype molecule, using just the first molecular block. The next block in the sequence, which is called the candidate block, is then combined onto the prototype. It must be ensured that this candidate block is combined onto the prototype at the appropriate place, with the correct atom connectivities. This correct placement is determined by

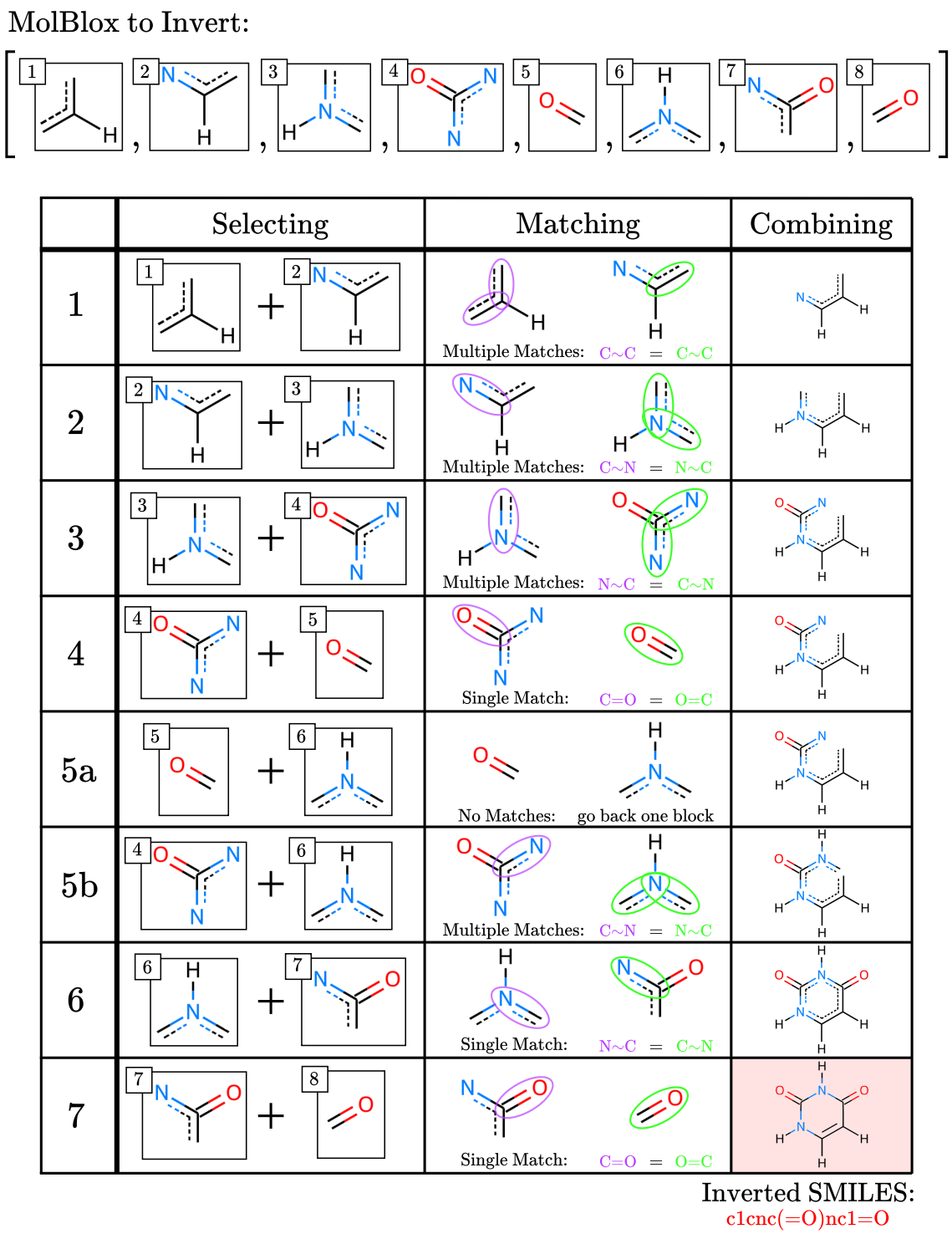

Figure 6.2: Diagram showing the inversion method that reconstructs a SMILES from a MolBlox sequence. The labelled MolBlox sequence of molecular blocks is shown at the top. The inversion process iteratively combines blocks together if they have matching, unoccupied, bonded pairs. If there are multiple matches then one is randomly chosen, with a subsequent full exploration of the decision tree. If there are no matches then the preceding blocks are iteratively selected until a bond match is found. The inverted SMILES is then determined using the fully recombined molecule.

a matching procedure, which examines atom species and bond types. The inversion method is an iterative process, where each successive molecular block in the MolBlox sequence is added to the prototype, until the end of the sequence is reached.

Finding those correctly matching molecular blocks forms the bulk of the inversion method. A key observation is that in order for two blocks to match, the central atom of each must be included in the neighbouring atoms of the other. Determining which neighbour refers to the central atom of the other block is achieved by comparing atom species and bond types. For example, consider the case of formic acid (O=CO) with MolBlox of [O{=C}, C{−H−O=O}, O{−C−H}]. The central atom comes first, followed by neighbouring atoms in braces. The first block contains a bonded pair O=C, while the second block has three bonded pairs: C−H, C−O, and C=O. The second block's last bonded pair of C=O is the only one of the three that matches the first block's O=C. Two matching bonded pairs must have the same atoms species and the same bond types. It can therefore be concluded that the =O neighbour in C{−H−O=O} is the central atom in O{=C}. Knowing this, the atoms in the other bonded pairs of C−H and C−O can now be correctly placed in the prototype molecule. Similarly, it can be concluded that the neighbour −C in the third block refers to the central atom of the second block. All atoms in the molecule can therefore be placed, the SMILES can be determined, and the inversion process can be considered complete.

There are three possible matching scenarios: single, multiple, or none. All three are illustrated in Figure 6.2. The most straightforward case is that of a single match, where atoms from the candidate molecular block can be immediately placed onto the prototype. In the second case, where no matches exist, it either means the MolBlox sequence is invalid and cannot be inverted, or else it means that the matching block to the candidate exists earlier in the sequence. This happens due to MolBlox using the same atom ordering as SMILES whereby branches are in parenthesis. It is impossible to order atoms from a branched molecule using just a one-dimensional sequence; some atoms which are not bonded together must be consecutive in the sequence. If there is no initial match, the candidate block will be matched against other preceding blocks. An example of this is shown in Figure 6.2 in rows 5a and 5b, where no match is found between blocks 6 and 5, so the method then tries to match 6 and 4, which is successful. It can be seen from the inverted SMILES at the bottom right of the figure,

that the fifth atom is a branch, hence it is not bonded to the sixth atom and no match was found. In the final of the three matching scenarios, multiple possible matches are found, then one is randomly selected, and the inversion method is continued. The unselected matches are considered as unchosen branches in a decision tree. For each MolBlox sequence, the inversion method is repeated until every branch in the tree has been explored. Most decision tree branches yield incomplete molecules which are discarded. If multiple valid molecules are returned by the inversion method, then all but the real one can be eliminated by comparing recalculated and pre-inversion MolBlox sequences.

This inversion method is sufficiently computationally efficient that, even in its unoptimised state, over ten thousand MolBlox sequences can be inverted every minute on a standard laptop, even for large systems with hundreds of atoms. There remain some specific edge cases where the inversion method fails. All molecules in the QM9 dataset [14, 90] and $\sim 96\%$ of those in the GuacaMol dataset [89] can be successfully inverted. It is hypothesised that this failure rate on the GuacaMol dataset can be brought down by addressing more of these edge cases.

6.3 MolGPT

Once it was decided to adopt a text-based representation instead of the previous many-body representation, the next major design choice was the generative-modelling architecture. Various different architectures have previously been used to generate molecules as SMILES. These include variational autoencoders [30], generative adversarial networks [218], and recurrent neural networks [313]. However, it was concluded that the hugely successful [76, 78, 97, 144, 145, 164–171] transformer architecture [29], which is specifically designed for learning sequences of text, would yield the best results. As this thesis is concerned with generative modelling, the specific variety of transformer that is used is a generative pre-trained transformer (GPT) [76, 78].

The workings of a GPT model are fully detailed in Section 2.5.2, but a brief summary is given here. Figure 2.6 shows the full GPT architecture, with a condensed version being shown in Figure 6.3. Consider a sequence of tokens $\boldsymbol{x} = (t_1, t_2, \ldots, t_T)$ of fixed length T. A token can be any arbitrary symbol in a sequence such as a words in

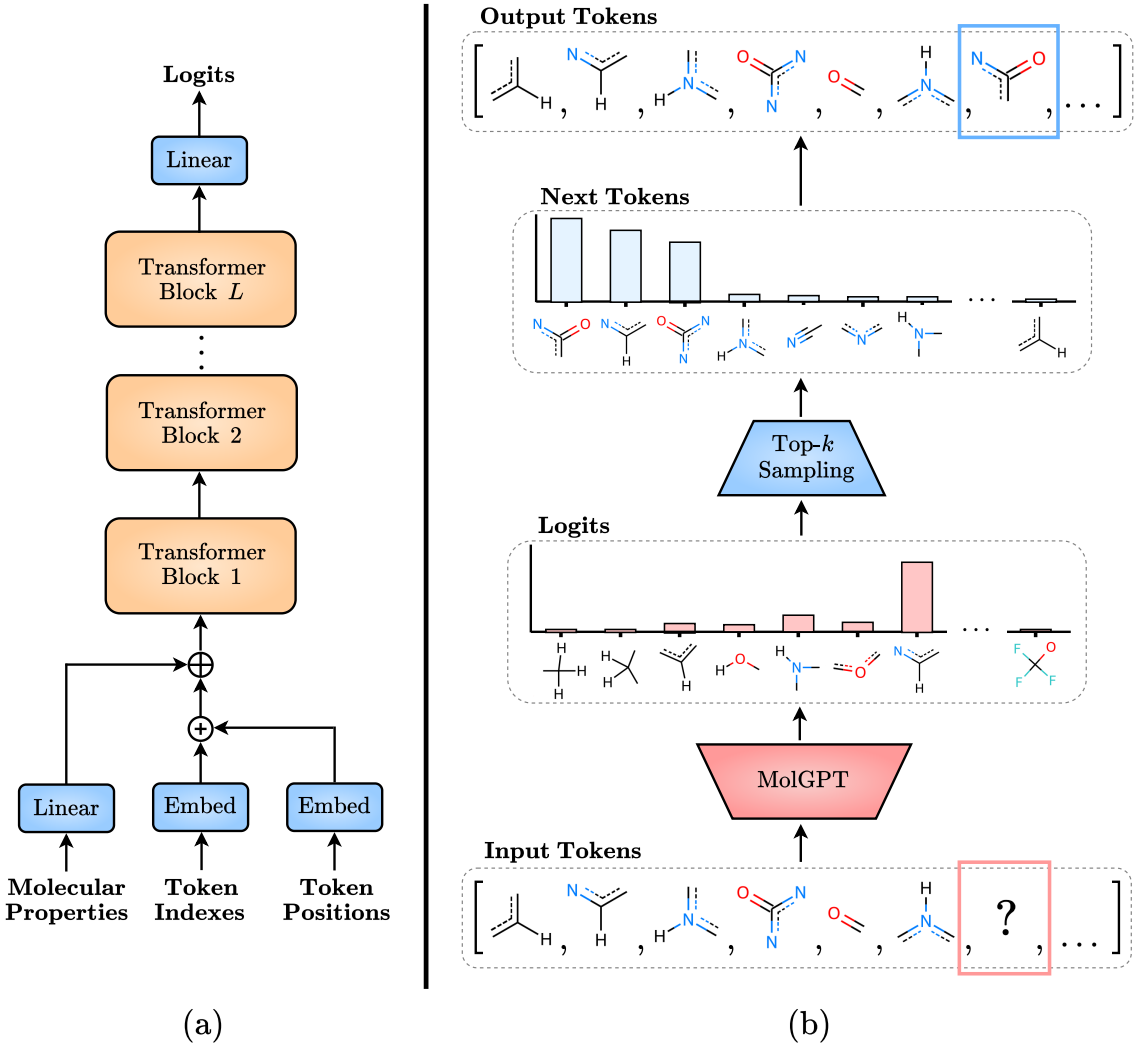

Figure 6.3: Diagram showing the architecture of MolGPT. Panel (a) shows model inputs being embedded and summed, before being concatenating with molecular property vectors. The resulting embeddings are then passed through L transformer blocks, yielding output logits. Panel (b) illustrates how MolGPT predicts the next token in a MolBlox sequence. First, MolGPT outputs unnormalised probability scores (logits) that each token in the vocabulary will be next. Top-k sampling redistributes the probabilities to only the k most likely tokens, and then randomly chooses one.

a sentence [29], or notes in a song [166]. The set of all unique tokens contained within the dataset is called the vocabulary $\mathcal{V}$. Each token is then represented as an integer index corresponding to its location in the vocabulary $\mathcal{V}$. Given a partial sequence of tokens $(t_1, t_2, \ldots, t_j)$ of length j where $j < T$, the training objective of a GPT model is to correctly estimate the probability that any token in the vocabulary $\mathcal{V}$ could be the next token t_{j+1}. Given this estimated probability distribution, an appropriate sampling method, such as top-k [184], chooses a realistic next token t_{j+1}. This top-k sampling method is illustrated in Figure 6.3.

All transformers consist of a series of L transformer blocks, acting on embedding vectors $\boldsymbol{h}_l \in \mathbb{R}^{T \times d_{\text{model}}}$ at layer l, where T is the maximum sequence length and d_{model} is the dimension of the embedding space. These embeddings $\boldsymbol{h}_l$ are trained to contain an increasingly rich contextual understanding of the tokens within the sequence. Each transformer block first applies the attention mechanism from Equation (2.30), followed by a fully connected neural network layer, outputting updated embeddings $\boldsymbol{h}_{l+1}$. A GPT model is inputted the sequence of token indexes in the vocabulary $\mathcal{V}$, as well as the token positions within the sequence. As both indexes and positions are integers, they must first be converted to appropriate vectors, which is achieved using embedding layers [179]. These two vectors are then summed together to form input embeddings $\boldsymbol{h}_0$. The output layer L of a GPT model uses the embeddings $\boldsymbol{h}_L^{(t_j)} \in \mathbb{R}^{d_{\text{model}}}$ of token t_j to predict the next token t_{j+1} in the sequence using a linear layer. These output embeddings $\boldsymbol{h}_L$ contain a deep contextual understanding of the sequence, which can be used as feature vectors in various other prediction tasks, besides the standard next token prediction. A pre-trained GPT model is typically then fine-tuned on a smaller, more domain-specific dataset, that may or may not contain additional labels $\boldsymbol{y}$. The output embeddings $\boldsymbol{h}_L$ are used to predict labels $\boldsymbol{y}$, allowing for conditional generation of sequences $\boldsymbol{x}$ for a desired $\boldsymbol{y}$.

