

Contextual Representations of the Chemical Space for Task Agnostic Machine Learning Methods

Hisham Abdel Aty

A thesis submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy to the Department of Chemistry
Imperial College London, April 2023

Abstract

The chemical space is a vast and complex multi-scale space that is difficult to represent in a way that is both computationally efficient and contextually meaningful. In practice, chemists represent molecular structures in a variety of ways, from skeletal and line formulae for organic compounds, to spectroscopic representations such as mass spectra (MS) and nuclear magnetic resonance (NMR) spectra. In this thesis, fast, contextually tailored computational representations of molecules and aggregate structures were developed. These representations not only give predictive power to computational predictors, but are also used to aid experimentalists with analysing more classical spectroscopic representations more rapidly.

The first representation developed within this thesis builds on the Small-angle X-ray scattering (SAXS) representation of lipids, an experimental spectroscopic representation of aggregate lipid structures. Synthetic modelling of lipid scattering patterns was used to generate training datasets for a convolutional neural network (CNN) to quickly predict the lipid phase of a given lipid system based on its SAXS pattern. This new 2D representation is more suitable for CNNs and provides good and fast predictive power. Combining this with a user friendly WebApp means that experimentalists are now able to analyse their results and adjust their experiments at the beamline facility if necessary.

The second representation developed within this thesis is a flexible molecular representation that initially abstracts structure data into a 768-dimensional vector, which can then be fine-tuned to a specific task. Whilst the task-agnostic, pre-trained performance of this representation is reasonable, it is the fine-tuning of the representation to a specific task that gives it its contextual power, as it can be applied to almost any chemical task given a small sample of data. The model architecture generates Molecular Fingerprints through Bidirectional Encoder Representations from Transformers (MFBERT) applied to SMILES strings.

Finally, a combination of MFBERT representations and Atomic Environment Vector (AEV) representations proved to be able to predict binding affinity of protein-ligand complexes, with performance that is competitive with the current best machine learning models in literature. The AEVs are used to represent the protein’s binding pocket, with MFBERT representing the ligand. The two representation models are pre-trained separately and fine-tuned synchronously.

Acknowledgements

Firstly, I would like to thank my supervisor, Ian Gould, for his patience and for providing consistent support throughout this PhD. I am especially grateful for Ian's understanding and flexibility during the unprecedented times of the global pandemic. I am also grateful for the Engineering and Physical Sciences Research Council for providing the funding and stipend to make this PhD possible.

I would also like to thank my colleagues within the Gould Group: Tony, Sara, Zhili, for having useful discussions about work, life and everything in-between. I would also like to thank all the other people within lab 109 for their kindness and for making working there a pleasant experience.

Finally I would like to thank my family and friends for their general support and encouragement throughout this PhD.

Declaration

All the work presented in this thesis is my own, unless otherwise stated or referenced. I have not previously submitted this work for any other research degree or at any other university. Collaboration with other researchers is acknowledged in the references to the respective publication.

Hisham Abdel Aty

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-NonCommercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Talks, Posters, Publications and Open Source Contributions

Talks

Abdel-Aty H, Gould I, *Large-scale distributed pre-training of transformers for data-driven molecular fingerprinting*, ACS Spring Meeting 2021

Abdel-Aty H, Gould I, *Contextual Representations of the Chemical Space for Task Agnostic Machine Learning Methods*, Imperial College London, PG Symposium 2022

Posters

Abdel-Aty H, Gould I, *Distributed pre-training of transformers for diverse latent representations of small molecules*, 4th RSC-BMCS / RSC-CICAG Artificial Intelligence in Chemistry 2021

Publications

Hisham Abdel Aty, Robert Strutt, Niall McIntyre, Matthew Allen, Nathan E. Barlow, Miguel Paez-Perez, John M. Seddon, Nick Brooks, Oscar Ces, and Ian R. Gould, *Machine learning platform for determining experimental lipid phase behaviour from small angle X-ray scattering patterns by pre-training on synthetic data*, Digital Discovery, 1(2):98–107, April 2022

Hisham Abdel-Aty and Ian R. Gould. *Large-Scale Distributed Training of Transformers for Chemical Fingerprinting*, Journal of Chemical Information and Modeling, 62(20):4852–4862, October 2022

Open Source Contributions

- MFBERT, 2021, <https://github.com/GouldGroup/MFBERT>
- AmberTools 21, 2021, <https://ambermd.org/AmberTools.php>
- SAXSpy, 2021, <https://github.com/GouldGroup/SAXSpy>
- Pytraj, 2020, 2022, <https://github.com/Amber-MD/pytraj>
- PyTorch Geometric, 2022, https://github.com/pyg-team/pytorch_geometric

Contents

Abstract	i
Acknowledgements	iii
1 Introduction	1
1.1 A (Brief) History of Machine Learning	4
1.2 An Introduction to Chemical Structure Representations	7
2 Background and Theory	15
2.1 Machine Learning	15
2.1.1 Regression	17
2.1.2 Capacity and Generalisation	20
2.1.3 Classification	22
2.1.4 Cross-Validation	25
2.1.5 Regularisation	27
2.1.6 Optimisation and Meta Learning	30
2.1.7 Data Augmentation	36

2.2	Deep Learning	38
2.2.1	Feed Forward Neural Networks	38
2.2.2	Recurrent Neural Networks	43
2.2.3	Convolutional Neural Networks	49
2.2.4	Seq2Seq, Autoencoding and Autoregressive models	53
2.2.5	Message Passing Neural Networks	53
2.2.6	The Transformer Model	56
2.3	Chemical Feature Representation	60
2.3.1	Molecular Fingerprints and Descriptors	61
2.3.2	String Representations	63
2.3.3	InChI	65
2.3.4	Protein Structure Representations	66
2.4	Molecular Dynamics	70
2.4.1	Molecular Mechanics	71
2.4.2	Energy Landscapes	74
2.4.3	Molecular Dynamics Simulations	75
2.4.4	Generalised Born Surface Area	76
3	Predicting Lipid Phase Behaviour from Small Angle X-ray Scattering (SAXS)	78
3.1	Lipid Phase Behaviour	80
3.1.1	Lamellar Phases	83
3.1.2	Hexagonal Phases	85

3.1.3	Inverse Bicontinuous Cubic Phases	85
3.2	Small Angle X-ray Scattering Patterns	86
3.2.1	X-ray Scattering	87
3.3	Simulating SAXS Patterns by Modelling Lipids	90
3.3.1	Lamellar Phase Modelling	90
3.3.2	Hexagonal Phase Modelling	93
3.3.3	Inverse Bi-continuous Cubic Phase Modelling	96
3.4	SAXS Data Analysis and Processing	99
3.5	Pipeline Workflow	109
3.5.1	Model Architecture	110
3.5.2	Training Workflow	113
3.6	Conclusion	116
4	Distributed Pre-training of Transformers for Property Prediction	117
4.1	Molecular Representations	118
4.2	Large Language Models and their Pre-training	119
4.2.1	Pre-training Tasks	119
4.2.2	Tokenization	121
4.3	Learning Representations of the Chemical Space	127
4.3.1	Data and Analysis	128
4.3.2	Model	134
4.3.3	Performance and Analysis	141

4.3.4	MFBERT/Siamese MFBERT as a Molecular Property Predictor	152
4.3.5	MultiTask Learning	157
4.4	Conclusion	160
5	Combining Learned Representation for Contextual Scoring Functions of Protein-Ligand Binding Affinity Predictions	161
5.1	Protein-Ligand Binding Affinity Prediction	162
5.1.1	Scoring Functions	163
5.2	ML Based Scoring Functions	166
5.2.1	Datasets	167
5.2.2	Neural Network Potentials	170
5.3	Learning Binding Affinities With MFBERT and Neural Network Potentials . . .	174
5.3.1	Data	174
5.3.2	Model Architecture	177
5.3.3	Training	181
5.3.4	Performance and Analysis	181
5.3.5	Visualisation	197
5.3.6	Ligand-only Prediction	199
5.3.7	Comparison with other models	203
5.3.8	Conclusion	205
6	Conclusion	206
6.0.1	Future Work	207