Transformers have already being applied to materials [170,326,327] and molecular systems [33,322,323,325]. In particular, Bagal et al. [33] introduced MolGPT, a generative pre-trained transformer (GPT) [76,78] applied to datasets [13,89] of molecules in SMILES. The standard SMILES tokenization scheme [325] was employed in MolGPT, with a novel method of conditioning on molecular properties. The architecture of MolGPT is that of a standard GPT [76,78], see Section 2.5.2, besides this method

of property conditioning[3]. In this work, a similar MolGPT architecture is applied to molecules in the MolBlox representation instead of SMILES. In Section 6.4.2, it is proposed to fine-tune this model on a dataset which contains molecular properties derived from full 3D electronic structure calculations [90]. MolGPT uses a linear layer to convert the scalar molecular properties $\boldsymbol{y} \in \mathbb{R}^{n_y}$ into a suitable vector $\boldsymbol{p} \in \mathbb{R}^{d_{\text{model}}}$, for n_y distinct properties. Each sequence of tokens is then conditioned by concatenating this molecular vector $\boldsymbol{p}$ to the previous input embeddings $\boldsymbol{h}_0$. This conditioning process allows the model to associate molecules with their corresponding molecular properties, allowing for the selective generation of molecules by specifying some desired molecular properties.

6.4 Results

To verify the effectiveness of this conditional language-modelling framework, the generated molecules must be compared to the real molecules from the training dataset. If the generated molecules are indistinguishable to the real ones, then the model can be considered to have successfully learned the underlying probability distribution of the training dataset. Various means of comparing the generated and the real molecules can be used. Additionally, the accuracy of the conditioning on molecular properties must be determined. This is achieved using parity plots which compare the desired property values inputted into the model against the actual calculated values of the generated molecules. A number of different versions of MolGPT are trained in order to analyse the framework's efficacy in various different scenarios. For example, models can be trained with no property-conditioning, where no desired property values are used as model input. Alternatively, a model can be trained to require any number of different properties. The more properties included, the more control over the conditional generation process is obtained. However, a disadvantage of this is that the model will then require multiple inputs to be specified in order to function.

The training procedure for a MolGPT model is as follows. The default settings from the open-source implementation nanoGPT [178] are used unless otherwise stated. All training data is compiled into binary files to avoid loading the entire dataset into

[3]In original paper by Bagal et al. [33], they use two additional inputs: molecular properties and a Murcko scaffold [328]. In this work only the molecular properties are used, the scaffolds are neglected.

memory at once. All molecular properties are feature-scaled to a mean of zero and standard deviation of one, see Section 2.2. The embedding dimension d_{model} defaults to 128. The number of transformer blocks L and the number of transformer head n_{head} are both set to four. A maximum sequence length T of 100 and 120 are respectively used for MolBlox and SMILES. Mini-batch gradient descent [105] was performed using the Adam optimiser [106] with a learning rate of $6{\times}10^{-5}$, and an effective batch size of 160. A total of 400,000 updates were performed in pre-training, with a further 200,000 in fine-tuning. All models were trained using a Nvidia RTX 3060Ti GPU, the funding for which was provided by the Irish Research Council. After training, top-k sampling was used, as shown in Figure 6.3, with k=500 and a softmax temperature of one, see Equation (2.32).

6.4.1 Pre-Training with 2D Properties

The GuacaMol dataset [89] serves as a benchmark, allowing similar methods of generating SMILES [329–332] to be compared. This is implemented as a GuacaMol code repository [333], which provides standardised methods of evaluating model performance. There are five different metrics included in the GuacaMol benchmarks. These are validity, uniqueness, novelty, Kullback–Leibler (KL) divergence [334], and Fréchet ChemNet distance (FCD) [335], which are all expressed as a percentage. The validity score measures the percentage of generated molecules which have valid SMILES with the correct semantic structure. The uniqueness score measures how often the generated molecules are repeated, where a score of 100% signifies that there is no repetition. The novelty score measures how many of the generated molecules are already present within the training dataset, with low scores indicating overfitting. The KL divergence measures the distance between distributions of various physicochemical properties [89]. Lastly, the FCD score compares generated and training molecules using a neural network called ChemNet [335], which is trained to predict biological activities of drugs. The performances of MolGPT, trained on both MolBlox and SMILES, are compared in Table 6.1 to various other methods of generating SMILES [329–332]. It can be concluded that MolGPT performs well, using either MolBlox or SMILES. A slightly lower validity rate is shown by the MolBlox version, perhaps due to unresolved edge cases in the inversion method, but it displays the highest KL divergence score.

Model	Validity	Uniqueness	Novelty	KL Div.	FCD
DiGress [329]	85.2	**100**	**99.9**	92.9	68
DisCo [330]	86.6	86.6	86.5	92.6	59.7
Cometh [331]	98.9	98.9	97.6	96.7	72.7
DeFoG [332]	**99.0**	99.0	97.9	97.9	73.8
G2PT [336]	95.3	**100**	99.5	95.6	**92.7**
MolGPT + SMILES	90.3	99.9	96.5	98.4	89.7
MolGPT + MolBlox	84.0	99.9	95.2	**98.8**	89.5

Table 6.1: Table showing the GuacaMol dataset [89] benchmarking metrics [333] for our proposed models compared to various other similar methods.

The most straightforward method of analysing the generated molecules' quality is to visually inspect them. In Figure 6.4 it can be seen that the generated molecules look visually similar to real ones from the training dataset. It can also be seen that generated molecules from a property-conditioned model appear to be smaller in size and less diverse. This indicates that the model has successfully learned to associate property values with particular regions of chemical space, which can then be selectively sampled from at will.

Another method of verifying the realism of the generated molecules involves examining their various different property distributions. In Figure 6.5, the distributions of 2D properties such as logP and TPSA are shown. It is evident that the 2D properties of the generated molecules form similar distributions to the training set molecules, regardless of whether MolBlox or SMILES were used as the representation. Also shown in Figure 6.5 are molecule size and chemical composition distributions, which are both correctly aligned. However, the MolBlox generated molecules under-represent the very infrequent chemical species such as boron, silicon, and selenium. This is thought to occur due to MolBlox having a much larger vocabulary size than SMILES, with a higher proportion of tokens discarded during top-k sampling. The frequency of molecular blocks containing these improbable species can be increased by adjusting either k or the softmax temperature, see Equation (2.32).

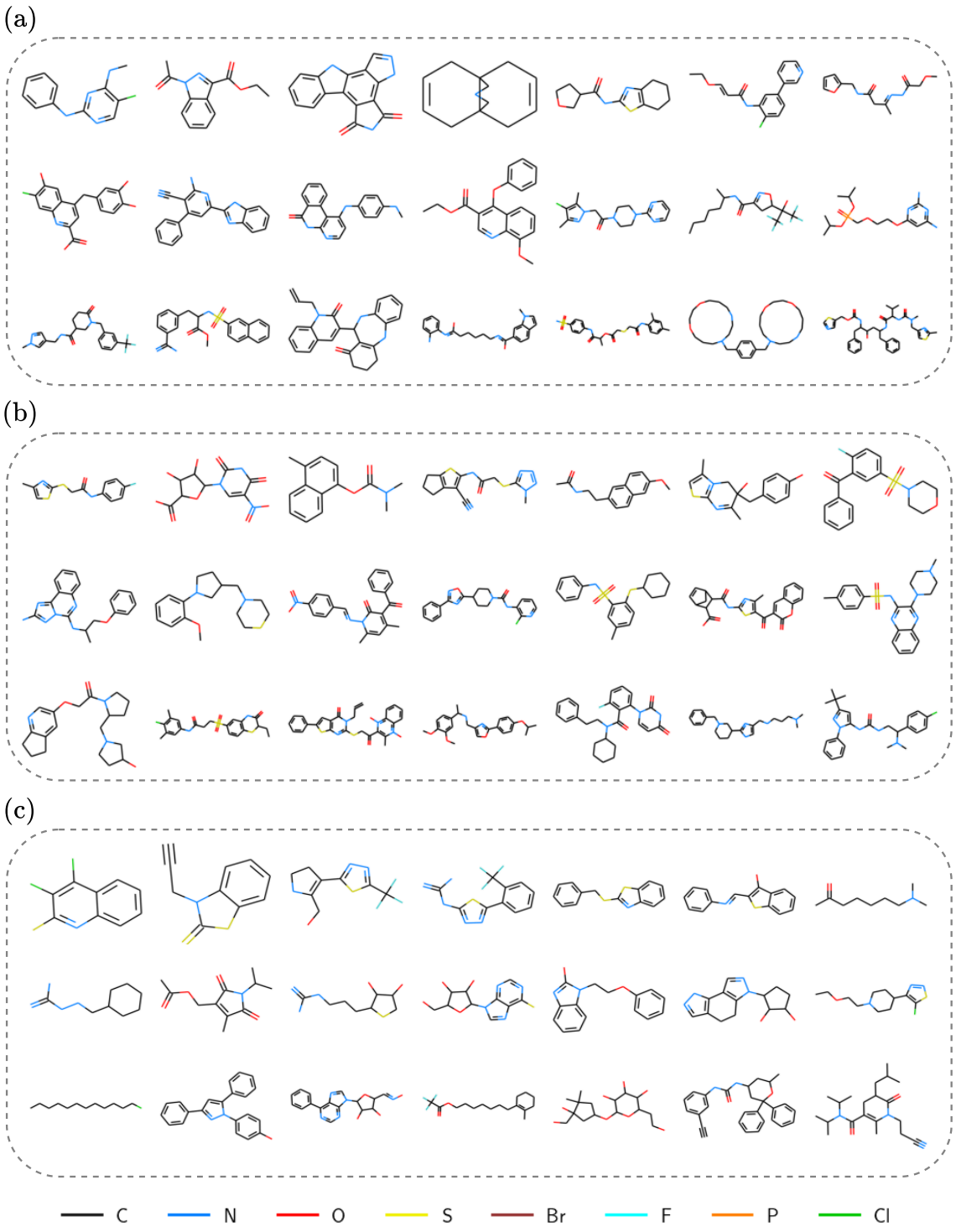

Figure 6.4: Diagram comparing real molecules from the GuacaMol dataset [89] to molecules generated using MolGPT with a MolBlox representation. Panel (a) shows real molecules randomly selected from this GuacaMol dataset. Panel (b) shows molecules generated with no property conditioning. Panel (c) shows property-conditioned generated molecules that aim to be similar to central-nervous system drugs [41, 337], with low polar surface area [38], medium hydrophobicity [37], and high druglikeness [40].

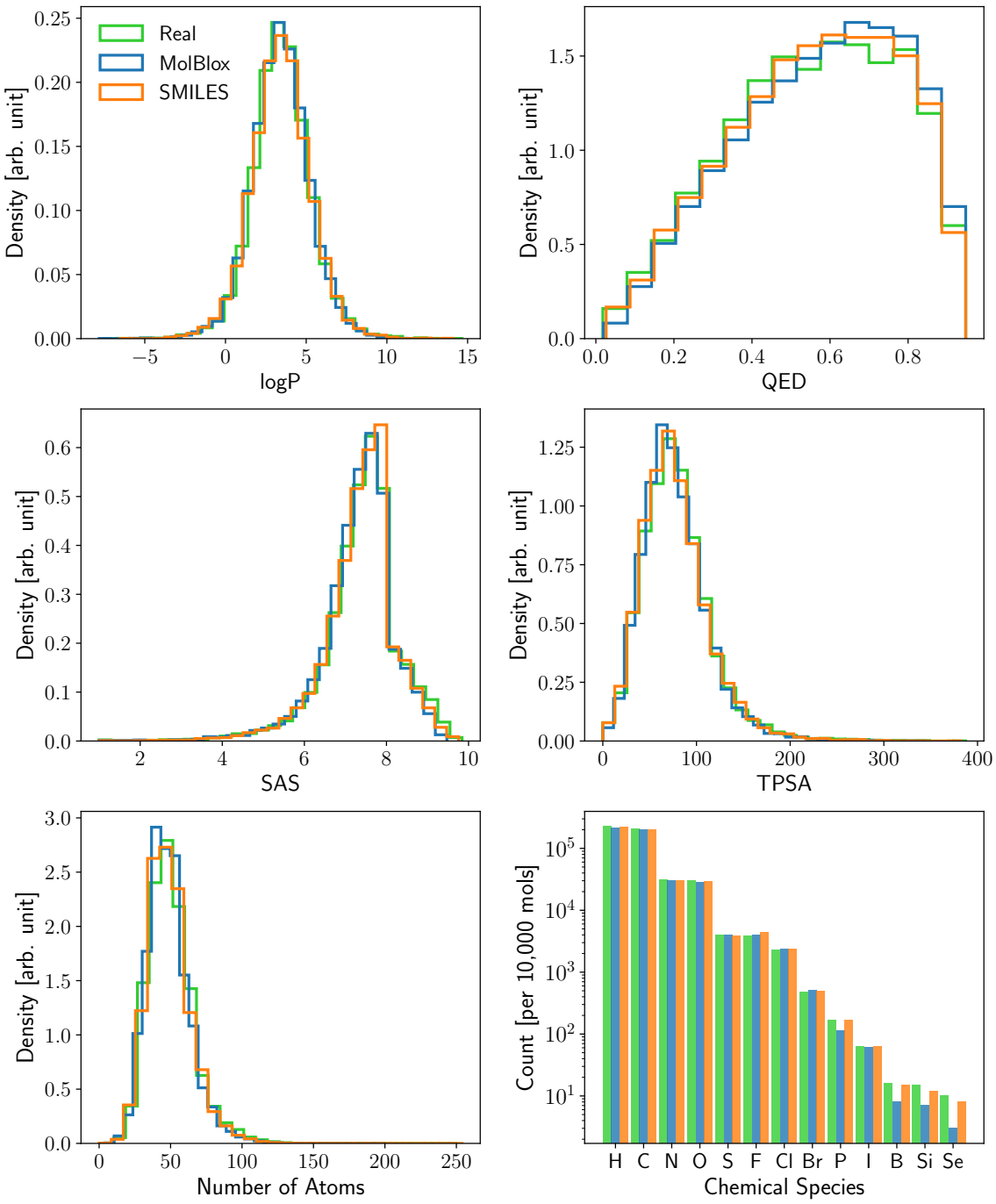

Figure 6.5: Figure analysing the properties, sizes, and compositions of molecules generated by MolGPT when pre-trained on the GuacaMol dataset [89]. Real GuacaMol molecules are compared against generated molecules from two different MolGPT models: one trained using MolBlox and another using SMILES. The top four plots show the distributions of different 2D properties [37–40] for molecules generated with no property-conditioning. These 2D properties are hydrophobicity (logP) [37], druglikeness (QED) [40], synthesisability (SAS) [39], and polar surface area (TPSA) [38]. The bottom left plot shows distributions of the total number of atoms. The bottom right plot shows the number of atoms of each chemical species per 10,000 molecules.