Bibliography	209
A Appendix	250
A.1 Expectation-Maximization Algorithm for SentencePiece	250
A.2 Finding Optimal Segmentations	251
A.3 RDKit’s benchmarking platform filtered target set and MFBERT scores	252
A.4 Meta Optimization of Hyperparameters	256
A.5 Copyright Licenses and Permissions	259

List of Tables

2.1	Non-linear activation functions.	40
3.1	SAXS model parameter ranges used to generate the Synthetic datasets	100
3.2	Data splits for the SAXS data	107
3.3	Model Results for predicting lipid phase behaviour on both the experimental and synthetic test sets for both 1 and 2 pre-training epochs over 0-20 fine-tuning epochs	114
4.1	The number of molecules in each raw dataset. The total number of molecules in the aggregate dataset is 1,575,933,388.	129
4.2	MFBERT’s distributed pre-training hyperparameters	137
4.3	MFBERT’s fine-tuning hyperparameters	140
4.4	Cumulative subsets of GDB-13	147
4.5	K-means performance on the pre-trained and fine-tuned models on the three classification datasets after PCA for dimensionality reduction	151
4.6	Molecular property prediction results on the MoleculeNet datasets	154
4.7	Comparison of MFBERT with state-of-the-art models on the MoleculeNet datasets	156
4.8	AUC-ROC scores for each task in the tox21 dataset using MFBERT with MTL .	159

5.1	Publicly accessible datasets of binding affinity data	167
-----	---	-----

List of Figures

1.1	A sample of the most notable milestones within the history of machine learning in the context of computational chemistry	4
1.2	Raw PDB Representation of Alanine Dipeptide in Vacuum	9
1.3	Rasterised Representation of Alanine Dipeptide PDB	10
1.4	2D Molecular Graph Representation of 7-(4,7-diazaspiro[2.5]octan-7-yl)-2-(2,8-dimethylimidazo[1,2-b]pyridazin-6-yl)pyrido[1,2-a]pyrimidin-4-one	11
1.5	Depth-First Search generation of Ciproflaxin SMILES, adapted from Macks et al. ⁶⁸	12
1.6	(a) Powder pattern of the calibration standard CeO ₂ (NIST, SRM674b) recorded at beamline I11 with a Pixium area detector (RF4343, Thales Electron Devices S. A.) using an X-ray energy of 25.5 keV. (b) The one-dimensional diffractogram, intensity versus q, of the azimuthally integrated two-dimensional pattern shown in (a). Adapted From Filik <i>et al.</i> ⁶¹	14
2.1	Illustration of model capacity through polynomial fitting: when a model has high capacity, it is more likely to overfit - high variance. When a model has low capacity, it is more likely to underfit - high bias.	21
2.2	An image visualising how ordinary regression compares to the Lasso, the Ridge and the Elastic Net Regressors. ⁸⁸ Image adapted from P. Mohan ⁸⁹	27

2.3	A visualisation of the effect of momentum on the stochastic gradient descent algorithm.	33
2.4	An example of SMILES augmentation. The canonical SMILES string of aspirin is shown on the left. The SMILES string is then augmented by varying the order of the 2D graph traversal.	37
2.5	A Multilayer Perceptron with 1 input layer (16-dim), 2 hidden layers (12-dim and 10-dim) and 1 output layer (1-dim).	39
2.6	A simple RNN architecture	44
2.7	Unrolled LSTM architecture. Image source ¹⁰⁸	49
2.8	A sequence of depth-wise convolutional layers, with a stride of 1 and no padding.	51
2.9	A sample of Max Pooling with a 2x2 filter and stride 2	52
2.10	The Transformer Model Architecture. ⁵	56
2.11	Scaled Dot-Product Attention and Multi-Head Attention. ⁵	58
2.12	An example of PDB ATOM records.	70
3.1	The three types of phospholipids and their curvature effects.	80
3.2	The lateral pressure profile of a single monomer phospholipid.	82
3.3	The lamellar phases of lipid bilayers, figure adapted from Seddon and Templer's work ¹⁵¹	83
3.4	Temperature-pressure phase diagram of a DOPC lipid bilayer in D_2O (blue) and H_2O (black). L_β' , $L_\beta I$ and P_β' are the tilted, inter-digitated and rippled phases respectively - all subsets of the L_β gel phase. Figure adapted from Matsuki et al. ¹⁵⁶	84

3.5	The normal and inverse hexagonal phase packing of cylindrical lipid micelles, figure adapted from Juliette Jouhet's work ¹⁴¹	85
3.6	The Inverse Bicontinuous Cubic Phases of Lipid Bilayers	86
3.7	The Ewald sphere for X-ray scattering, figure from Kevin Yager ¹⁷³	88
3.8	The synthetic model for a one-dimensional lamellar phase was created by convolving the reciprocal unit cell with the lattice. The lattice, b) is composed of a delta array with a lattice parameter of α . The electron density of the unit cell, a) was modelled using three Gaussian functions, which consider the relative positions of the headgroup and terminal methyl lipid ends in relation to the electron density of water.	92
3.9	A 1D lamellar phase SAXS model was created by convolving the reciprocal unit cell with the lattice peaks and multiplying their power spectra with reciprocal lattice parameter α^* . This is demonstrated by overlaying the real space amplitudes of the unit cell with the lattice peaks.	93
3.10	A 2-dimensional representation of the hexagonal lipid phase. The lattice consists of cylindrical micelles packed hexagonally along the z axis. The lattice parameter α is the distance between the centres of the micelles.	94
3.11	Raw SAXS peaks for the 5 phases - Lamellar, Hexagonal, Primitive, Gyroid and Diamond	99
3.12	Sample of blank samples used for convolution	102
3.13	Sample of the final synthetic data	102
3.14	Feature space of the synthetic data compared to that of the experimental data ¹⁹¹	103
3.15	A screenshot of the experimental SAXS labelling web app	105
3.16	Feature matrix for each phase, adapted from own work ¹⁹¹	108

3.17 Feature matrix for coexistent lipid phases - lamellar and inverse hexagonal, adapted from own work ¹⁹¹	109
3.18 Full SAXS pipeline workflow, from synthetic data generation and experimental data acquisition/labelling to model pre-training and fine-tuning. ¹⁹¹	110
3.19 EfficientNet-B0 model architecture ¹⁹¹	111
3.20 Depthwise and pointwise convolutions, together, form a depthwise separable convolution ¹⁹⁹	112
3.21 Accuracy trajectories with varying pre-training and fine-tuning epochs on both experimental and synthetic test sets ¹⁹¹	115
3.22 A screenshot of the model’s final web app that can be used by experimental scientists at the beamline facility	116
4.1 An example of a single masked language model (MLM) task. The model is being trained to predict the masked token given its context and vocabulary. Adapted from Weyssow et al. ²¹²	121
4.2 The workflow for learning representations of the chemical space ²¹⁵	127
4.3 The frequency of 85 molecular groups in a 1 million molecule random sample of the aggregate dataset. ²¹⁵	131
4.4 Three MFBERT architectures. a) is the base model, b) is a Siamese classifier and c) is the Siamese regressor, adapted from own work ²¹⁵	135
4.5 Tree-based reduction algorithm such as the one used in NCCL, Figure adapted from Carl Yang’s work ²²⁶	138
4.6 Attention of Aspirin and a bioactive molecule, adapted from own work ²¹⁵	141
4.7 RDKit’s benchmarking platform results	144

4.8	AUCROC scores for each of the 7 models/subsets. The AUCROC for pre-training on the whole of GDB-13 (i.e. subset A) was 0.67 - in-line with other subsets and language models assessed in this study.	148
4.9	t-SNE visualisation of the latent space of the pre-trained and fine-tuned models. ²¹⁵	150
5.1	Example AEV of an atom in a H, C, N, O system. Reproduced from Gao et al. ³¹⁰	173
5.2	Representation of a general deep learning system applied to a protein-ligand complex (PDB: 1XD0). The binding pocket is represented computationally (usually as a graph or a matrix) and is passed through a set of hidden layers, depending on the architecture of the model to predict the pKa. Protein illustration was generated with Mol* ³²³	177
5.3	Schematic representation of the MFBERT-ANI model series. The protein-ligand complex (PDB ID: 1XD0, rendered with Illustrate ³²⁹). Schematic a) shows the latent model of the MFBERT-ANI model series. Schematic b) shows the sum MFBERT-ANI architecture	179
5.4	CASF-2013 Scoring Performance of both latent and sum model variants of MFBERT-ANI. Values are given in pKa.	183
5.5	CASF-2013 Scoring Performance on each target cluster of both latent and sum model variants of MFBERT-ANI. RMSE Values are given in pKa.	184
5.6	CASF-2013 Scoring Performance of both latent and sum model variants of ensembles of MFBERT-ANI. Values are given in pKa.	187
5.7	CASF-2013 Scoring Performance on each target cluster of both latent and sum model variants of the Ensembled MFBERT-ANI. RMSE Values are given in pKa.	189
5.8	CASF-2016 Scoring Performance of both latent and sum model variants of MFBERT-ANI. Values are given in pKa.	191

5.9	CASF-2016 Scoring and Ranking Performance on each target cluster of both the latent and sum model variants of MFBERT-ANI. RMSE Values are given in pKa.	192
5.10	CASF-2016 Scoring Performance of both latent and sum model variants of ensembles of MFBERT-ANI. Values are given in pKa.	194
5.11	CASF-2016 Scoring and Ranking Performance on each target cluster of both the latent and sum ensemble model variants of MFBERT-ANI. RMSE Values are given in pKa.	196
5.12	Web App for visualising the regression plot and the corresponding 3D structure. This screenshot shows the selected point in the regression plot for the single latent model and the corresponding 3D structure. The point represents PDB ID: 2WCA which is a member of cluster 36, where there was a negative Pearson's r and Spearman's rank correlation.	197
5.13	Web App for visualising the regression plot and the corresponding 3D structure. This screenshot shows the selected point in the regression plot for the single latent model and the corresponding 3D structure. The point represents PDB ID: 1S38 which is a member of cluster 40, where there was a negative Pearson's r and Spearman's rank correlation.	198
5.14	CASF-2016 Scoring and Ranking Performance overall and per target cluster of the MFBERT ligand-only model. RMSE Values are given in pKa.	200
5.15	CASF-2013 Scoring and Ranking Performance overall and per target cluster of the MFBERT ligand-only model. RMSE Values are given in pKa.	202

5.16	Comparison of the MFBERT-ANI and MFBERT-ligand-only models with other state-of-the-art models from literature on the CASF-2013 and CASF-2016 benchmarks. The models are denoted as follows: the Sum only model as MFBERT-ANI S, the latent model as MFBERT-ANI L, the ligand-only model as MFBERT-ANI LO, and the ensemble models as MFBERT-ANI ES and MFBERT-ANI EL for the sum and latent variants respectively. The models used for comparison can be found here ^{263,265,266,268,269,324,326,336–342}	204
A.1	a) Trellis-like diagram for the word "hello" where the transitions are unknown. b) Trie for the word "hello". Adapted from https://towardsdatascience.com/sentencepiece-tokenizer-demystified-d0a3aac19b15	252
A.2	RDKit's benchmarking platform filtered target set and MFBERT scores.	255
A.3	Convergence, evaluation and objective plots for the meta-optimization of the hyperparameters of the MFBERT-ANI combination model.	258

Chapter 1

Introduction

Over the past decade, a number of Machine Learning (ML) algorithms have been proposed and applied to a variety of fields both within and outside the realm of the natural sciences.¹⁻⁷ This recent boom in ML has resulted in many breakthrough models designed for complex computational tasks such as Natural Language Processing (NLP) and computer vision.^{1,5} Most of these models are Deep Learning (DL) based and their development has been driven by the rapid improvements and ease of access of new tools and techniques for data mining, data analysis, data handling and processing⁸⁻¹⁴ as well as hardware improvements such as Graphical Processing Unit (GPU) speeds and Video Random Access Memory (VRAM) sizes.¹⁵

With the advent of new deep learning algorithms, the applications of ML have expanded beyond computational tasks to include a wide range of subjects, including applications in various areas of physics,¹⁶ biology^{3,17} and chemistry.¹⁸ Of course, whilst these algorithms provide impressive performance, they are not always the best performing predictors for a given application. For many tasks, usually those that don't have sufficient training data, performance of these are often not as good as the performance of simpler ML and/or statistical models.¹⁹

As these models become larger and more complex, along with the datasets they are trained on, the interpretability and comparability of their results becomes more difficult.²⁰ This drove the development of rigorous benchmarks, metrics and evaluation methods specifically for large

machine learning models and their respective tasks.^{21–29} These methods are used to compare the performance of different models and to assess the impact of new data on the performance of existing models. Task specific benchmarks are usually required due to the fact that the majority of highly performing deep learning models are not designed to be general purpose, but are tailored to a specific application/task. More recently, significant efforts have been made to develop more general models and a respective framework for evaluating them.³⁰

In order to utilise DL models effectively, careful consideration must be taken when posing the problem, that is the task at hand. The problem must be posed in such a way where: a) it is well-defined and bounded, b) it is answered by the training data with variance and bias robustness and c) can be abstracted into some linear/non-linear algebraic combination based inputs and outputs. These design tactics will improve the probability of being able to confirm that a model would be a suitable predictor for the task. All DL models also need to be iteratively optimised for the task at hand; this introduces another challenge for the model designer, where optimisation hyperparameters must be selected and optimised, usually through meta optimisation techniques.^{31,32}

Model design and validation are crucial steps for the quality assurance of such models.³¹ Although such models may seem fast, reliable and robust enough to be fully deployed in a production environment, there have been many examples where such models failed catastrophically post deployment. Such failures can have a significant impact on people’s lives when the stakes are high. One example of such failure case is where biases against women were observed in Amazon’s recruitment AI tools in 2018.³³ These edge cases are often difficult to identify and address because of the inherent difficulty of evaluating the performance of such models.

In parallel to this rapid development in ML over the past decade, computational chemistry and simulation software have also benefited from the improvements in GPU performance and data analysis tools.^{34–36} These advances have accelerated *in-silico* research and enabled broader access to the research tools. Not only that, but the combination of these advances has enabled the development of a number of new predictive techniques for understanding the behaviour of molecules,³⁷ chemical reactions,⁶ protein structure,³ protein function,³⁸ chemical topic mod-

elling,³⁹ and protein interaction prediction.³⁸ Combining these new techniques with older, more theoretical computational chemistry methods such as Density Functional Theory (DFT) calculations, Molecular Dynamics (MD) simulations and Markov Chain Monte Carlo (MCMC) simulations has enabled new approaches to computer aided drug design (CADD) which can be much faster and much more efficient in a drug discovery pipeline.

The computational performance and data analysis improvements over the past decades also resulted in the generation and availability of well constructed and well structured datasets for the research community. These datasets, along with experimental data can be used to train ML models and are often used to assess the performance of these models. One example that used bioinformatics techniques and experimental data as well as MD simulations in order to train a machine learning model to predict the folding of proteins is the AlphaFold project,³ where unprecedented performance for protein folding accuracy was achieved on a standardised benchmark.⁴⁰

This thesis will provide an overview of the state of the art techniques of ML in biological chemistry and drug discovery. The potential and current impact of ML algorithms at multiple stages of the chemical discovery pipeline will be explored, along with implementations of such algorithms across the stages and data types. The challenges of applying such algorithms to the field will be discussed as well as some examples where such techniques fail. Finally, the combination of traditional computational chemistry techniques with ML techniques will be discussed and how the development of new tools from both fields may allow them to become interchangeable and useful in the future.

The structure of the thesis will involve a series of chapters consisting of novel projects that utilise both traditional and ML techniques and/or their combination. In this part, chapters 1 and 2 will introduce the history and theoretical background of the techniques used and the challenges of applying them to chemical research problems. The next section will discuss the state of the art of ML in the field of computational chemistry and how it has been applied to the development of new tools. The third section will discuss the challenges of applying ML techniques to the field and how they may be addressed.

Chapter 3 will involve details of generating synthetic spectral data for lipid systems. A Convolutional Neural Network (CNN) is then implemented and pre-trained on this synthetic data, and later fine-tuned on experimental data to predict the packing shapes of lipid molecules given their spectroscopic data.