The efficacy of conditioning MolGPT on 2D properties is now analysed. Various different models are used, trained on both MolBlox and SMILES representations. Parity plots are employed to show the discrepancy between the desired property values inputted into the model, and the actual calculated values of the generated molecules. Conditioning on a singular property versus conditioning on multiple properties is also compared. In Figure 6.6, four different MolGPTs are trained, one for each 2D property. Random desired property values are uniformly sampled across the entire training set range, showing the accuracy of the conditioning in all regions. It is clear that MolBlox is equally accurate as SMILES, with little discernable difference between the two. The conditioning can broadly be considered accurate, particularly in the cases of logP and TPSA, with coefficients of determination R^2 [304] being close to one. However, it is clear that the conditioning on SAS fails for scores below eight. This is unexpected considering that a MolGPT with no property-conditioning previously produced the correct SAS distributions in Figure 6.5. These low values of SAS are more useful, as they signify easily-synthesisable molecules. It is not fully understood why the SAS scoring algorithm [39] fails for these generated molecules, more investigation is needed. Similarly, in Figure 6.7, MolGPT is conditioned using all four 2D properties, displaying slightly reduced accuracies as measured by R^2 values. In Figure 6.7, the desired properties are randomly selected properties of real molecules, which accounts for the reduced range as compared to Figure 6.6.

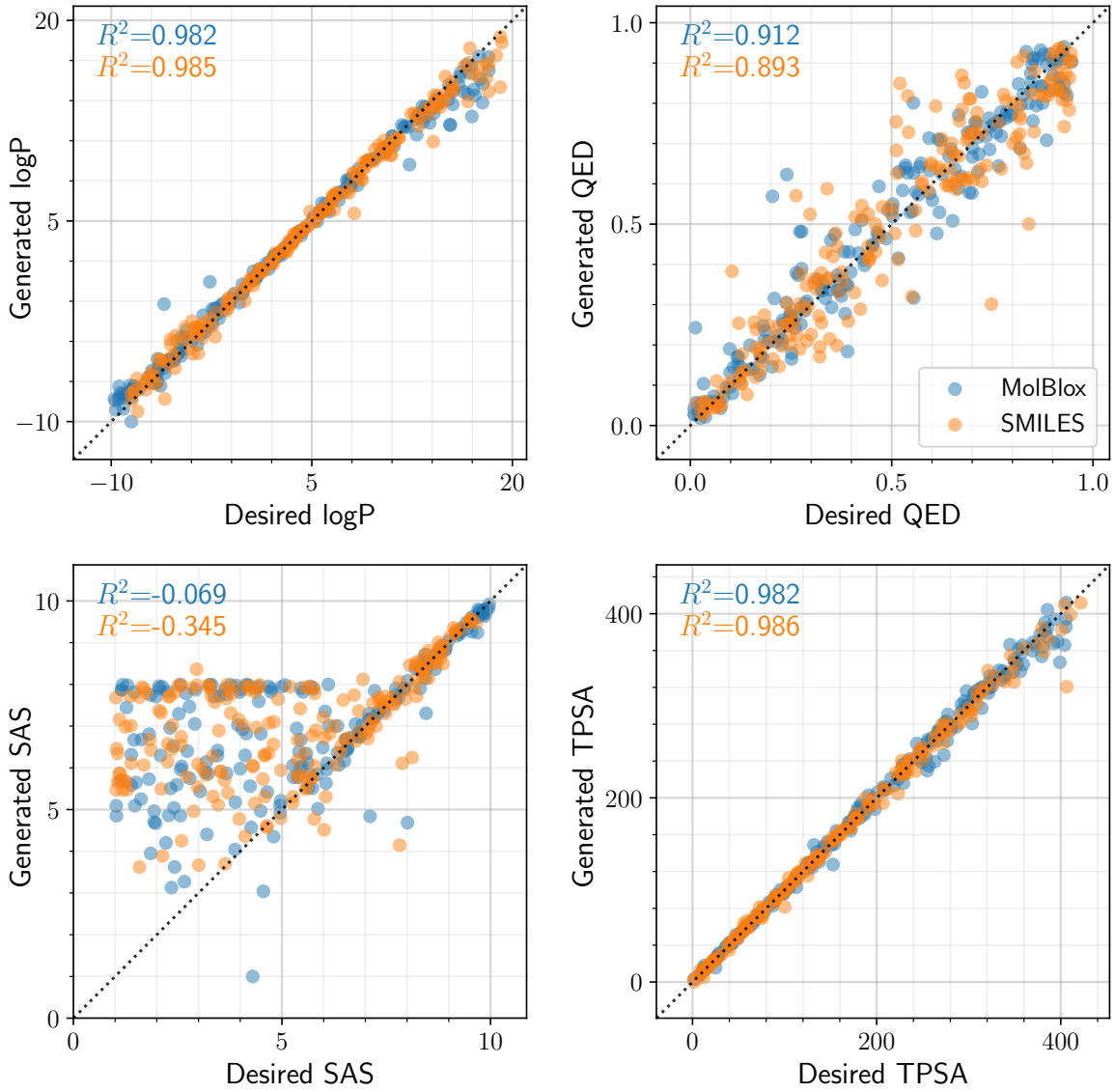

Figure 6.6: Figure showing parity plots for property-conditioning MolGPT on the GuacaMol dataset [89], with each 2D property being trained separately. These 2D properties are hydrophobicity (logP) [37], druglikeness (QED) [40], synthesisability (SAS) [39], and polar surface area (TPSA) [38]. Two different MolGPT models are compared: one trained using MolBlox and another using SMILES. The desired property values inputted into the model are compared against the actual calculated values of the generated molecules. The desired property values are randomly chosen between the maximum and minimum values occurring in the training dataset for each property.

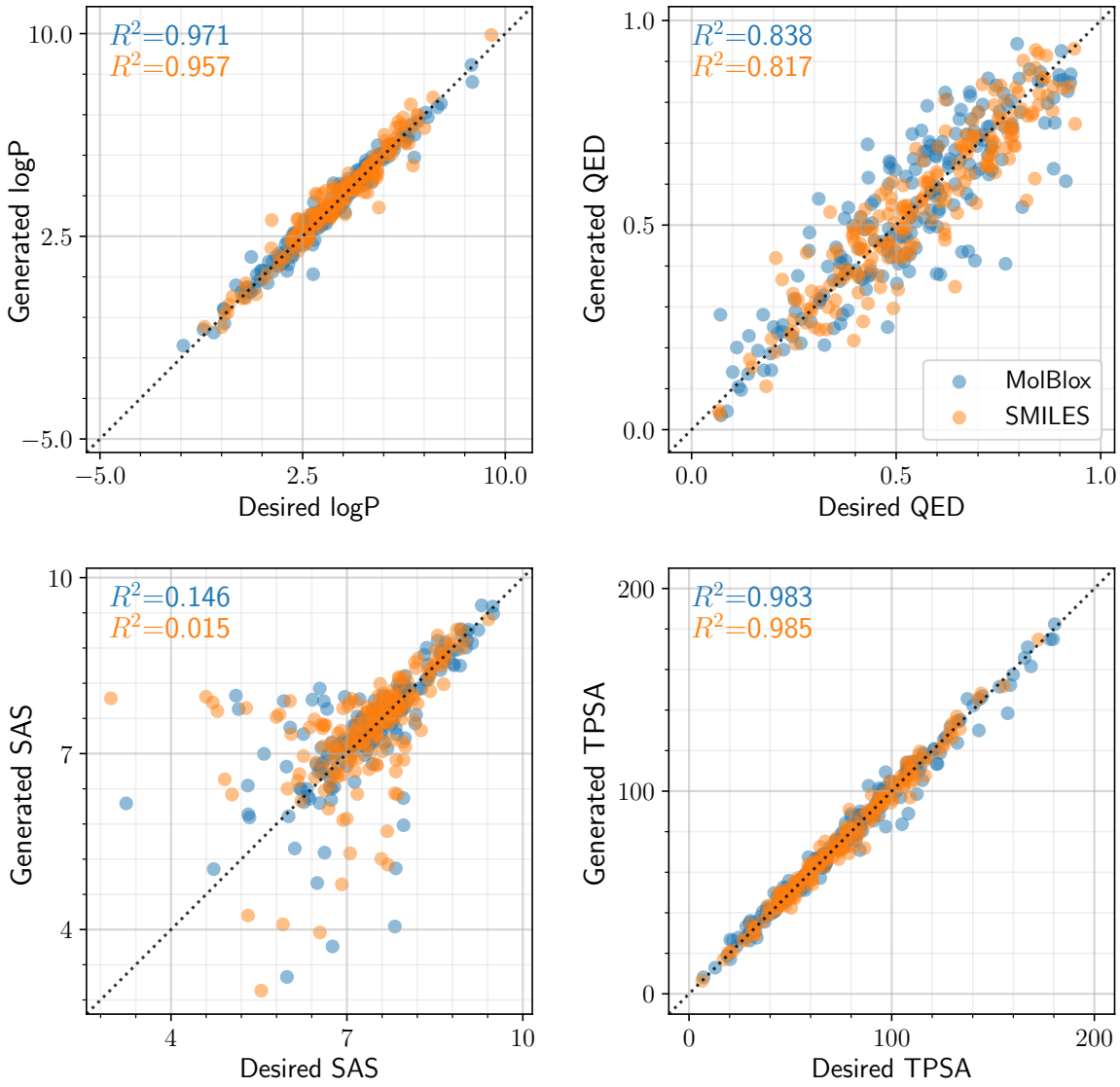

Figure 6.7: Figure showing parity plots for property-conditioning MolGPT on the GuacaMol dataset [89], with all four 2D properties being trained simultaneously. These 2D properties are hydrophobicity (logP) [37], druglikeness (QED) [40], synthesisability (SAS) [39], and polar surface area (TPSA) [38]. Two different MolGPT models are compared: one trained using MolBlox and another using SMILES. The desired property values inputted into the model are compared against the actual calculated values of the generated molecules. Property values of molecules randomly chosen from the training dataset are used as the desired values.

6.4.2 Fine-Tuning with DFT Properties

After finishing the pre-training on the GuacaMol dataset, it is then possible to fine-tune on a more domain-specific dataset. The QM9 dataset [14, 90] is an excellent candidate as it contains DFT calculated properties, which can be used to condition the model. Every molecule in this dataset has three components: SMILES, DFT properties, and molecular geometries. Only the SMILES and the DFT properties are used as inputs into the MolGPT model. The QM9 dataset uses the CORINA conformer generation method [324] to propose initial molecular geometries for SMILES, which are a subset of the large GDB17 database [14]. These initial geometries then undergo successive stages of geometry relaxations, using increasingly strict electronic and geometric convergence criteria, as implemented by the Gaussian 09 DFT code [338]. The properties of the optimised geometries are then calculated using the B3LYP functional [199–202] and a 6-31G(2df,p) basis set [320, 321], see Section 3.3.1.

To verify the accuracy of conditioning MolGPT on DFT properties, a similar procedure must be used to convert the generated SMILES into molecular geometries to calculate their DFT properties. However, due to QM9 using proprietary software packages, some open-source alternatives are employed in this work. The ETKDG conformer generation method [91, 92] is used instead of CORINA [324]. It employs experimental-torsion distance geometry (ETDG), with added aromatic knowledge (K), to assign atomic coordinates, with appropriate interatomic distances, from known atom connectivities as specified by SMILES. Experimental knowledge of torsions is used to rotate specific atoms until a realistic molecular geometry is obtained. This freely-available method exhibits comparable, if not improved, performance to those commercial packages such as CORINA [339].

A similar geometry optimisation procedure to that of QM9 is then used. The ETKDG molecular geometries are first relaxed using a force-field [340], followed by two successive stages of relaxation using the PySCF package [42–44]. The first stage uses a computationally inexpensive PBE functional [198] with a def2-SVP basis set [278]. In the second stage of relaxation, the final B3LYP functional [199–202] and 6-31G(2df,p) basis set [320, 321] are used. Other than the basis set and functional, the same DFT parameters and design decisions were used as in Section 4.4.2. It is expected that using a more similar geometry optimisation procedure to that of QM9 would allow

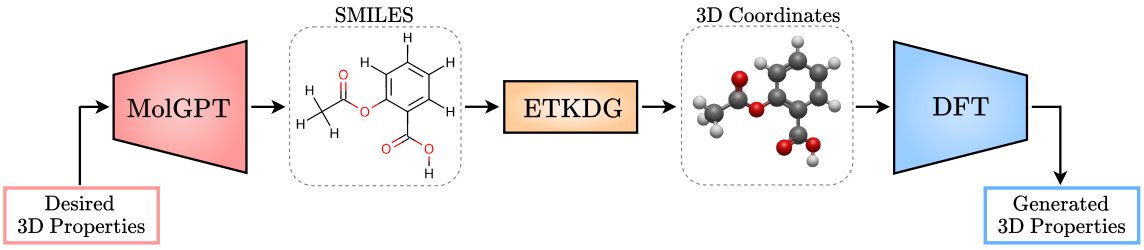

Figure 6.8: Diagram showing the workflow used to assess the accuracy of conditioning MolGPT on DFT properties from the QM9 dataset [14, 90]. For a given set of desired DFT properties, MolGPT outputs 2D molecules as SMILES or as MolBlox. The ETKDG conformer generation method [91, 92] converts a 2D SMILES into a 3D molecule. DFT calculations and geometry optimisations are then performed using PySCF [42–44]. The desired and the generated DFT properties can then be compared.

for the accuracy of the property-conditioning to be more accurately assessed.

The quality of the molecules generated by the fine-tuned MolGPT can be assessed as before, by visual inspection and by comparing real and generated property distributions. In Figure 6.9, real molecules from the QM9 dataset are displayed next to generated molecules. It can be seen that the generated molecules look visually similar. Also shown are some molecules generated by a version of MolGPT conditioned on both 2D properties and DFT properties, further demonstrating the versatility of the proposed framework. The distributions of molecule sizes are compared in Figure 6.10, with both MolBlox and SMILES versions showing a similar distribution to the real training set molecules. In Figure 6.10, the chemical composition of these fine-tuned, generated molecules is determined to closely align to the training set composition.