The fourth chapter will detail an NLP based Transformer and its data aggregation, processing and training procedure for the generation of generalised chemical fingerprints. The generated fingerprints are then tested as features for ML models and benchmarked against other state-of-the-art techniques.

The fifth chapter will detail the development of a new deep learning model for the prediction of protein-ligand interactions. This model is based on the fingerprints model in chapter 4 combined with other 3D based deep learning models. A pre-training dataset was generated using MD simulations of protein systems. MD analysis techniques were used to compute system energies to complete the dataset. The model was then fine-tuned on experimental data using a combination of traditional and deep learning techniques.

1.1 A (Brief) History of Machine Learning

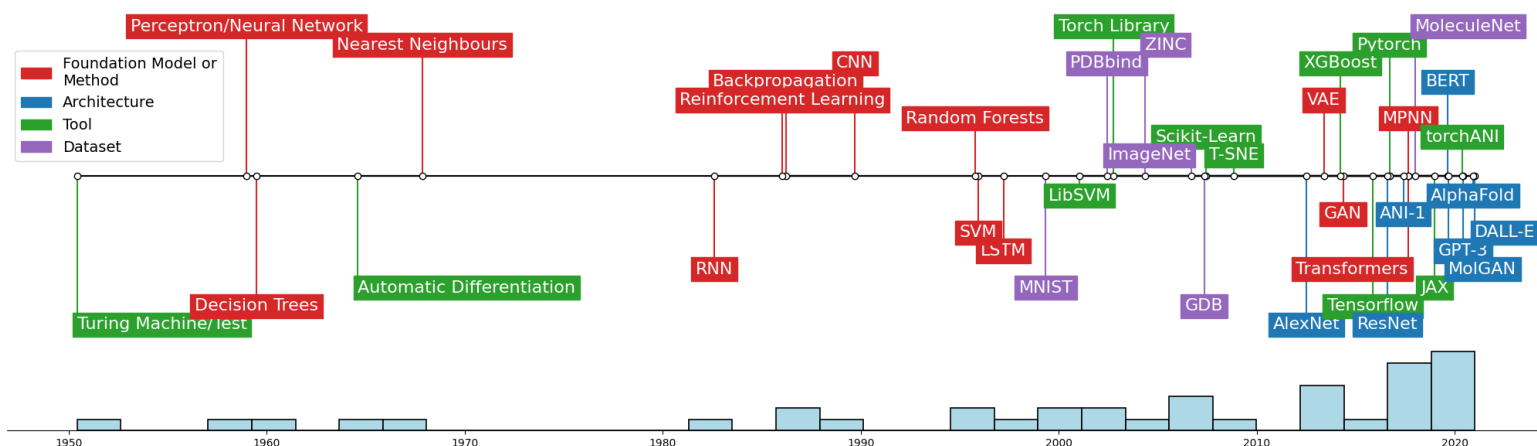


Figure 1.1: A sample of the most notable milestones within the history of machine learning in the context of computational chemistry

The concept of a non-human intelligent and/or conscious system begins in antiquity, with many cultural and more mythical references to some Artificial Intelligence (AI) beginning in the early

19th century.⁴¹

The more practical implementations and theorisations of AI began in the early 1950s. Figure 1.1 shows the more notable milestones across the timeline of AI development that are relevant for the work in this thesis. It can be seen that the rate of development in this field has increased exponentially in later years, as well as some gaps in the timeline where there was little development.

The concepts of AI were introduced by Alan Turing, who introduced the Turing Learning Machine and the Turing Test.⁴² These proposed Turing learning Machines were the first rigorous proposals for the development of artificial intelligence and foreshadow the development of modern genetic algorithms. The Turing Test was the first test of intelligence. It involved a conversation with a machine and a human, as well as a human with another human. If the human is unable to distinguish between the two conversations, the test is passed.

Later in the 1950s we saw the emergence of the first implemented artificial intelligence system, a perceptron. At the time, this was a voltage-driven analogue linear algebra machine. The perceptron was a simple machine that was able to learn the function of a single input to output. This Perceptron was built by Frank Rosenblatt.⁴³ It consisted of potentiometers for voltage control and weight updates were performed by electric motors. With a total of 400 photocells, the perceptron was designed for image recognition, and whilst there was some success with the perceptron, it was not a viable solution for image classification at the time - failing to predict many image classes and patterns. This perceptron however is the basis of all modern Feed-Forward Neural Networks (FFNNs) and is considered a sub-unit (node) of a larger modern network.

The classification was determined by the output of the perceptron. The output was the sum of the weighted inputs. If the sum was greater than zero, the output was 1, otherwise it was 0.

$$\begin{aligned}
O_p &= \sigma\left(\sum_{i=1}^{400} i_i \times W_i + b\right) \\
&= \begin{cases} 1 & \text{if } \sum_{i=1}^{400} i_i \times W_i + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.1}
\end{aligned}$$

Later in the 1950s and 1960s other, less general and less computationally expensive approaches to intelligence were explored such as Decision Trees⁴⁴ and nearest neighbour classification.⁴⁵

The Decision Tree was a recursive structure that was used to classify data based on some criteria at each level within the tree. This was designed to be used in a supervised learning setting, where the data was known in advance and the decision tree was designed to predict the class of the data.

Next came (K-)Nearest Neighbour Classification. This is also a supervised learning algorithm that was used to classify data based on the distance between new data and the nearest neighbour of a known class. This assumes that the data is linearly separable and that close neighbours in the applied Euclidean space are more likely to be in the same class.

In parallel, computational advances in automatic gradient algorithms (Automatic Differentiation⁴⁶) ushered in the later development of neural networks. When they were rediscovered as Backpropagation⁴⁷ in the 1980s, these algorithms later became essential to training modern Neural Networks.

In the late 1960s and the 1970s, there were very few advances (if any) in AI. This period is commonly referred to as the *AI Winter*. This was due to the lack of interest and funding for the development of AI, which was mostly a result of the relatively poor performance of learning algorithms at the time, as they were largely bottlenecked by a lack of big data and available computation.

Over the 1980s and 1990s, the fundamental algorithms for modern AI were developed and the first successful AI systems were used. These systems include modern recurrent neural networks (RNN),^{47,48} deep neural networks (DNN),^{47,49} convolutional neural networks (CNN),⁵⁰ random

forests (RF),⁵¹ support vector machines (SVM)⁵² as well as reinforcement learning (RL).⁵³ Initially, the classical ML algorithms (RF, SVM) were outperforming more elaborate neural algorithms. As such they became the more popular choice for ML during the 1990s/early 2000s. From the mid 2000s to the mid 2010s, a huge amount of publicly available training data was made available,²² and tooling for developing ML systems became much more efficient and more accessible.¹² This caused a shift in algorithm preferences within the ML community. Deep Neural Networks were now able to be trained effectively on large datasets and were outperforming SVMs and RFs.¹

The massive boost in performance gained from DNNs (notably the AlexNet CNN on the ImageNet dataset¹) resulted in better, more accessible tooling for the development of these models, which resulted in further development of new Foundation models and training methods (notably Transformers⁵ and Message Passing Neural Networks (MPNN)⁵⁴). This cycle of better data, tools and algorithms has propagated over the past decade, and has culminated in the development of a number of state of the art machine learning algorithms.^{4,55,56} The histogram of the timeline shows the pace of development within the ML community is ever increasing and the field is becoming ever more multidisciplinary and complex.

1.2 An Introduction to Chemical Structure Representations

The chemical space involves a wide variety of representations of chemical structures. The most simple representations are the 2D bonded-atomic representations, usually drawn by organic chemists. This molecular representation represents the chemical structure as a collection of atoms connected by bonds. These are usually the smallest sub-units of a larger, biological chemical structure such as a protein or sugar. Other representations used by humans are the molecular formula, which just states the number and type of atoms in a chemical structure but contains no bonding information (for example, C_2H_6O for ethanol).