It is considered necessary to perform a brief ablation test, where components of the workflow are removed to better assess the performance of the property-conditioning. This ablation involves removing the generative model, and just comparing the DFT properties from the QM9 dataset to those obtained using PySCF on the ETKDG molecular geometries. The top row of plots in Figure 6.11 show the results of this ablation. It can be concluded that the EKTDG geometry optimisation procedure

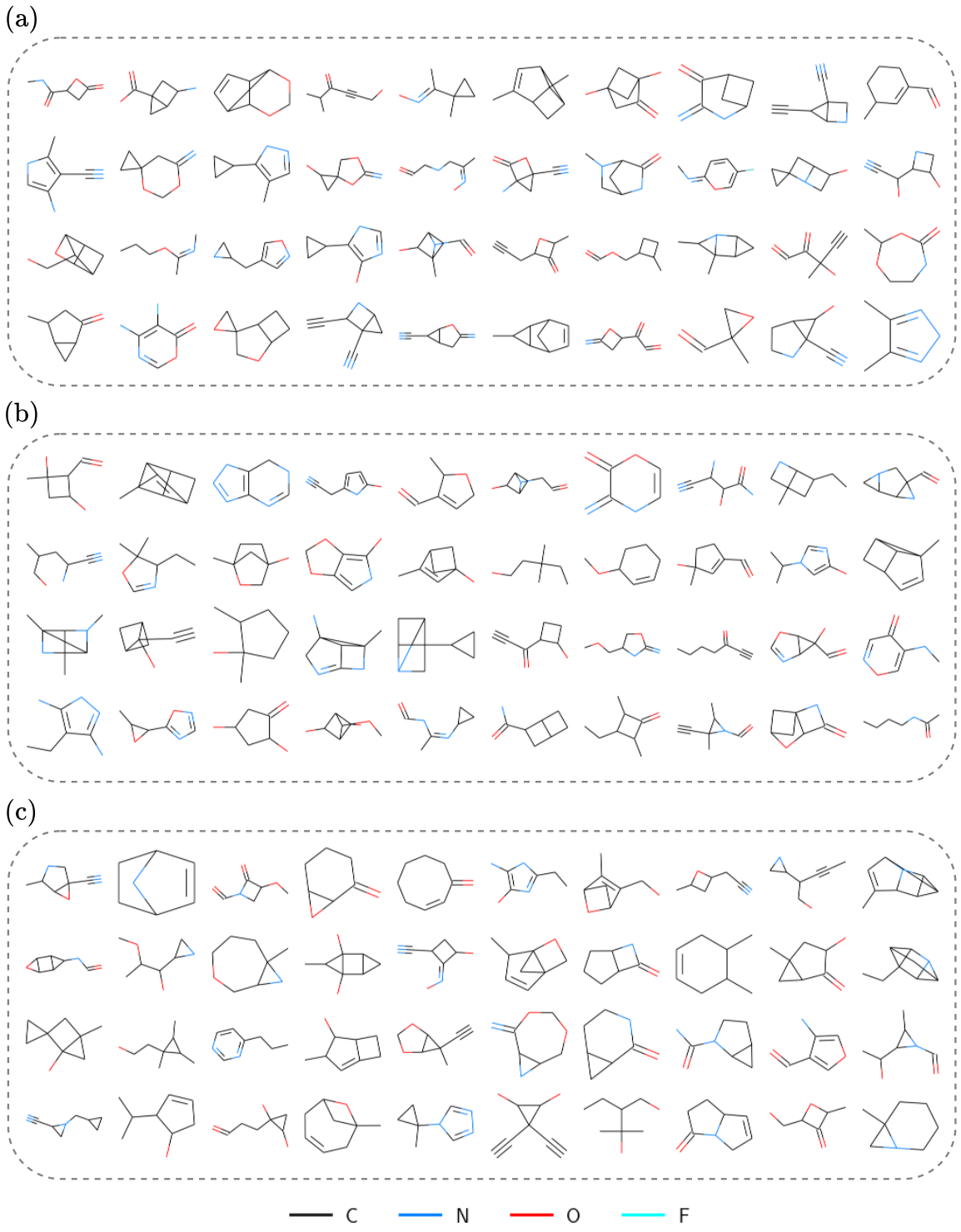

Figure 6.9: Diagram comparing real molecules from the QM9 dataset [14, 90] to molecules generated using MolGPT with a MolBlox representation. Panel (a) shows real molecules randomly selected from this QM9 dataset. Panel (b) shows molecules generated with no property conditioning. Panel (c) shows property-conditioned generated molecules that aim to be similar to central-nervous system drugs [41,337], with low polar surface area [38], medium hydrophobicity [37], high druglikeness [40], low DFT total energy, and medium HOMO-LUMO gap energy.

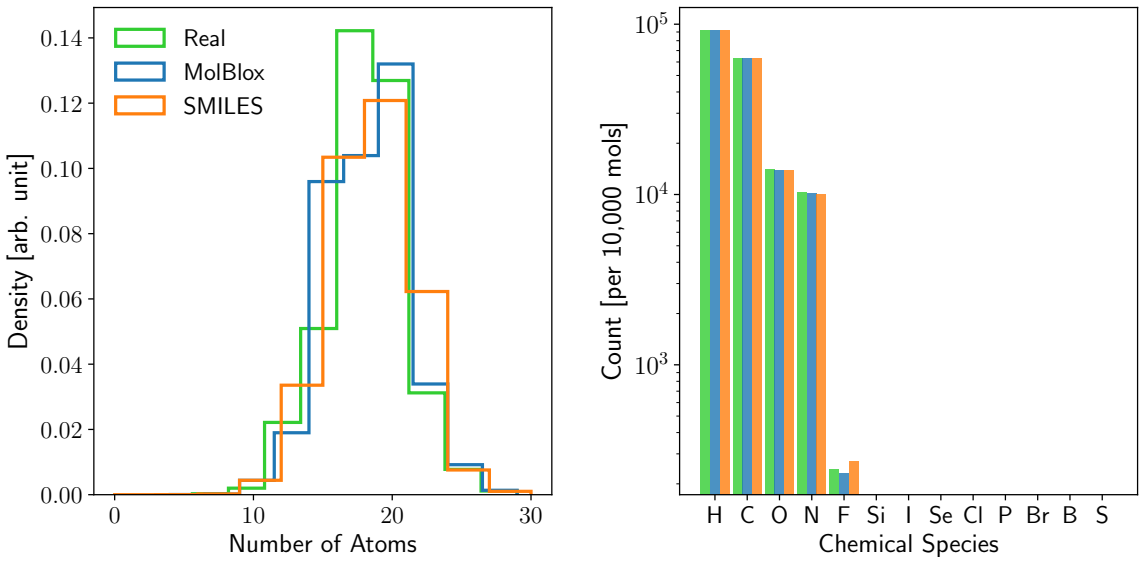

Figure 6.10: Figure analysing the size and composition of molecules generated by MolGPT when fine-tuned on the QM9 dataset [14,90]. Real QM9 molecules are compared against generated molecules from two different MolGPT models: one trained using MolBlox and another using SMILES. On the left distributions of the total number of atoms are shown. On the right is shown the number of atoms of each chemical species per 10,000 molecules.

introduces a large source of error into the workflow, particularly in the case of dipole moments. The difference is lower after optimisation but is still deemed significant[4]. This deviation acts as a lower limit on the accuracy of the property-conditioning verification process; even if the conditioning of MolGPT is perfect, the difference between the desired and the generated properties will still exhibit at least this level of inaccuracy. It is worth noting that only relates to the verification process, not the actual model. Therefore, it is expected that the generated properties would more closely align to the desired ones if the same conformer generation method and relaxation procedure was used as in QM9.

Finally, the accuracy of the conditioning of MolGPT on DFT properties is assessed. The workflow for doing this is shown in Figure 6.8. Parity plots are used to compare the desired DFT properties inputted into the model against the calculated properties of the generated molecules. This analysis is shown in Figure 6.11, for separate models trained on both MolBlox and SMILES. Three properties from the QM9 dataset are

[4]It is noted that PySCF [42–44] and Gaussian 09 [338] resulted in near identical calculated properties, when the same geometries and the same functional and basis set were used.

used to condition the model: total energies, HOMO-LUMO gap energies, and dipole moments [49]. It can be seen from the lower six plots in Figure 6.11 that the success of the property conditioning is heavily affected by the molecular geometries suggested by the ETKDG method. The property-conditioning seems more successful when MolBlox are used as the representation, motivating their continued use. This increased accuracy shown by the MolBlox representation could be due to the larger vocabulary size [341], which allows for more fine granularity in describing a molecule as a sequence.

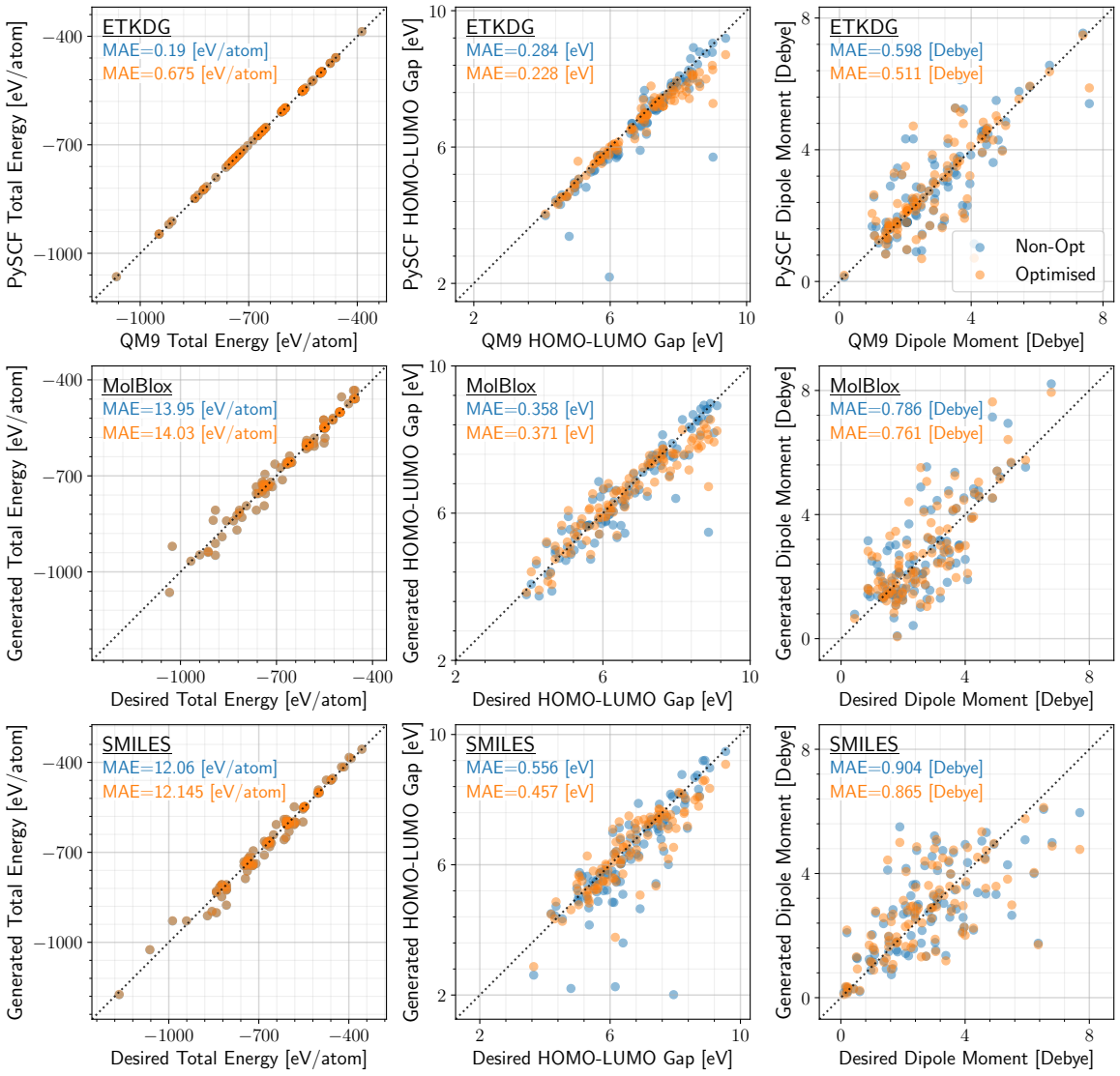

Figure 6.11: Figure showing the accuracy of conditioning MolGPT on total energies, HOMO-LUMO gaps, dipole moments from the QM9 dataset [14, 90]. The top row of plots serves as an ablation, showing the discrepancy between the QM9 dataset values and the PySCF [42–44] calculated values on structures obtained from the ETKDG conformation generation method [91, 92]. The middle row of plots shows the desired DFT values versus the actual DFT values of molecules generated by MolGPT with a MolBlox representation. The bottom row of plots similarly shows the accuracy of the property conditioning of MolGPT when trained on SMILES.

6.5 Summary

Motivated by the challenges encountered in Chapter 5, a language-modelling workflow was created, allowing a diverse range of molecules to be generated for selective values of certain properties. A text-based representation was adopted instead of the previous many-body representation. This facilitated the desired increase in diversity, at the cost of losing 3D structural information. A generative pre-trained transformer (GPT) [76,78] was employed as the generative-modelling architecture. The model was conditioned using some so-called 2D properties [37–40], which don't require any 3D structure to calculate. These include physicochemical properties such as hydrophobicity [37] and polar surface area [38].

A major contribution of this chapter was a proposed text-based chemical representation called MolBlox. This representation aims to be more physically inspired than the standard text-based representation of SMILES [72]. MolBlox decomposes a molecule into a sequence of textual molecular blocks. A method to invert these molecular blocks back into a whole molecule, with an interpretable SMILES, was developed. MolBlox are specifically designed for use in a generative language modelling scenario. An existing architecture called MolGPT was implemented [33], to adapt the standard GPT architecture for conditional property generation.

The large GuacaMol dataset [89] was used for pre-training and the QM9 dataset [14,90] was used for fine-tuning. The MolGPT architecture was applied to both MolBlox and SMILES representations, conditioned on various different combinations of properties. The efficacy of the workflow was initially analysed using the GuacaMol benchmarking statistics such as validity, uniqueness, and novelty rates. It was verified that the generated molecules had similar distributions of size, chemical species, and properties as the training set molecules. The property conditioning was deemed reasonably accurate, except for the synthetic accessibility scores (SAS) [39], which suffered from unexplained errors. MolGPT was then fine-tuned on the QM9 dataset, property conditioned on a number of DFT properties. These generated molecules were similarly validated. Considering the model has no knowledge of 3D structure, property conditioning on DFT total energies and HOMO-LUMO gaps was deemed to work surprisingly well. However, the model only showed a slight correlation between desired values and calculated values for the case of dipole moments.

Chapter 7

Conclusions

The objective of this thesis was to implement generative-modelling workflows that can output novel molecules. Particular importance was placed on conditional generation, where molecules with some set of desirable properties can be selectively generated. It can be concluded that this thesis introduces novel workflows with sufficient capabilities to have satisfied this objective to a reasonable degree. However, further testing and experimental verification is needed to fully validate the findings of this work. Each chapter in this thesis is concluded with a summary section, which the reader should view for more details on each chapter. Additionally, a more succinct overview of each chapter is provided at the end of Chapter 1.

Chapter 2 was the first of two methods chapters. It introduces the fundamentals of machine learning. A supervised learning scenario was described, where each sample $\boldsymbol{x}$ in the training dataset has a corresponding label $\boldsymbol{y}$. The workings of neural networks were then detailed, as they form the basis of most machine-learning algorithms. The majority of the chapter was spent introducing the two different generative-modelling architectures used throughout this work: generative adversarial networks (GANs) [77] and generative pre-trained transformers (GPTs) [76, 78].

Chapter 3 detailed the actual physical systems to which these generative models were applied. This thesis only considers molecular systems, but similar methods could theoretically be applied to periodic systems. Density-functional theory (DFT) was introduced as the preferred method of performing electronic structure calculations. Machine-learned interatomic potentials (MLIAPs), which predict quantum-mechanical (QM) properties at a fraction of the cost of full DFT calculations, were

then introduced. MLIAPs require a sophisticated chemical representation to describe interacting systems as feature vectors to accurately predict these QM properties. Many-body representations were introduced as an efficient means of describing such systems, particularly bispectrum components [75].

In Chapter 4, a method of transforming many-body representations back into Cartesian coordinates was proposed. This inverse transformation starts from a specified initial structure, which is iteratively updated to minimise a defined loss function. At the end of the procedure, the updated initial structure should have many-body descriptors equal to those which were to be inverted. The intended use case of the method was to allow a generative model to benefit from the accuracies of a many-body representation, while still being able to interpret the generated samples. A full global inversion not being guaranteed, due to the cut-off functions used in the many-body descriptors. The accuracy of the inversion method was shown to diminish with increasing system size and torsional degrees of freedom. Additionally, there are a number of undesirable characteristics of the inversion method. It is a stochastic, gradient-based method which can often fail to converge, falling into local minima. The topology of the surface of the loss function requires further examination. It is also an inherently sequential process, which cannot be parallelised, acting as a bottle-neck in a generative-modelling workflow. The requirement that some initial structure be specified drastically limits the method's potential use cases. It is hypothesised that the method would benefit from integrating other components into the workflow, to ease the amount of work the gradient-based optimisation needs to do. For example, some torsional structure search [342], with fixed interatomic distances. Despite its drawbacks, this local inversion method does facilitate the generation of realistic 3D configurations.

Chapter 5 applied the inversion method from Chapter 4 to its intended use case in a generative-modelling framework. A framework called GANdalf was further developed, using a generative adversarial network (GAN) to produce new 3D molecules directly in the format of a many-body descriptor. The inversion method was then used to convert these generated many-body descriptors into human-interpretable Cartesian coordinates. The generated molecules were verified to have similar distributions of bond distances and angles as the training set molecules. Single or multiple differ-

ent molecule types can be included in the training dataset. A property-conditioned version of GANdalf was trained to output molecules for specified DFT total energy values. As the GANdalf framework uses the many-body inversion method, it shares all the same problems and challenges. The biggest restriction of GANdalf is that it cannot extrapolate out to new molecules types. This is due to the inversion method requiring some initial structure from which to begin its iterative process. If GANdalf were to output novel molecule types, the inversion process would not know from where to begin. However, there are still scenarios where GANdalf may be useful. It can successfully learn the underlying probability distribution of a training dataset, allowing for selective generation of molecules in those hard-to-sample regions of a molecular dynamics trajectory.

One potential improvement to the GANdalf framework would entail having a generator capable of outputting samples directly in Cartesian coordinates, eliminating the need for the inversion method altogether. This was previously attempted, see Section 5.2, but it led to unrealistic outputs. It was later attempted to generate in 3D, while still retaining the accuracy of a sophisticated many-body representation within the discriminator, see Section 5.3.1. This led to significantly slower training times as these many-body descriptors needed to be calculated by an external library in the inner training loop of the GAN. This also interrupted the backpropagation of gradients, as it constitutes a non-differentiable layer in the model. It was attempted to manually implement these derivatives, allowing for proper backpropagation, but problems with feature scaling were encountered. However, it can be considered that the ideal version of GANdalf would have a generator outputting directly in Cartesian coordinates, and a discriminator learning from the accuracies of a many-body representation.

Another problem with GANdalf is that the property-conditioning is sometimes inaccurate. This is especially the case for larger molecules types such as aspirin, as shown in Figure 5.17. This discrepancy between the desired properties and the calculated properties is significantly more evident in the case of DFT calculated values. According to the internal structure-to-property regression network, the property conditioning is actually working as intended. The problem is therefore that this internal regression network is not sufficiently accurate as compared to DFT. It is hypothe-

sised that a more accurate regression model would improve the property conditioning. Perhaps some active learning scheme [343, 344] could be implemented to update the weights of this regression network throughout the GAN training. A different potential improvement would be to change to model architecture from a GAN to a diffusion model [28, 138]. This is a more modern architecture, typically considered more stable and versatile than GANs, with higher quality outputs. However, this would be a relatively minor change, keeping the same fundamental workflow.

In Chapter 6 a generative language model was used to produce molecules using a textual representation. This facilitated a larger degree of diversity as compared to GANdalf, at the cost of losing 3D structural information. A new text-based representation called MolBlox was proposed and compared to the existing SMILES representation [72]. A method to invert MolBlox back into the more interpretable SMILES representation was detailed. The standard generative pre-trained transformer (GPT) was modified into MolGPT [33], allowing for the conditioning on molecular properties. MolGPT was pre-trained on the GuacaMol dataset with 2D physicochemical properties [37–40]. It was then fine-tuned on the QM9 dataset [14, 90] with DFT properties. In both cases, the generated molecules were verified as being similar to those contained in the training set. The 2D property conditioning was deemed reasonably accurate, except for the synthetic accessibility score (SAS) [39], which suffered from unexplained errors. For the 3D property conditioning, only high or low values of dipole moments were able to be reliably specified. This was a much lower granularity than was possible for the total energies or the HOMO-LUMO gaps.

A major motivation for developing the MolBlox representation was to allow some symbolic notion of 3D structure to be incorporated. A generative language model could then output molecules directly in 3D, despite only using a text-based representation. This has not yet been successfully achieved, due to challenges with inverting such a representation. However, MolBlox can still theoretically be adapted to include 3D structure. One potential way would be to find the number of distinct geometry classes for unique token in the vocabulary. Each molecular block could then be modified like C{=C−H−H}X, where X would be a number symbolically signifying the geometry class. Such a scheme would result in generated molecules with known atom connectivities, and known local geometries for each molecular block. The problem

would then be to correctly align each molecular block with appropriate torsional angles. A similar method to the ETKDG conformer generation process [91, 92] could be used, where a large database of experimentally determined torsions are queried. Or perhaps a Bayesian optimisation [235, 261] scheme could be used to search across the space of all possible torsions until a suitably stable configuration is found [342].

A significant contribution of Chapter 6 was the fine-tuning on the QM9 dataset [14, 90], and the subsequent conditioning on DFT properties. This fine-tuning retains the benefits of a 2D representation such as versatility and efficiency, while also allowing the generation of molecules for selective values of DFT properties. To verify the accuracy of the conditioning on DFT properties, a similar workflow is used as in the construction of the QM9 dataset, as seen in Figure 6.8. However, this thesis uses different packages to those used in the QM9 dataset, ones that are open-source. The conformer generation tool of ETKDG [91, 92] is used instead of CORINA [324]. Additionally, DFT calculations and geometry optimisations are performed using PySCF [42–44] instead of Gaussian 09 [338]. These two differences make assessment of the property conditioning difficult, as seen in the row of plots in Figure 6.11. It is noted that PySCF is a similar DFT implementation to Gaussian 09, with near identical calculated property values when using the same B3LYP functional [199–202] and 6-31G(2df,p) basis set [320, 321]. An accurate assessment of the 3D property conditioning should adopt a conformer generation and geometry optimisation procedure more similar to that of QM9. Alternatively, an entirely new fine-tuning dataset could be constructed.

In conclusion, this thesis seeks to provide novel solutions to the problem of chemical design. Generative-modelling techniques, based on machine-learning methods, are proposed as an alternative paradigm to the traditional approach of chemical design. Particular importance is placed on accurate property conditioning of these generative models, allowing for the generation of molecules with desired values of specified properties. Two different molecular generation frameworks are proposed, one using sophisticated many-body descriptors and the other using a versatile text-based representation. Inversion methods are developed to interpret the generated molecules. These two frameworks are both considered successful examples of generative models applied to molecular systems.

Bibliography

[1] Reymond, J.-L., van Deursen, R., Blum, L. C. & Ruddigkeit, L. Chemical space as a source for new drugs. *Medicinal Chemistry Communications* **1**, 30–38 (2010).

[2] Ade, P. A. *et al.* Planck 2015 results-XIII. cosmological parameters. *Astronomy & Astrophysics* **594**, A13 (2016).

[3] Kim, S. *et al.* PubChem substance and compound databases. *Nucleic Acids Research* **44**, D1202–D1213 (2016).

[4] Zunger, A. Inverse design in search of materials with target functionalities. *Nature Reviews Chemistry* **2**, 0121 (2018).

[5] Scannell, J. W., Blanckley, A., Boldon, H. & Warrington, B. Diagnosing the decline in pharmaceutical R&D efficiency. *Nature Reviews Drug Discovery* **11**, 191–200 (2012).

[6] Kola, I. The state of innovation in drug development. *Clinical Pharmacology & Therapeutics* **83**, 227–230 (2008).

[7] Fernald, K. D., Förster, P. C., Claassen, E. & van de Burgwal, L. H. The pharmaceutical productivity gap–incremental decline in R&D efficiency despite transient improvements. *Drug Discovery Today* **29**, 104160 (2024).

[8] Drews, J. Drug discovery: a historical perspective. *Science* **287**, 1960–1964 (2000).

[9] Ringel, M. S., Scannell, J. W., Baedeker, M. & Schulze, U. Breaking Eroom's law. *Nature Reviews Drug Discovery* **19**, 833–834 (2020).

[10] Kim, S. *et al.* Pubchem 2023 update. *Nucleic Acids Research* **51**, D1373–D1380 (2023).

[11] Pence, H. E. & Williams, A. ChemSpider: An online chemical information resource. *Journal of Chemical Education* **87**, 1123–1124 (2010).

[12] Gaulton, A. *et al.* The ChEMBL database in 2017. *Nucleic Acids Research* **45**, D945–D954 (2017).

[13] Irwin, J. J. & Shoichet, B. K. Zinc - A free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling* **45**, 177–182 (2005).

[14] Ramakrishnan, R., Dral, P. O., Rupp, M. & von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data* **1**, 1–7 (2014).

[15] Shoichet, B. K. Virtual screening of chemical libraries. *Nature* **432**, 862–865 (2004).

[16] Lyne, P. D. Structure-based virtual screening: an overview. *Drug Discovery Today* **7**, 1047–1055 (2002).

[17] Lengauer, T., Lemmen, C., Rarey, M. & Zimmermann, M. Novel technologies for virtual screening. *Drug Discovery Today* **9**, 27–34 (2004).

[18] Scior, T. *et al.* Recognizing pitfalls in virtual screening: a critical review. *Journal of Chemical Information and Modeling* **52**, 867–881 (2012).

[19] Warr, W. A. Combinatorial chemistry and molecular diversity. An overview. *Journal of Chemical Information and Computer Sciences* **37**, 134–140 (1997).

[20] Pyzer-Knapp, E. O., Suh, C., Gómez-Bombarelli, R., Aguilera-Iparraguirre, J. & Aspuru-Guzik, A. What is high-throughput virtual screening? A perspective from organic materials discovery. *Annual Review of Materials Research* **45**, 195–216 (2015).

[21] Gómez-Bombarelli, R. *et al.* Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach. *Nature Materials* **15**, 1120–1127 (2016).

[22] Bajorath, J. Integration of virtual and high-throughput screening. *Nature Reviews Drug Discovery* **1**, 882–894 (2002).

[23] Gimeno, A. *et al.* The light and dark sides of virtual screening: what is there to know? *International Journal of Molecular Sciences* **20**, 1375 (2019).

[24] Klebe, G. Virtual ligand screening: strategies, perspectives and limitations. *Drug Discovery Today* **11**, 580–594 (2006).

[25] Zhu, H. Big data and artificial intelligence modeling for drug discovery. *Annual Review of Pharmacology and Toxicology* **60**, 573–589 (2020).

[26] Lusher, S. J., McGuire, R., van Schaik, R. C., Nicholson, C. D. & de Vlieg, J. Data-driven medicinal chemistry in the era of big data. *Drug Discovery Today* **19**, 859–868 (2014).

[27] Arjovsky, M., Chintala, S. & Bottou, L. Wasserstein GAN. *arXiv preprint arXiv:1701.07875 [stat.ML]* (2017).

[28] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N. & Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2256–2265 (Proceedings of Machine Learning Research, 2015).

[29] Vaswani, A. *et al.* Attention is all you need. *Advances in Neural Information Processing Systems* **30** (2017).

[30] Gómez-Bombarelli, R. *et al.* Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science* **4**, 268–276 (2018).

[31] Gromski, P. S., Henson, A. B., Granda, J. M. & Cronin, L. How to explore chemical space using algorithms and automation. *Nature Reviews Chemistry* **3**, 119–128 (2019).

[32] Mirza, M. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784 [cs.LG]* (2014).

[33] Bagal, V., Aggarwal, R., Vinod, P. & Priyakumar, U. D. MolGPT: molecular generation using a transformer-decoder model. *Journal of Chemical Information and Modeling* **62**, 2064–2076 (2021).

[34] Kadurin, A., Nikolenko, S., Khrabrov, K., Aliper, A. & Zhavoronkov, A. DruGAN: An advanced generative adversarial autoencoder model for de novo generation of new molecules with desired molecular properties in silico. *Molecular Pharmaceutics* **14**, 3098–3104 (2017).

[35] Zhou, Z., Kearnes, S., Li, L., Zare, R. N. & Riley, P. Optimization of molecules via deep reinforcement learning. *Scientific Reports* **9**, 10752 (2019).

[36] Lim, J., Ryu, S., Kim, J. W. & Kim, W. Y. Molecular generative model based on conditional variational autoencoder for de novo molecular design. *Journal of Cheminformatics* **10**, 1–9 (2018).

[37] Leo, A., Hansch, C. & Elkins, D. Partition coefficients and their uses. *Chemical Reviews* **71**, 525–616 (1971).

[38] Prasanna, S. & Doerksen, R. Topological polar surface area: a useful descriptor in 2D-QSAR. *Current Medicinal Chemistry* **16**, 21–41 (2009).

[39] Ertl, P. & Schuffenhauer, A. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics* **1**, 1–11 (2009).

[40] Bickerton, G. R., Paolini, G. V., Besnard, J., Muresan, S. & Hopkins, A. L. Quantifying the chemical beauty of drugs. *Nature Chemistry* **4**, 90–98 (2012).