There exists computational representations of chemical structures in 1D, 2D and 3D. The simplest representation is that of the atomic representation in 3D Euclidean space. Cartesian coordinates are used to represent the position of each atom in the space. This representation is most common in situations where the 3D structure of a system affects its function such as polymer systems like polymers, proteins, and deoxyribonucleic acid (DNA)/ribonucleic acid(RNA); whereas other representations are used in situations where the 3D structural information of a system is not essential.

Whilst 3D representations contain a large amount of data to define atom positions in space, bonding information is usually not included and/or is predefined elsewhere. The Protein Data-Bank (PDB)⁵⁷ format builds on this atom-wise cartesian system by abstracting away bonding information by grouping atoms into one of the 22 standard amino acid residues and chains (including Selenocysteine and Pyrrolysine), or the ambiguous and undetermined groups - GLX, ASX, UNK. This limits this representation to only residues defined in the PDB format. Further iterations of the PDB format allow custom bonding information and custom residues to be included. Figures 1.2 and 1.3 show the raw PDB representation of alanine dipeptide in a vacuum and the rasterised representation of the PDB respectively.

Figure 1.2: Raw PDB Representation of Alanine Dipeptide in Vacuum

```
REMARK    ALANINE-DIPEPTIDE
ATOM      1  1HH3 ACE      1      2.000    1.000   -0.000
ATOM      2   CH3 ACE      1      2.000    2.090    0.000
ATOM      3  2HH3 ACE      1      1.486    2.454    0.890
ATOM      4  3HH3 ACE      1      1.486    2.454   -0.890
ATOM      5   C    ACE      1      3.427    2.641   -0.000
ATOM      6   O    ACE      1      4.391    1.877   -0.000
ATOM      7   N    ALA      2      3.555    3.970   -0.000
ATOM      8   H    ALA      2      2.733    4.556   -0.000
ATOM      9   CA   ALA      2      4.853    4.614   -0.000
ATOM     10   HA   ALA      2      5.408    4.316    0.890
ATOM     11   CB   ALA      2      5.661    4.221   -1.232
ATOM     12  1HB   ALA      2      5.123    4.521   -2.131
ATOM     13  2HB   ALA      2      6.630    4.719   -1.206
ATOM     14  3HB   ALA      2      5.809    3.141   -1.241
ATOM     15   C    ALA      2      4.713    6.129    0.000
ATOM     16   O    ALA      2      3.601    6.653    0.000
ATOM     17   N    NME      3      5.846    6.835    0.000
ATOM     18   H    NME      3      6.737    6.359   -0.000
ATOM     19   CH3  NME      3      5.846    8.284    0.000
ATOM     20  1HH3  NME      3      4.819    8.648    0.000
ATOM     21  2HH3  NME      3      6.360    8.648    0.890
ATOM     22  3HH3  NME      3      6.360    8.648   -0.890
TER
END
```

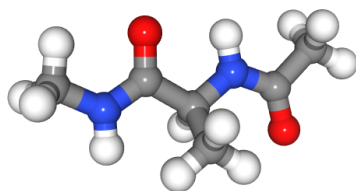


Figure 1.3: Rasterised Representation of Alanine Dipeptide PDB

Other 3D representations include Mol files and SDF files (which are multiple Mol files combined in essence),⁵⁸ these are commonly used for storing chemical structures of small molecules. These include connection tables which explicitly define the bonding information. They are not normally used for protein structures as the PDB format is a more efficient representation, since proteins are usually limited to a set of bond/atom configurations (residues), thus repeated bonding information is not necessary.

2D representations are used for storing chemical structures of small and large molecules. These include molecular graphs,⁵⁹ molecular fingerprinting algorithms⁶⁰ as well as some spectroscopic representations of molecules.⁶¹ These representations reduce the real 3D structure into subset of extracted features, usually specific to the task required. Since 2D representations are significantly more efficient than 3D computationally, they are often used for large scale chemical space filtering and prediction⁶² as they provide a good compromise between computational efficiency and accuracy.

One of the most common 2D representations is the 2D molecular graph.⁶³ This is a representation of the chemical structure as a collection of nodes and edges. Each node represents an atom and each edge represents a bond between two atoms. The nodes are connected by the edges. Graph algorithms can then be used on this representation to traverse through the molecule and find partitions, communities and based on these graph features predict molecular properties.

Furthermore, ML algorithms can be applied to such graphs as each graph can be represented as an *Adjacency matrix*. This is a matrix of size $n \times n$ where n is the number of nodes in the graph. The matrix entry A_{ij} is 1 if there is an edge between node i and node j and 0 otherwise. In

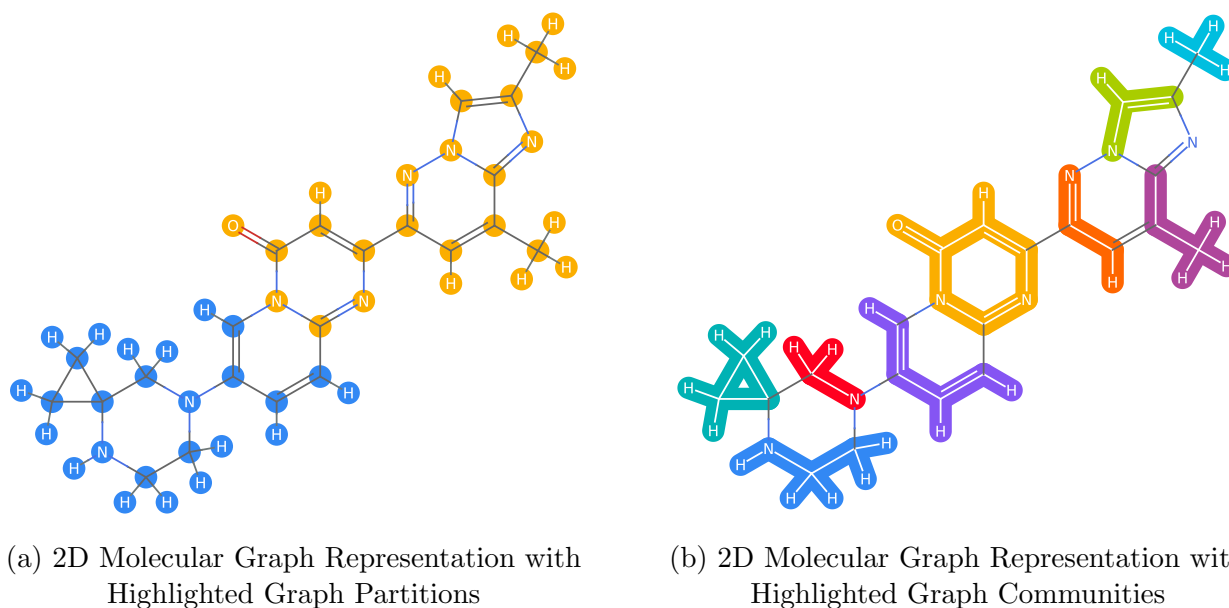


Figure 1.4: 2D Molecular Graph Representation of 7-(4,7-diazaspiro[2.5]octan-7-yl)-2-(2,8-dimethylimidazo[1,2-b]pyridazin-6-yl)pyrido[1,2-a]pyrimidin-4-one

most cases, matrix entries A_{ij} are weighted to represent more complex graph features. Figure 1.4 shows a 2D molecular graph representation of the drug-molecule risdiplam with highlighted graph partitions and graph communities.

One of the more successful Graph Neural Networks (GNNs) is the *Graph Convolutional Network* (GCN).⁶⁴ This is a neural network architecture that uses graph convolutional layers to represent the graph. The graph convolutional layers are used to represent the graph as a matrix. The matrix is then fed into a fully connected neural network to predict the graph features. This was a precursor to *Message Passing Neural Networks* (MPNNs)⁵⁴ which are more recent and more powerful, achieving state of the art performance on quantum molecular property prediction tasks.

Finally, we have 1D representations of molecules and molecular graphs. The most common human-legible 1D representation is the *Simplified Molecular-Input Line-Entry System* (SMILES) representation.⁶⁵ This is a concise and readable string representation of the chemical structure. Developed in the 1980s it has since been used as one of the most preferred molecular representations within chemical databases.