[41] Shityakov, S., Neuhaus, W., Dandekar, T. & Förster, C. Analysing molecular polar surface descriptors to predict blood-brain barrier permeation. *International Journal of Computational Biology and Drug Design* **6**, 146–156 (2013).

[42] Sun, Q. *et al.* PySCF: the Python-based simulations of chemistry framework. *Wiley Interdisciplinary Reviews: Computational Molecular Science* **8**, e1340 (2018).

[43] Sun, Q. *et al.* Recent developments in the PySCF program package. *The Journal of Chemical Physics* **153** (2020).

[44] Sun, Q. Libcint: An efficient general integral library for Gaussian basis functions. *Journal of Computational Chemistry* **36**, 1664–1671 (2015).

[45] Yuan, Q., Santana-Bonilla, A., Zwijnenburg, M. A. & Jelfs, K. E. Molecular generation targeting desired electronic properties via deep generative models. *Nanoscale* **12**, 6744–6758 (2020).

[46] Jørgensen, P. B. *et al.* Machine learning-based screening of complex molecules for polymer solar cells. *The Journal of Chemical Physics* **148** (2018).

[47] Pang, C., Qiao, J., Zeng, X., Zou, Q. & Wei, L. Deep generative models in de novo drug molecule generation. *Journal of Chemical Information and Modeling* **64**, 2174–2194 (2023).

[48] Westermayr, J., Gilkes, J., Barrett, R. & Maurer, R. J. High-throughput property-driven generative design of functional organic molecules. *Nature Computational Science* **3**, 139–148 (2023).

[49] Martin, R. M. *Electronic Structure: Basic Theory and Practical Methods* (Cambridge University Press, 2004).

[50] Parr, R. G., Gadre, S. R. & Bartolotti, L. J. Local density functional theory of atoms and molecules. *Proceedings of the National Academy of Sciences* **76**, 2522–2526 (1979).

[51] Mishra, A. & Bäuerle, P. Small molecule organic semiconductors on the move: promises for future solar energy technology. *Angewandte Chemie International Edition* **51**, 2020–2067 (2012).

[52] Bonate, P. L. & Howard, D. R. *Pharmacokinetics in Drug Development: Regulatory and Development Paradigms*, vol. 2 (Springer Science & Business Media, 2005).

[53] Di, L. & Kerns, E. H. *Drug-like Properties: Concepts, Structure Design and Methods* (Academic press, 2015).

[54] Chenthamarakshan, V. *et al.* CogMol: Target-specific and selective drug design for COVID-19 using deep generative models. *Advances in Neural Information Processing Systems* **33**, 4320–4332 (2020).

[55] Feng, H., Wang, R., Zhan, C.-G. & Wei, G.-W. Multiobjective molecular optimization for opioid use disorder treatment using generative network complex. *Journal of Medicinal Chemistry* **66**, 12479–12498 (2023).

[56] Martinelli, D. D. Generative machine learning for de novo drug discovery: A systematic review. *Computers in Biology and Medicine* **145**, 105403 (2022).

[57] Kola, I. & Landis, J. Can the pharmaceutical industry reduce attrition rates? *Nature Reviews Drug Discovery* **3**, 711–716 (2004).

[58] Merlot, C. Computational toxicology - a tool for early safety evaluation. *Drug Discovery Today* **15**, 16–22 (2010).

[59] Wermuth, C. G. *The Practice of Medicinal Chemistry* (Academic Press, 2011).

[60] Li, Y., Zhang, L. & Liu, Z. Multi-objective de novo drug design with conditional graph generative model. *Journal of Cheminformatics* **10**, 1–24 (2018).

[61] Ragoza, M., Masuda, T. & Koes, D. R. Generating 3D molecules conditional on receptor binding sites with deep generative models. *Chemical Science* **13**, 2701–2713 (2022).

[62] Yang, L. *et al.* Transformer-based generative model accelerating the development of novel BRAF inhibitors. *ACS Omega* **6**, 33864–33873 (2021).

[63] Li, Y. *et al.* Generative deep learning enables the discovery of a potent and selective ripk1 inhibitor. *Nature Communications* **13**, 6891 (2022).

[64] Yoshimori, A. *et al.* Design and synthesis of DDR1 inhibitors with a desired pharmacophore using deep generative models. *ChemMedChem* **16**, 955–958 (2021).

[65] Korshunova, M. *et al.* Generative and reinforcement learning approaches for the automated de novo design of bioactive compounds. *Communications Chemistry* **5**, 129 (2022).

[66] Gilson, M. K., Given, J. A., Bush, B. L. & McCammon, J. A. The statistical-thermodynamic basis for computation of binding affinities: a critical review. *Biophysical Journal* **72**, 1047–1069 (1997).

[67] Ward, R. A. & Goldberg, F. W. *Kinase Drug Discovery: Modern Approaches* (Royal Society of Chemistry, 2018).

[68] Ertl, P. Polar surface area. In *Molecular Drug Properties: Measurement and Prediction*, 111–126 (Wiley Online Library, 2007).

[69] Connolly, M. L. Computation of molecular volume. *Journal of the American Chemical Society* **107**, 1118–1124 (1985).

[70] Morris, G. M. & Lim-Wilby, M. Molecular docking. In *Molecular Modeling of Proteins*, 365–382 (Humana Press, 2008).

[71] Meng, X.-Y., Zhang, H.-X., Mezei, M. & Cui, M. Molecular docking: a powerful approach for structure-based drug discovery. *Current Computer-Aided Drug Design* **7**, 146–157 (2011).

[72] Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences* **28**, 31–36 (1988).

[73] Krenn, M., Häse, F., Nigam, A., Friederich, P. & Aspuru-Guzik, A. Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. *Machine Learning: Science and Technology* **1**, 045024 (2020).

[74] Kuzminykh, D. *et al.* 3D molecular representations based on the wave transform for convolutional neural networks. *Molecular Pharmaceutics* **15**, 4378–4385 (2018).

[75] Bartók, A., Payne, M., Kondor, R. & Csányi, G. Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons. *Physical Review Letters* **104**, 136403 (2010).

[76] Radford, A. *et al.* Language models are unsupervised multitask learners. *OpenAI Blog* **1**, 9 (2019).

[77] Goodfellow, I. *et al.* Generative adversarial nets. *Advances in Neural Information Processing Systems* **27** (2014).

[78] Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. Improving language understanding by generative pre-training. *OpenAI Blog* (2018).

[79] Griffiths, D. J. & Schroeter, D. F. *Introduction to Quantum Mechanics* (Cambridge University Press, 2019).

[80] Hohenberg, P. & Kohn, W. Inhomogeneous electron gas. *Physical Review* **136**, B864 (1964).

[81] Kohn, W. & Sham, L. J. Self-consistent equations including exchange and correlation effects. *Physical Review* **140**, A1133 (1965).

[82] Kohn, W. Density functional and density matrix method scaling linearly with the number of atoms. *Physical Review Letters* **76**, 3168 (1996).

[83] Behler, J. & Parrinello, M. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Physical Review Letters* **98**, 146401 (2007).

[84] Thompson, A., Swiler, L., Trott, C., Foiles, S. & Tucker, G. Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials. *Journal of Computational Physics* **285**, 316–330 (2015).

[85] Zuo, Y. *et al.* Performance and cost assessment of machine learning interatomic potentials. *The Journal of Physical Chemistry A* **124**, 731–745 (2020).

[86] Cobelli, M., Cahalane, P. & Sanvito, S. Local inversion of the chemical environment representations. *Physical Review B* **106**, 035402 (2022).

[87] Christensen, A. S. & von Lilienfeld, A. Revised MD17 Dataset (2020). URL `https://doi.org/10.6084/m9.figshare.12672038.v3`. (Accessed: 08/01/2025).

[88] Liu, Z., Rodrigues, S. P. & Cai, W. Simulating the Ising model with a deep convolutional generative adversarial network. *arXiv preprint arXiv:1710.04987 [cond-mat.dis-nn]* (2017).

[89] Brown, N., Fiscato, M., Segler, M. H. & Vaucher, A. C. GuacaMol: Benchmarking models for de novo molecular design. *Journal of Chemical Information and Modeling* **59**, 1096–1108 (2019).

[90] Ruddigkeit, L., Van Deursen, R., Blum, L. C. & Reymond, J.-L. Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. *Journal of Chemical Information and Modeling* **52**, 2864–2875 (2012).

[91] Riniker, S. & Landrum, G. A. Better informed distance geometry: Using what we know to improve conformation generation. *Journal of Chemical Information and Modeling* **55**, 2562–2574 (2015).

[92] Wang, S., Witek, J., Landrum, G. A. & Riniker, S. Improving conformer generation for small rings and macrocycles based on distance geometry and experimental torsional-angle preferences. *Journal of Chemical Information and Modeling* **60**, 2044–2058 (2020).

[93] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* **3**, 210–229 (1959).

[94] OpenAI. ChatGPT: Language model for dialogue (2023). URL `https://chat.openai.com/`. (Accessed: 08/01/2025).

[95] Mitchell, T. M. *Machine Learning*, vol. 1 (McGraw-Hill, 1997).

[96] Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).

[97] Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929 [cs.CV]* (2020).

[98] Russakovsky, O. *et al.* Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* **115**, 211–252 (2015).

[99] Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* **25** (2012).

[100] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* **65**, 386 (1958).

[101] Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986).

[102] Montavon, G., Orr, G. & Müller, K.-R. *Neural Networks: Tricks of the Trade* (springer, 2012).

[103] Hoerl, A. E. & Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics* **12**, 55–67 (1970).

[104] Ng, A. Y. Feature selection, L1 vs. L2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning*, 78 (2004).

[105] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747 [cs.LG]* (2016).

[106] Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980 [cs.LG]* (2014).

[107] PyTorch. RMSprop. URL `https://pytorch.org/docs/stable/generated/torch.optim.RMSprop.html`. (Accessed: 08/01/2025).

[108] Duchi, J., Hazan, E. & Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* **12**, 2121–2159 (2011).

[109] LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**, 2278–2324 (1998).

[110] Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT: 19th International Conference on Computational Statistics*, 177–186 (Springer, 2010).

[111] McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* **5**, 115–133 (1943).

[112] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* **15**, 1929–1958 (2014).

[113] Hinton, G. E. & Zemel, R. Autoencoders, minimum description length and Helmholtz free energy. *Advances in Neural Information Processing Systems* **6** (1993).

[114] Ackley, D. H., Hinton, G. E. & Sejnowski, T. J. A learning algorithm for Boltzmann machines. *Cognitive Science* **9**, 147–169 (1985).

[115] Salakhutdinov, R. & Hinton, G. Deep Boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, vol. 5, 448–455 (Proceedings of Machine Learning Research, 2009).

[116] Hinton, G. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580 [cs.NE]* (2012).

[117] Ba, J. L., Kiros, J. R. & Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450 [stat.ML]* (2016).

[118] Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition. *Biometrika* **71**, 6 (1986).

[119] Nair, V. & Hinton, G. E. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 807–814 (2010).

[120] Nickolls, J., Buck, I., Garland, M. & Skadron, K. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *ACM Queue* **6**, 40–53 (2008).

[121] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* **2**, 303–314 (1989).

[122] Amodei, D. *et al.* Deep Speech 2: End-to-end speech recognition in English and Mandarin. In *International Conference on Machine Learning*, 173–182 (Proceedings of Machine Learning Research, 2016).

[123] Silver, D. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).

[124] Hendrycks, D. & Gimpel, K. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415 [cs.LG]* (2016).

[125] Maas, A. L., Hannun, A. Y., Ng, A. Y. *et al.* Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning*, vol. 28, 3 (2013).

[126] Bishop, C. M. *Neural Networks for Pattern Recognition* (Oxford University Press, 1995).

[127] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J. & Seung, H. S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* **405**, 947–951 (2000).

[128] Bridle, J. S. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, Architectures and Applications*, 227–236 (Springer, 1990).

[129] Dan, Y. *et al.* Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials. *npj Computational Materials* **6**, 84 (2020).

[130] Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal* **27**, 379–423 (1948).

[131] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics* **21**, 1087–1092 (1953).

[132] Markov, A. Extension of the limit theorems of probability theory to a sum of variables connected in a chain. *Dynamical Probabilistic Systems* **1**, 552 (1971).

[133] Bengio, Y. *et al.* Markovian models for sequential data. *Neural Computing Surveys* **2**, 129–162 (1999).

[134] Radford, A., Metz, L. & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434 [cs.LG]* (2016).

[135] Karras, T. Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196 [cs.NE]* (2017).

[136] Dinh, L., Krueger, D. & Bengio, Y. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516 [cs.LG]* (2014).

[137] Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S. & Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research* **22**, 1–64 (2021).

[138] Ho, J., Jain, A. & Abbeel, P. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems* **33**, 6840–6851 (2020).

[139] Odena, A., Olah, C. & Shlens, J. Conditional image synthesis with auxiliary classifier GANs. In *International Conference on Machine Learning*, 2642–2651 (Proceedings of Machine Learning Research, 2017).

[140] Yu, L., Zhang, W., Wang, J. & Yu, Y. SeqGAN: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31 (2017).

[141] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I. & Frey, B. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644 [cs.LG]* (2015).

[142] Karras, T., Laine, S. & Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4401–4410 (2019).

[143] Kong, J., Kim, J. & Bae, J. HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in Neural Information Processing Systems* **33**, 17022–17033 (2020).

[144] Brown, T. B. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165 [cs.CL]* (2020).

[145] Devlin, J. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805 [cs.CL]* (2018).

[146] Von Neumann, J. & Morgenstern, O. *Theory of Games and Economic Behavior* (Princeton University Press, 2007).

[147] LeCun, Y. *et al.* Backpropagation applied to handwritten zip code recognition. *Neural Computation* **1**, 541–551 (1989).

[148] Brock, A. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096 [cs.LG]* (2018).

[149] Lin, J. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory* **37**, 145–151 (1991).

[150] Arjovsky, M. & Bottou, L. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862 [stat.ML]* (2017).

[151] Villani, C. The Wasserstein Distances. In *Optimal Transport: Old and New*, 93–111 (Springer, 2009).

[152] Rudin, W. *Principles of Mathematical Analysis* (McGraw-Hill, 1976).

[153] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. & Courville, A. Improved training of Wasserstein GANs. *arXiv preprint arXiv:1704.00028 [cs.LG]* (2017).

[154] Sutskever, I. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215 [cs.CL]* (2014).

[155] Song, J., Zhang, J., Gao, L., Liu, X. & Shen, H. T. Dual conditional GANs for face aging and rejuvenation. In *International Joint Conferences on Artificial Intelligence*, 899–905 (2018).

[156] Xu, D., Yuan, S., Zhang, L. & Wu, X. Fairgan: Fairness-aware generative adversarial networks. In *International Conference on Big Data*, 570–575 (IEEE, 2018).

[157] Chen, X. *et al.* Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in Neural Information Processing Systems* **29** (2016).

[158] Yoon, J., Drumright, L. N. & van der Schaar, M. Anonymization through data synthesis using generative adversarial networks (ADS-GAN). *Journal of Biomedical and Health Informatics* **24**, 2378–2388 (2020).

[159] Yoon, J., Jarrett, D. & Van der Schaar, M. Time-series generative adversarial networks. *Advances in Neural Information Processing Systems* **32** (2019).

[160] Méndez-Lucio, O., Baillif, B., Clevert, D.-A., Rouquié, D. & Wichard, J. De novo generation of hit-like molecules from gene expression signatures using artificial intelligence. *Nature Communications* **11**, 10 (2020).

[161] He, Y.-L., Li, X.-Y., Ma, J.-H., Lu, S. & Zhu, Q.-X. A novel virtual sample generation method based on a modified conditional Wasserstein GAN to address the small sample size problem in soft sensing. *Journal of Process Control* **113**, 18–28 (2022).

[162] Cipra, B. A. An introduction to the Ising model. *The American Mathematical Monthly* **94**, 937–959 (1987).

[163] Onsager, L. Crystal statistics. I. A two-dimensional model with an order-disorder transition. *Physical Review* **65**, 117 (1944).

[164] Liu, Y. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692 [cs.CL]* (2019).

[165] Sanh, V. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108 [cs.CL]* (2019).

[166] Banar, B. & Colton, S. A systematic evaluation of GPT-2-based music generation. In *International Conference on Computational Intelligence in Music, Sound, Art and Design*, 19–35 (Springer, 2022).

[167] Lewis, M. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461 [cs.CL]* (2019).

[168] Raffel, C. *et al.* Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research* **21**, 1–67 (2020).

[169] Sun, C., Qiu, X., Xu, Y. & Huang, X. How to fine-tune BERT for text classification? In *Chinese Computational Linguistics*, 194–206 (Springer, 2019).

[170] Gilligan, L. P. J., Cobelli, M., Sayeed, H. M., Sparks, T. D. & Sanvito, S. Sampling latent material-property information from LLM-derived embedding representations. *arXiv preprint arXiv:2409.11971 [cs.CL]* (2024).

[171] Achiam, J. *et al.* GPT-4 technical report. *arXiv preprint arXiv:2303.08774 [cs.CL]* (2024).

[172] Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural Networks* **2**, 359–366 (1989).

[173] Jelinek, F. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice* (1980).

[174] Bengio, Y., Ducharme, R. & Vincent, P. A neural probabilistic language model. *Advances in Neural Information Processing Systems* **13** (2000).

[175] Chen, M. *et al.* Generative pretraining from pixels. In *International Conference on Machine Learning*, 1691–1703 (Proceedings of Machine Learning Research, 2020).

[176] Gage, P. A new algorithm for data compression. *The C Users Journal* **12**, 23–38 (1994).

[177] Sennrich, R., Haddow, B. & Birch, A. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, vol. 1, 1715–1725 (2016).

[178] Karpathy, A. NanoGPT. URL `https://github.com/karpathy/nanoGPT`. (Accessed: 08/01/2025).

[179] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems* **26** (2013).

[180] Child, R., Gray, S., Radford, A. & Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509 [cs.LG]* (2019).

[181] Beltagy, I., Peters, M. E. & Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150 [cs.CL]* (2020).

[182] He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016).

[183] Lowerre, B. P. & Reddy, B. R. Harpy, a connected speech recognition system. *The Journal of the Acoustical Society of America* **59**, S97–S97 (1976).

[184] Fan, A., Lewis, M. & Dauphin, Y. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833 [cs.CL]* (2018).

[185] Holtzman, A., Buys, J., Du, L., Forbes, M. & Choi, Y. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751 [cs.CL]* (2019).

[186] Pan, J. Scaling up system size in materials simulation. *Nature Computational Science* **1**, 95–95 (2021).

[187] Čížek, J. On the correlation problem in atomic and molecular systems. calculation of wavefunction components in Ursell-type expansion using quantum-field theoretical methods. *The Journal of Chemical Physics* **45**, 4256–4266 (1966).

[188] Bartlett, R. J. & Musiał, M. Coupled-cluster theory in quantum chemistry. *Reviews of Modern Physics* **79**, 291–352 (2007).

[189] Schrödinger, E. An undulatory theory of the mechanics of atoms and molecules. *Physical Review* **28**, 1049–1070 (1926).

[190] Born, M. & Heisenberg, W. Zur quantentheorie der molekeln. In *Original Scientific Papers Wissenschaftliche Originalarbeiten*, 216–246 (Springer, 1985).

[191] Parr, R. G. & Yang, W. Density-functional theory of the electronic structure of molecules. *Annual Review of Physical Chemistry* **46**, 701–728 (1995).

[192] Bernstein, N., Csányi, G. & Deringer, V. L. De novo exploration and self-guided learning of potential-energy surfaces. *npj Computational Materials* **5**, 99 (2019).

[193] Colò, G. Nuclear density functional theory. *Advances in Physics: X* **5**, 1740061 (2020).

[194] Allen, M. P. & Tildesley, D. J. *Computer Simulation of Liquids* (Oxford University Press, 2017).

[195] Seidl, A., Görling, A., Vogl, P., Majewski, J. A. & Levy, M. Generalized Kohn-Sham schemes and the band-gap problem. *Physical Review B* **53**, 3764 (1996).

[196] Lehtola, S., Blockhuys, F. & Van Alsenoy, C. An overview of self-consistent field calculations within finite basis sets. *Molecules* **25**, 1218 (2020).

[197] Perdew, J. P. & Schmidt, K. Jacob's ladder of density functional approximations for the exchange-correlation energy. In *AIP Conference Proceedings*, vol. 577, 1–20 (American Institute of Physics, 2001).

[198] Perdew, J. P., Burke, K. & Ernzerhof, M. Generalized gradient approximation made simple. *Physical Review Letters* **77**, 3865 (1996).

[199] Becke, A. D. Density-functional exchange-energy approximation with correct asymptotic behavior. *Physical Review A* **38**, 3098 (1988).

[200] Lee, C., Yang, W. & Parr, R. G. Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density. *Physical Review B* **37**, 785 (1988).

[201] Becke, A. D. Density-functional thermochemistry. III. The role of exact exchange. *The Journal of Chemical Physics* **98**, 5648–5652 (1993).

[202] Stephens, P. J., Devlin, F. J., Chabalowski, C. F. & Frisch, M. J. Ab initio calculation of vibrational absorption and circular dichroism spectra using density functional force fields. *The Journal of Physical Chemistry* **98**, 11623–11627 (1994).

[203] Perdew, J. P. & Zunger, A. Self-interaction correction to density-functional approximations for many-electron systems. *Physical Review B* **23**, 5048 (1981).

[204] Hartree, D. R. The wave mechanics of an atom with a non-Coulomb central field. Part II. Some results and discussion. In *Mathematical Proceedings of the*

Cambridge Philosophical Society, vol. 24, 111–132 (Cambridge University Press, 1928).

[205] Fiolhais, C., Nogueira, F. & Marques, M. A. *A Primer in Density Functional Theory*, vol. 620 (Springer, 2008).

[206] Hoerl, A. E. & Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics* **12**, 55–67 (1970).

[207] Drautz, R. Atomic cluster expansion for accurate and transferable interatomic potentials. *Physical Review B* **99**, 014104 (2019).

[208] Domina, M., Patil, U., Cobelli, M. & Sanvito, S. Cluster expansion constructed over Jacobi-Legendre polynomials for accurate force fields. *Physical Review B* **108**, 094102 (2023).

[209] Nelson, J. & Sanvito, S. Predicting the curie temperature of ferromagnets using machine learning. *Physical Review Materials* **3**, 104405 (2019).

[210] Scott, D., Coveney, P., Kilner, J., Rossiny, J. & Alford, N. M. N. Prediction of the functional properties of ceramic materials from composition using artificial neural networks. *Journal of the European Ceramic Society* **27**, 4425–4435 (2007).

[211] Legrain, F., Carrete, J., van Roekeghem, A., Curtarolo, S. & Mingo, N. How chemical composition alone can predict vibrational free energies and entropies of solids. *Chemistry of Materials* **29**, 6220–6227 (2017).

[212] Heller, S. R., McNaught, A., Pletnev, I., Stein, S. & Tchekhovskoi, D. InChI, the IUPAC international chemical identifier. *Journal of Cheminformatics* **7**, 1–34 (2015).

[213] Duvenaud, D. K. *et al.* Convolutional networks on graphs for learning molecular fingerprints. *Advances in Neural Information Processing Systems* **28** (2015).

[214] Putin, E. *et al.* Adversarial threshold neural computer for molecular de novo design. *Molecular Pharmaceutics* **15**, 4386–4397 (2018).

[215] Gupta, R. *et al.* Artificial intelligence to deep learning: machine intelligence approach for drug discovery. *Molecular Diversity* **25**, 1315–1360 (2021).

[216] Popova, M., Isayev, O. & Tropsha, A. Deep reinforcement learning for de novo drug design. *Science Advances* **4**, eaap7885 (2018).

[217] Sanchez-Lengeling, B. & Aspuru-Guzik, A. Inverse molecular design using machine learning: Generative models for matter engineering. *Science* **361**, 360–365 (2018).

[218] Guimaraes, G. L., Sanchez-Lengeling, B., Outeiral, C., Farias, P. L. C. & Aspuru-Guzik, A. Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models. *arXiv preprint arXiv:1705.10843 [stat.ML]* (2018).

[219] Dan, Y. *et al.* Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials. *Computational Materials* **6** (2020).

[220] Faber, F. A. *et al.* Prediction errors of molecular machine learning models lower than hybrid DFT error. *Journal of Chemical Theory and Computation* **13**, 5255–5264 (2017).

[221] Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R. & Tkatchenko, A. Quantum-chemical insights from deep tensor neural networks. *Nature Communications* **8**, 13890 (2017).

[222] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 1263–1272 (Proceedings of Machine Learning Research, 2017).

[223] Rupp, M., Tkatchenko, A., Müller, K.-R. & von Lilienfeld, O. A. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Letters* **108**, 058301 (2012).

[224] Kim, S., Noh, J., Gu, G. H., Aspuru-Guzik, A. & Jung, Y. Generative adversarial networks for crystal structure prediction. *ACS Central Science* **6**, 1412–1420 (2020).

[225] Fung, V. *et al.* Atomic structure generation from reconstructing structural fingerprints. *Machine Learning: Science and Technology* **3**, 045018 (2022).

[226] Luo, S., Guan, J., Ma, J. & Peng, J. A 3D generative model for structure-based drug design. *Advances in Neural Information Processing Systems* **34**, 6229–6239 (2021).

[227] Li, Y., Pei, J. & Lai, L. Structure-based de novo drug design using 3D deep generative models. *Chemical Science* **12**, 13664–13675 (2021).

[228] Nesterov, V., Wieser, M. & Roth, V. 3DMolNet: a generative network for molecular structures. *arXiv preprint arXiv:2010.06477 [q-bio.BM]* (2020).

[229] Gebauer, N., Gastegger, M. & Schütt, K. Symmetry-adapted generation of 3D point sets for the targeted discovery of molecules. *Advances in Neural Information Processing Systems* **32** (2019).

[230] Bartók, A., Payne, M., Kondor, R. & Csányi, G. On representing chemical environments. *Physical Review B* **87**, 184115 (2013).

[231] Shapeev, A. V. Moment tensor potentials: A class of systematically improvable interatomic potentials. *Multiscale Modeling & Simulation* **14**, 1153–1173 (2016).

[232] Behler, J. Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *The Journal of Chemical Physics* **134**, 074106 (2011).

[233] Pozdnyakov, S. N. *et al.* Incompleteness of atomic structure representations. *Physical Review Letters* **125**, 166001 (2020).

[234] Bergstra, J. & Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* **13**, 281–305 (2012).

[235] Frazier, P. I. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811 [stat.ML]* (2018).

[236] Behler, J. First principles neural network potentials for reactive simulations of large molecular and condensed systems. *Angewandte Chemie International Edition* **56**, 12828–12840 (2017).

[237] Smith, J. S., Isayev, O. & Roitberg, A. E. ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost. *Chemical Science* **8**, 3192–3203 (2017).

[238] Thompson, A. P. *et al.* LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications* **271**, 108171 (2022).

[239] Kohlmeyer, A. *et al.* LAMMPS molecular dynamics simulator. URL `https://www.lammps.org/`. (Accessed: 08/01/2025).

[240] Varshalovich, D. A., Moskalev, A. N., & Khersonskii, V. K. *Quantum Theory of Angular Momentum* (World Scientific, 1988).

[241] Wood, M. A. & Thompson, A. P. Extending the accuracy of the snap interatomic potential form. *The Journal of Chemical Physics* **148**, 241721 (2018).

[242] Batatia, I., Kovacs, D. P., Simm, G., Ortner, C. & Csányi, G. MACE: Higher order equivariant message passing neural networks for fast and accurate force fields. *Advances in Neural Information Processing Systems* **35**, 11423–11436 (2022).

[243] Lysogorskiy, Y. *et al.* Performant implementation of the atomic cluster expansion (PACE) and application to copper and silicon. *npj Computational Materials* **7**, 97 (2021).

[244] Bochkarev, A. *et al.* Efficient parametrization of the atomic cluster expansion. *Physical Review Materials* **6**, 013804 (2022).

[245] Qamar, M., Mrovec, M., Lysogorskiy, Y., Bochkarev, A. & Drautz, R. Atomic cluster expansion for quantum-accurate large-scale simulations of carbon. *Journal of Chemical Theory and Computation* **19**, 5151–5167 (2023).

[246] Lennard-Jones, J. E. Cohesion. *Proceedings of the Physical Society* **43**, 461 (1931).

[247] Morse, P. M. Diatomic molecules according to the wave mechanics. II. vibrational levels. *Physical Review* **34**, 57 (1929).

[248] Tersoff, J. Modeling solid-state chemistry: Interatomic potentials for multicomponent systems. *Physical Review B* **39**, 5566 (1989).

[249] Stillinger, F. H. & Weber, T. A. Computer simulation of local order in condensed phases of silicon. *Physical Review B* **31**, 5262 (1985).

[250] Daw, M. S. & Baskes, M. I. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Physical Review B* **29**, 6443 (1984).

[251] Van Duin, A. C., Dasgupta, S., Lorant, F. & Goddard, W. A. ReaxFF: a reactive force field for hydrocarbons. *The Journal of Physical Chemistry A* **105**, 9396–9409 (2001).

[252] Vanommeslaeghe, K. *et al.* CHARMM general force field: A force field for drug-like molecules compatible with the CHARMM all-atom additive biological force fields. *Journal of Computational Chemistry* **31**, 671–690 (2010).