Given that SMILES is a 1D representation of 2D chemical structures, it can be thought of

as a projection of the 2D structure onto a 1D plane. Depending on the path of the Graph traversal, many 1D representations of the same 2D chemical structure can be obtained. These redundancies prevent SMILES to be used as a chemical identifier. Attempts have been made to standardise SMILES through canonicalisation, but different canonicalisation algorithms result in different SMILES representations.^{66,67} Figure 1.5 shows a Depth-First Search (DFS) generation of the SMILES from the 2D graph chemical structure of Ciprofloxacin.

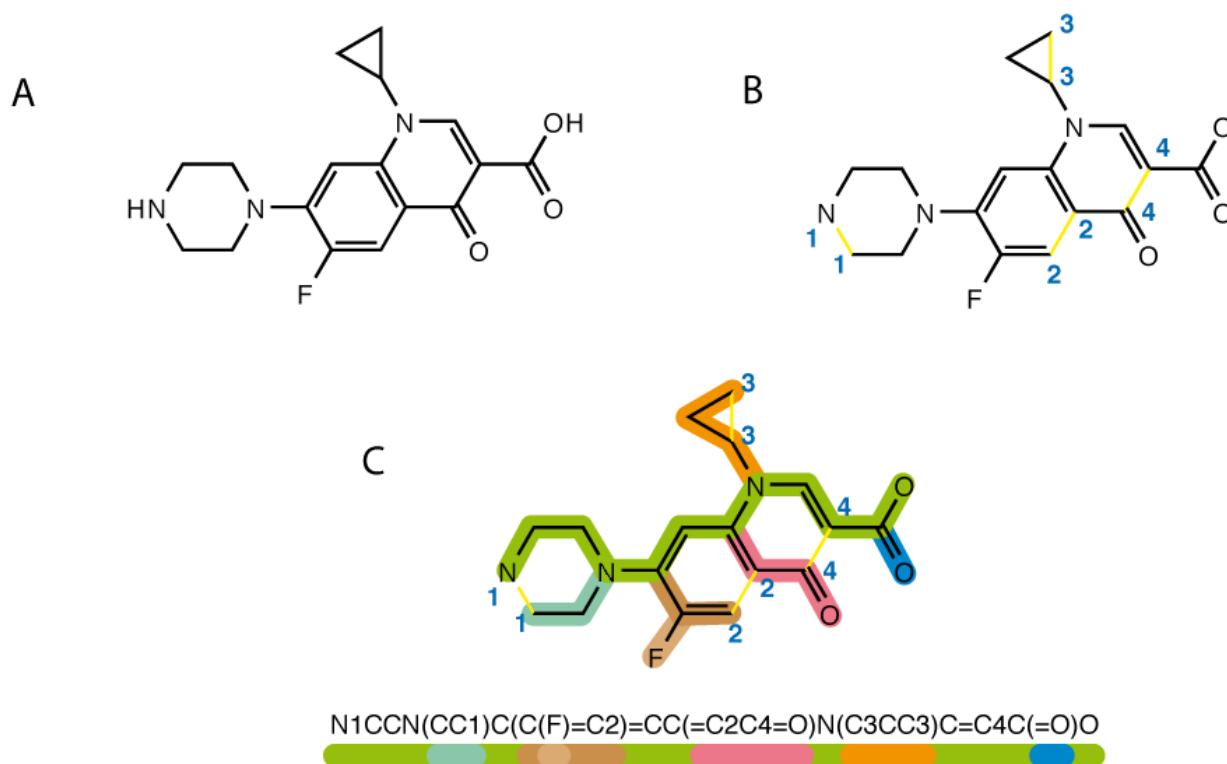


Figure 1.5: Depth-First Search generation of Ciprofloxacin SMILES, adapted from Macks et al.⁶⁸

Recent works have shown that SMILES is a good representation for some chemical space filtering and prediction.^{29,37,62,69} Further work has been done to constrain SMILES therefore reducing redundancies and increasing consistency. One such iteration on SMILES is another string representation known as Self-Referencing Embedded Strings (SELFIES).⁷⁰ This representation is a 1 to 1 mapping between strings and molecules in the chemical space. That is, every possible SELFIES string is a valid molecule. This is in contrast to the SMILES representation where some strings are not valid molecules. This is an important feature of the representation, especially when it comes to Generative Molecular Models such that the model can only generate valid molecules.⁷¹

Other 1D representations include non-human-legible molecular fingerprints. These are usually computational representations of some extracted features from the molecule. The most common representation is a circular fingerprint (ECFC4),⁶⁰ which is a type of Morgan Fingerprint,⁷² which is a bit vector of size n where n is a predefined number of bits. Each atom is identified, then the molecule is traversed circularly and iteratively, hashing and compressing the identified atoms in each iteration, until the entire molecule is traversed. This representation exhibits rotational, translational and permutational invariance and is thus unique to each molecule.

Whilst the above representations are useful to perform computations effectively, they are still theoretical representations which abstract predefined structures and behaviours. Experimental representations can be used to verify and confirm the validity of the theoretical and computational representations. Moreover they can be used to extract features from the experimental data which can be used to aid a predictive model. However, experimental data is commonly noisy and usually requires manual pre-processing and analysis to be used effectively.

One example of such data is spectroscopic data, more specifically, Small Angle X-ray Scattering (SAXS) diffraction patterns.⁷³ SAXS is a technique used to measure the size and shape of a molecule. It involves scattering a beam of X-rays off a sample and measuring the intensity of the scattered beam. The intensity of the scattered beam is a function of the size and shape of the sample. This is a very useful technique for measuring the size and shape of large aggregates of molecules, such as lipids and proteins, which are difficult to measure using other techniques.

Figure 1.6 (a) shows that the raw SAXS patterns are 2D diffraction rings from the beamline. These rings are then radially integrated to produce a 1D diffractogram, as shown in Figure 1.6 (b). This 1D diffractogram is then analysed by a scientist and is used to extract features from the experimental data. These features can then be formalised computationally and used to train a predictive model.

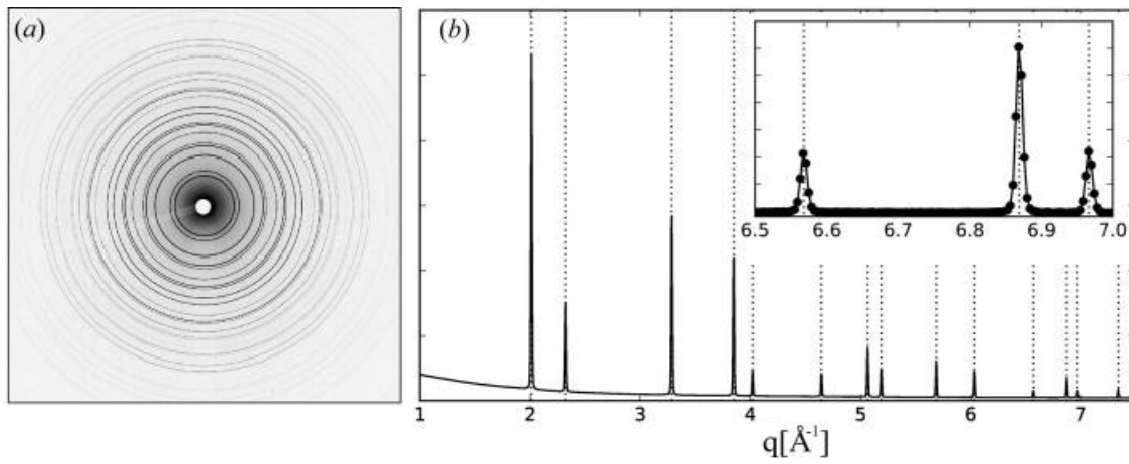


Figure 1.6: (a) Powder pattern of the calibration standard CeO₂ (NIST, SRM674b) recorded at beamline I11 with a Pixium area detector (RF4343, Thales Electron Devices S. A.) using an X-ray energy of 25.5 keV. (b) The one-dimensional diffractogram, intensity versus q , of the azimuthally integrated two-dimensional pattern shown in (a). Adapted From Filik *et al.*⁶¹

Chapter 2

Background and Theory

2.1 Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that focuses on the development of algorithms that allow computers to learn from data.⁷⁴ These are usually based on, or are scaled versions of statistical models. ML algorithms are used in a wide variety of applications, such as email filtering⁷⁵ and computer vision,¹ and have dramatically improved the performance of many real-world systems. This section provides an overview of the main approaches within ML.³¹

Supervised Learning

Supervised learning is the task of learning a function that maps an input to an output based on example input-output pairs (training). These are known as supervisory signals and guide the latent space of the model such that maximum separation is achieved between the different outputs. Inputs are usually multidimensional feature-vectors and the training data is a matrix. Then, through iterative optimisation of some objective function, the model learns to map the input to the output. Examples of supervised algorithms include SVMs, RF, and NNs. The

ultimate goal of supervised learning is to find a function that generalizes well to unseen (test) data from the given training data. Supervised learning is often divided into classification and regression problems. In classification problems, the goal is to predict a discrete-valued output. In regression problems, the goal is to predict a continuous-valued output.