[253] Wang, J., Wolf, R. M., Caldwell, J. W., Kollman, P. A. & Case, D. A. Development and testing of a general Amber force field. *Journal of Computational Chemistry* **25**, 1157–1174 (2004).

[254] Li, X.-G. *et al.* Quantum-accurate spectral neighbor analysis potential models for ni-mo binary alloys and fcc metals. *Physical Review B* **98**, 094104 (2018).

[255] Chen, C. *et al.* Accurate force field for molybdenum by machine learning large materials data. *Physical Review Materials* **1**, 043603 (2017).

[256] Hastie, T., Tibshirani, R. & Friedman, J. *The Elements of Statistical Learning: Data mining, Inference, and Prediction* (Springer, 2009).

[257] Frenkel, D. & Smit, B. *Understanding Molecular Simulation: From Algorithms to Applications* (Elsevier, 2023).

[258] Himanen, L. *et al.* DScribe: Library of descriptors for machine learning in materials science. *Computer Physics Communications* **247**, 106949 (2020).

[259] Laakso, J. *et al.* Updates to the DScribe library: New descriptors and derivatives. *The Journal of Chemical Physics* **158**, 234802 (2023).

[260] Bäck, T. & Schwefel, H. P. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation* **1**, 1–23 (1993).

[261] Snoek, J., Larochelle, H. & Adams, R. P. Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems* **25** (2012).

[262] Gelatt, C. Optimization by simulated annealing. *Science* **200**, 671 (1983).

[263] Gao, X., Ramezanghorbani, F., Isayev, O., Smith, J. S. & Roitberg, A. E. Torchani: A free and open source PyTorch-based deep learning implementation of the ANI neural network potentials. *Journal of Chemical Information and Modeling* **60**, 3408–3415 (2020).

[264] Dick, T., Wong, E. & Dann, C. How many random restarts are enough (2014). URL `https://www.cs.cmu.edu/~epxing/Class/10715/project-reports/DannDickWong.pdf`.

[265] Hanwell, M. D. *et al.* Avogadro: an advanced semantic chemical editor, visualization, and analysis platform. *Journal of Cheminformatics* **4**, 1–17 (2012).

[266] Ali, S. *et al.* Avogadro: an open-source molecular builder and visualization tool. URL `http://avogadro.cc/`. (Accessed: 08/01/2025).

[267] Nakamura, T. *et al.* Selecting molecules with diverse structures and properties by maximizing submodular functions of descriptors learned with graph neural networks. *Scientific Reports* **12**, 1124 (2022).

[268] Holliday, J. D., Ranade, S. S. & Willett, P. A fast algorithm for selecting sets of dissimilar molecules from large chemical databases. *Quantitative Structure-Activity Relationships* **14**, 501–506 (1995).

[269] Garcia-Hernandez, C., Fernandez, A. & Serratosa, F. Ligand-based virtual screening using graph edit distance as molecular similarity measure. *Journal of Chemical Information and Modeling* **59**, 1410–1421 (2019).

[270] Öztürk, H., Ozkirimli, E. & Özgür, A. A comparative study of SMILES-based compound similarity functions for drug-target interaction prediction. *BMC Bioinformatics* **17**, 1–11 (2016).

[271] Cao, Y., Jiang, T. & Girke, T. A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics* **24**, i366–i374 (2008).

[272] Fukutani, T., Miyazawa, K., Iwata, S. & Satoh, H. G-RMSD: root mean square deviation based method for three-dimensional molecular similarity determination. *Bulletin of the Chemical Society of Japan* **94**, 655–665 (2021).

[273] Lajiness, M. S. Molecular similarity-based methods for selecting compounds for screening. In *Computational Chemical Graph Theory*, 299–316 (Nova Science Publishers, Inc., 1990).

[274] Chmiela, S. *et al.* Machine learning of accurate energy-conserving molecular force fields. *Science Advances* **3(5)**, e1603015 (2017).

[275] Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K.-R. & Tkatchenko, A. Quantum-chemical insights from deep tensor neural networks. *Nature Communications* **8**, 13890 (2017).

[276] Chmiela, S., Sauceda, H. E., Müller, K.-R. & Tkatchenko, A. Towards exact molecular dynamics simulations with machine-learned force fields. *Nature Communications* **9**, 3887 (2018).

[277] Tkatchenko, A. & Scheffler, M. Accurate molecular Van der Waals interactions from ground-state electron density and free-atom reference data. *Physical Review Letters* **102**, 073005 (2009).

[278] Weigend, F. & Ahlrichs, R. Balanced basis sets of split valence, triple zeta valence and quadruple zeta valence quality for H to Rn: Design and assessment of accuracy. *Physical Chemistry Chemical Physics* **7**, 3297–3305 (2005).

[279] Kromann, J. C. *et al.* Calculate root-mean-square deviation (RMSD) of two molecules using rotation. URL `http://github.com/charnley/rmsd`. (Accessed: 08/01/2025).

[280] Kabsch, W. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A* **32**, 922–923 (1976).

[281] Landrum, G. *et al.* Rdkit: Open-source cheminformatics. URL `https://www.rdkit.org`. (Accessed: 08/01/2025).

[282] van Beek, B. Auto-FOX: Automated forcefield optimization extension. URL `https://github.com/nlesc-nano/auto-FOX`. (Accessed: 08/01/2025).

[283] Swamidass, S. *et al.* Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics* **21**, i359–i368 (2005).

[284] Neese, F. The ORCA program system. *Wiley Interdisciplinary Reviews: Computational Molecular Science* **2**, 73–78 (2012).

[285] Neese, F. Software update: The ORCA program system - Version 5.0. *Wiley Interdisciplinary Reviews: Computational Molecular Science* **12**, e1606 (2022).

[286] Young, D. *Computational Chemistry: A Practical Guide for Applying Techniques to Real World Problems* (John Wiley & Sons, 2004).

[287] Durrant, J. D. & McCammon, J. A. Molecular dynamics simulations and drug discovery. *BMC Biology* **9**, 1–9 (2011).

[288] Ladd, M. F. C., Palmer, R. A. & Palmer, R. A. *Structure Determination by X-Ray Crystallography*, vol. 233 (Springer, 1977).

[289] Hore, P. J. *Nuclear Magnetic Resonance* (Oxford University Press, 2015).

[290] Tersoff, J. & Hamann, D. R. Theory of the scanning tunneling microscope. *Physical Review B* **31**, 805 (1985).

[291] Marx, D. & Hutter, J. Ab initio molecular dynamics: Theory and implementation. In *Modern Methods and Algorithms of Quantum Chemistry*, vol. 1, 141 (John von Neumann Institute for Computing, 2000).

[292] Szabo, A. & Ostlund, N. S. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory* (Courier Corporation, 1996).

[293] Nash Jr, J. F. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences* **36**, 48–49 (1950).

[294] Nash, J. F. Non-cooperative games. In *The Foundations of Price Theory Vol 4*, 329–340 (Routledge, 2024).

[295] Paszke, A. *et al.* PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* **32** (2019).

[296] Larsen, A. H. *et al.* The atomic simulation environment - a Python library for working with atoms. *Journal of Physics: Condensed Matter* **29**, 273002 (2017).

[297] Bahn, S. R. & Jacobsen, K. W. An object-oriented scripting interface to a legacy electronic structure code. *Computing in Science & Engineering* **4**, 56–66 (2002).

[298] Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011).

[299] Virtanen, P. *et al.* SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods* **17**, 261–272 (2020).

[300] Montero de Hijes, P., Dellago, C., Jinnouchi, R., Schmiedmayer, B. & Kresse, G. Comparing machine learning potentials for water: Kernel-based regression and Behler-Parrinello neural networks. *The Journal of Chemical Physics* **160**, 114107 (2024).

[301] Behler, J., Martoňák, R., Donadio, D. & Parrinello, M. Metadynamics simulations of the high-pressure phases of silicon employing a high-dimensional neural network potential. *Physical Review Letters* **100**, 185501 (2008).

[302] Eshet, H., Khaliullin, R. Z., Kühne, T. D., Behler, J. & Parrinello, M. Ab initio quality neural-network potential for sodium. *Physical Review B - Condensed Matter and Materials Physics* **81**, 184107 (2010).

[303] Zhang, Y. *et al.* Neural information retrieval: A literature review. *arXiv preprint arXiv:1611.06792 [cs.IR]* (2016).

[304] Pearson, K. LIII. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **2**, 559–572 (1901).

[305] Davis, R. A., Lii, K.-S. & Politis, D. N. Remarks on some nonparametric estimates of a density function. In *Selected Works of Murray Rosenblatt*, 95–100 (Springer, 2011).

[306] Parzen, E. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics* **33**, 1065–1076 (1962).

[307] Scott, D. W. *Multivariate Density Estimation: Theory, Practice, and Visualization* (John Wiley & Sons, 2015).

[308] Devroye, L. Nonuniform random variate generation. *Handbooks in Operations Research and Management Science* **13**, 83–121 (2006).

[309] Chen, Y.-C. A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology* **1**, 161–187 (2017).

[310] Hotelling, H. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology* **24**, 417 (1933).

[311] Ikebata, H., Hongo, K., Isomura, T., Maezono, R. & Yoshida, R. Bayesian molecular design with a chemical language model. *Journal of Computer-Aided Molecular Design* **31** (2017).

[312] Ertl, P., Lewis, R., Martin, E. & Polyakov, V. In silico generation of novel, drug-like chemical matter using the LSTM neural network. *arXiv preprint arXiv:1712.07449 [cs.LG]* (2018).

[313] Olivecrona, M., Blaschke, T., Engkvist, O. & Chen, H. Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics* **9**, 1–14 (2017).

[314] Medina-Franco, J. L. & Saldívar-González, F. I. Cheminformatics to characterize pharmacologically active natural products. *Biomolecules* **10**, 1566 (2020).

[315] Wolf, T. *et al.* HuggingFace's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771 [cs.CL]* (2020).

[316] Moi, A. & Patry, N. HuggingFace's tokenizers. URL `https://github.com/huggingface/tokenizers`. (Accessed: 17/01/2025).

[317] Ahn, S., Kim, J., Lee, H. & Shin, J. Guiding deep molecular optimization with genetic exploration. *Advances in Neural Information Processing Systems* **33**, 12008–12021 (2020).

[318] Wang, C., Ong, H. H., Chiba, S. & Rajapakse, J. C. De novo molecule generation with graph latent diffusion model. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2121–2125 (2024).

[319] Rogers, D. & Hahn, M. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling* **50**, 742–754 (2010).

[320] Hariharan, P. C. & Pople, J. A. The influence of polarization functions on molecular orbital hydrogenation energies. *Theoretica Chimica Acta* **28**, 213–222 (1973).

[321] Francl, M. M. *et al.* Self-consistent molecular orbital methods. XXIII. A polarization-type basis set for second-row elements. *The Journal of Chemical Physics* **77**, 3654–3665 (1982).

[322] Ross, J. *et al.* GP-MolFormer: A foundation model for molecular generation. *arXiv preprint arXiv:2405.04912 [q-bio.BM]* (2024).

[323] Adilov, S. Generative pre-training from molecules. *ChemRxiv preprint ChemRxiv:2021-5fwjd* (2021).

[324] Sadowski, J. & Gasteiger, J. From atoms and bonds to three-dimensional atomic coordinates: automatic model builders. *Chemical Reviews* **93**, 2567–2581 (1993).

[325] Schwaller, P. *et al.* Molecular Transformer: A model for uncertainty-calibrated chemical reaction prediction. *ACS Central Science* **5**, 1572–1583 (2019).

[326] Fu, N. *et al.* Material transformers: deep learning language models for generative materials design. *Machine Learning: Science and Technology* **4**, 015001 (2023).

[327] Chithrananda, S., Grand, G. & Ramsundar, B. ChemBERTa: Large-scale self-supervised pretraining for molecular property prediction. *arXiv preprint arXiv:2010.09885 [cs.LG]* (2020).

[328] Bemis, G. W. & Murcko, M. A. The properties of known drugs. 1. Molecular frameworks. *Journal of Medicinal Chemistry* **39**, 2887–2893 (1996).

[329] Vignac, C. *et al.* DiGress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734* (2022).

[330] Xu, Z. *et al.* Discrete-state continuous-time diffusion for graph generation. *arXiv preprint arXiv:2405.11416 [cs.LG]* (2024).

[331] Siraudin, A., Malliaros, F. D. & Morris, C. Cometh: A continuous-time discrete-state graph diffusion model. *arXiv preprint arXiv:2406.06449 [cs.LG]* (2024).

[332] Qin, Y., Madeira, M., Thanou, D. & Frossard, P. DeFoG: Discrete flow matching for graph generation. *arXiv preprint arXiv:2410.04263 [cs.LG]* (2024).

[333] Voucher, A. *et al.* GuacaMol: Benchmarking models for de novo molecular design. URL `http://github.com/BenevolentAI/guacamol`. (Accessed: 30/01/2025).

[334] Kullback, S. & Leibler, R. A. On information and sufficiency. *The Annals of Mathematical Statistics* **22**, 79–86 (1951).

[335] Preuer, K., Renz, P., Unterthiner, T., Hochreiter, S. & Klambauer, G. Fréchet ChemNet distance: a metric for generative models for molecules in drug discovery. *Journal of Chemical Information and Modeling* **58**, 1736–1741 (2018).

[336] Chen, X. *et al.* Graph generative pre-trained transformer. *arXiv preprint arXiv:2501.01073 [cs.LG]* (2025).

[337] Hansch, C., Björkroth, J. & Leo, A. Hydrophobicity and central nervous system agents: on the principle of minimal hydrophobicity in drug design. *Journal of Pharmaceutical Sciences* **76**, 663–687 (1987).

[338] Frisch, M. *et al. Gaussian 09, Revision D. 01* (Gaussian Inc, 2009).

[339] Ebejer, J.-P., Morris, G. M. & Deane, C. M. Freely available conformer generation methods: How good are they? *Journal of Chemical Information and Modeling* **52**, 1146–1158 (2012).

[340] Halgren, T. A. Merck molecular force field. I. Basis, form, scope, parameterization, and performance of mmff94. *Journal of Computational Chemistry* **17**, 490–519 (1996).

[341] Huang, H. *et al.* Over-tokenized transformer: Vocabulary is generally worth scaling. *arXiv preprint arXiv:2501.16975 [cs.CL]* (2025).

[342] Todorović, M., Gutmann, M. U., Corander, J. & Rinke, P. Bayesian inference of atomistic structure in functional materials. *npj Computational Materials* **5**, 35 (2019).

[343] Smith, J. S., Nebgen, B., Lubbers, N., Isayev, O. & Roitberg, A. E. Less is more: Sampling chemical space with active learning. *The Journal of Chemical Physics* **148** (2018).

[344] Settles, B. Active learning literature survey. *Science* **10**, 237–304 (1995).