Unsupervised Learning

Unsupervised learning is the task of learning some grouping that describes the structure of the data without any training examples. This is achieved by looking at commonalities and differences within the feature vectors. These can be distance based or otherwise. The advantage of unsupervised learning is that it does not require labelled, classified or categorised data, which can be expensive to obtain, or is simply unavailable. Unsupervised learning is used abundantly in data mining and is a key component of many clustering algorithms. Examples of unsupervised algorithms include K-means, Density-Based Spatial Clustering of Applications with Noise (DBSCAN),⁷⁶ and Principal Component Analysis (PCA).⁷⁷ Unsupervised learning is also the backbone for density estimation for example for finding the probability density function (PDF) of a given data set.

Semi-Supervised Learning

This approach is for one of the most common situations encountered in practice, where only a small fraction of the data is labelled. Semi-supervised learning is the task of learning a function that maps an input to an output based on example input-output pairs and unlabelled data. The goal is to find a function that generalizes well to unseen (test) data from the given training data. Examples of semi-supervised algorithms include Self-Training (ST),⁷⁸ Co-training (CT),⁷⁹ and Transductive Support Vector Machines (TSVM).⁸⁰

Self-Supervised Learning

Self-supervised learning is one of the more recent training approaches, and currently one of the most common for text data and NLP.^{2,55,81,82} This approach obtains supervisory signals from the unlabelled text data by leveraging the underlying structure in the data. The usual goal of this approach in NLP is to learn a representation of the text that is useful for downstream tasks. This is achieved by hiding some part of the unlabelled text and asking the model to predict the hidden text given words in the vocabulary. This technique is known as Masked Language Modelling (MLM). Examples of self-supervised algorithms include Word2Vec,⁸³ GloVe,⁸⁴ and BERT.⁵⁵ Self-supervised learning also gives rise to the notion of pre-training and fine-tuning. Pre-training is the process of training a model on a large corpus of unlabelled text. Fine-tuning is the process of training a model on a smaller corpus of labelled text. This is a common approach for NLP tasks, where a large corpus of unlabelled text is usually available to pre-train a model, and then the model is fine-tuned on a smaller, more targeted corpus of labelled text. Alternative pre-training methods exist that build upon both MLM techniques and pre-training, such as ELECTRA, which uses a discriminator to detect masked and replaced tokens in the pre-training process to create a new replaced token detection (RTD) task.⁸⁵ These approaches are all examples of transfer learning, where information is learned across different tasks.

Reinforcement Learning

Reinforcement learning is a system/environment where a software agent is provided with positive or negative feedback given its actions in an environment. The goal is a form of simulation-based optimisation where the agent learns to navigate, usually a Markov Decision Process (MDP), in order to maximise positive feedback.⁸⁶

2.1.1 Regression

Machine Learning problems can be broadly separated into two categories - **classification** and **regression**. Regression is the task of predicting a continuous-valued output. This is in contrast

to classification, where the goal is to predict a discrete-valued output. This section provides an overview of the main approaches within regression.

Linear Regression

Within a linear regression problem, the task is to take a feature vector $\mathbf{x} \in \mathbb{R}$ and predict a continuous-valued vector or scalar $\mathbf{y} \in \mathbb{R}$. This assumes that $\mathbf{y} = f(\mathbf{x})$ where $f(\mathbf{x})$ is a linear function. We denote the predicted value as $\hat{\mathbf{y}}$ and define the linear regression formula as follows:

$$\hat{\mathbf{y}} = \mathbf{w}^\top \mathbf{x} + \mathbf{b} \quad (2.1)$$

where $\mathbf{w} \in \mathbb{R}^d$ and $\mathbf{b} \in \mathbb{R}^d$ are the parameters of the model. $\mathbf{w}$ can be thought of as the weights applied to each feature in $\mathbf{x}$ that contributes to the output. The goal of linear regression training is to find the optimal parameters $\mathbf{w}$ and b that minimise the loss function $L(\mathbf{y}, \mathbf{w}, b)$, which is usually the mean squared error (MSE) or mean absolute error (MAE).³¹ The loss function is applied to both training and test samples (training loss and test loss respectively), if they are correlating then the model is generalising/learning. The MSE is defined as follows:

$$L(y, \mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (2.2)$$

where n is the number of samples in the training or testing set. The MAE is defined as follows:

$$L(y, \mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^n |\hat{y}_i - y_i| \quad (2.3)$$

From these loss functions, we can observe that $L \rightarrow 0$ as $\hat{y} \rightarrow y$. Thus in order to get a perfect prediction, our objective is to minimise the loss function towards $L = 0$.

The MSE is more sensitive to outliers than the MAE and is still differentiable. It is therefore more robust in practice as it is more suitable for gradient-based optimisation algorithms. The

MSE is however, more computationally expensive to compute than the MAE. The magnitude of the distance scalar (or vector in multidimensions) $\hat{y}_i - y_i$ can be denoted as the L1 norm, and the squared magnitude can be denoted as the L2 norm. These norms can be used for model Regularisation as discussed in Section 2.1.5. Since the L2 norm is the Euclidean norm which measures distance, we may also want to apply it to a matrix, this then becomes known as the Frobenius norm which is defined as follows:

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n A_{ij}^2} \quad (2.4)$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the matrix. The Frobenius norm is usually used in the context of matrix factorisation.

Now that we have our objective, in this case, we can simply solve for the optimal parameters $\mathbf{w}$ and b by taking the partial derivatives of the loss function with respect to $\mathbf{w}$ and b and setting them to zero. This is known as the normal equation. The normal equation is defined as follows:

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \quad (2.5)$$

where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the feature matrix, and $\mathbf{y} \in \mathbb{R}^n$ is the target vector. The normal equation can be computationally expensive to compute, and is only feasible for small datasets. It can be solved using the Moore-Penrose pseudoinverse, which is computationally more efficient. The pseudoinverse is defined as follows:

$$\mathbf{X}^\dagger = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \quad (2.6)$$

The pseudoinverse can be used to solve the normal equation as follows:

$$\mathbf{w} = \mathbf{X}^\dagger \mathbf{y} \quad (2.7)$$

The pseudoinverse can also be used to solve the normal equation for the bias term b as follows:

$$b = \frac{1}{n} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i) \quad (2.8)$$

where $\mathbf{x}_i$ is the i^{th} row of $\mathbf{X}$.

2.1.2 Capacity and Generalisation

We have now trained (computed the parameters of) our linear regression model. We have done so by minimising the loss on the training data. However, what we are really interested in is how well the model generalises to unseen data. This is where the test set comes in. The test set is a set of samples that are not used during training, and are used to evaluate the performance of the model. It is used to compute the test loss, which is then used to evaluate the performance of the model. If the test loss is significantly higher than the training loss, then the model is **overfitting**. If the test loss is lower than the training loss and the losses for both are high, then the model is likely **underfitting**. If the test loss is similar to the training loss, then the model is generalising well. Whether a model is able to overfit or underfit is known as its **Capacity**. The capacity of a model is the amount of information it can store. A model with high capacity can store a lot of information, and therefore is more likely to overfit. A model with low capacity can store less information, and therefore is more likely to underfit. The capacity of a model is determined by the number of parameters in the model. The more parameters, the higher the capacity. This is demonstrated by fitting a polynomial of degree 4 by varying the degree of the estimator polynomial as shown in Figure 2.1 The capacity of a model can be tuned by Regularisation, which is discussed in Section 2.1.5.

The Bias-Variance Trade-off

There exists an inherent trade-off that must be made with every ML model. In short, The bias-variance trade-off is the trade-off between the bias and variance of a model. The bias of a

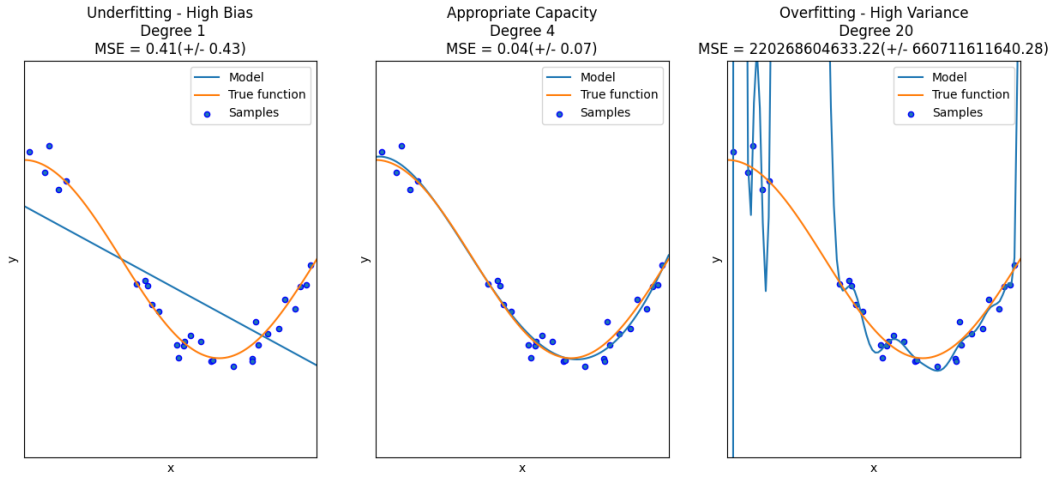


Figure 2.1: Illustration of model capacity through polynomial fitting: when a model has high capacity, it is more likely to overfit - high variance. When a model has low capacity, it is more likely to underfit - high bias.

model is the difference between the expected prediction of the model and the true value. The variance of a model is the variability of the model's predictions.

The bias of an estimator is defined as:

$$\text{Bias}(\hat{f}) = \mathbb{E}[\hat{f}(x)] - f(x) \quad (2.9)$$

where $\hat{f}(x)$ is the prediction of the model, and $f(x)$ is the true value. An estimator is said to be unbiased if $\text{Bias}(\hat{f}) = 0$. An estimator is said to be consistent if $\text{Bias}(\hat{f}) \rightarrow 0$ as $n \rightarrow \infty$. And asymptotically unbiased if $\text{Bias}(\hat{f}) \rightarrow 0$ as $n \rightarrow \infty$.

The variance of an estimator is defined as:

$$\text{Var}(\hat{f}) = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2] \quad (2.10)$$

where $\hat{f}(x)$ is the prediction of the model, and $\mathbb{E}[\hat{f}(x)]$ is the expected prediction of the model.

The bias-variance trade off is thus defined as follows:

$$\text{Bias}(\hat{f})^2 + \text{Var}(\hat{f}) = \text{MSE}(\hat{f}) \quad (2.11)$$

where $\hat{f}(x)$ is the prediction of the model, and $\text{MSE}(\hat{f})$ is the mean squared error of the model. The bias-variance trade off is a fundamental concept in ML, and is important to understand when tuning the capacity of a model.

2.1.3 Classification

The other broad category of ML tasks is classification. Classification tasks can either be supervised, or unsupervised. Most commonly, with unsupervised classification, the task is some form of clustering task where similar samples are grouped (classified) to be the same. Supervised classification is mostly based on estimating a probability distribution of a sample belonging to a class. The probability distribution is then used to classify the sample. The probability distribution, $p(y|\mathbf{x})$ is commonly estimated using a **Softmax** function. The Softmax function is defined as follows:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (2.12)$$

Logistic Regression

Logistic regression is a classification algorithm that is used to estimate the probability distribution of a sample belonging to a class. It relies on maximum likelihood estimation to estimate the parameters of the model and considers the log odds of a sample belonging to a class. This is done by simply defining a new family of distributions, where there are only 2 classes (0 and 1). We only therefore need to define the probability distribution of the positive class, $p(y = 1|\mathbf{x})$. As the probability distribution of the negative class is simply $p(y = 0|\mathbf{x}) = 1 - p(y = 1|\mathbf{x})$. The log odds (logits) of a sample belonging to a class is defined as follows:

$$\log\left(\frac{p(y = 1|\mathbf{x})}{p(y = 0|\mathbf{x})}\right) = \mathbf{w}^\top \mathbf{x} + b \quad (2.13)$$

We know that the probability distribution from linear regression is defined as follows:

$$p(y|\mathbf{x}) = \mathcal{N}(\mathbf{w}^\top \mathbf{x} + b, \sigma^2) \quad (2.14)$$

where $\mathbf{w}$ is the weight vector, b is the bias term, and σ^2 is the variance of the distribution. Logistic regression is a special case of linear regression, where the target variable is binary. The probability distribution is estimated using the sigmoid function, which is defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.15)$$

The probability distribution from logistic regression is defined as follows:

$$p(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b) \quad (2.16)$$

where $\mathbf{w}$ is the weight vector, b is the bias term, and σ is the sigmoid function.

Support Vector Machines

Another classification algorithm is the Support Vector Machine (SVM).⁵² SVM still relies on the linear function, but instead of using a linear regression model to estimate the probability distribution of a sample belonging to a class, it uses linear regression to estimate the distance of a sample from a decision boundary. The decision boundary is defined as the line that separates the 2 classes. The distance of a sample from the decision boundary is defined as follows:

$$\text{distance}(\mathbf{x}) = \frac{\mathbf{w}^\top \mathbf{x} + b}{\|\mathbf{w}\|} \quad (2.17)$$

During training SVM determines the optimal weight vector and bias term that maximizes the distance of the samples from the decision boundary. The distance of a sample from the decision boundary is called the margin. The optimal weight vector and bias term are determined by maximizing the margin. The margin is defined as follows:

$$\text{margin} = \frac{2}{\|\mathbf{w}\|} \quad (2.18)$$

Whilst the aim is to maximize the margins, the SVM algorithm also tries to minimize the weight vector. This is done to prevent overfitting. The SVM algorithm is thus defined as follows:

$$\text{minimize } \frac{1}{2} \|\mathbf{w}\|^2 \text{ subject to } y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad (2.19)$$

where y_i is the label of the sample, $\mathbf{x}_i$ is the sample, $\mathbf{w}$ is the weight vector, and b is the bias term.

Samples within the margin are called support vectors. The SVM algorithm is thus defined as follows:

$$\text{minimize } \frac{1}{2} \|\mathbf{w}\|^2 \text{ subject to } y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \forall i \in \text{support vectors} \quad (2.20)$$

One of the most important features of SVM is known as the kernel trick. The kernel trick is a method of transforming the data into a higher dimensional space, where the data is linearly separable in a computationally efficient way. This can be achieved by writing the linear function as a generalised dot product, which can then be evaluated in some higher dimension through a kernel. There are many kernel functions that can be used, but in general, a kernel can be defined as:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j \quad (2.21)$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the samples. The kernel trick is defined as follows:

$$\mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^\top \mathbf{x} + b \quad (2.22)$$

where $\mathbf{w}$ is the weight vector, b is the bias term, α_i is the Lagrange multiplier, y_i is the label of the sample, and $\mathbf{x}_i$ is the sample.

The generalised dot product is thus defined as follows:

$$\mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^n \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b \quad (2.23)$$

where $\mathbf{w}$ is the weight vector, b is the bias term, α_i is the Lagrange multiplier, k is the kernel function, and $\mathbf{x}_i$ is the sample.

One of the most common kernels is the radial basis function (RBF) kernel,⁸⁷ which is defined as follows:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \quad (2.24)$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the samples, and γ is a hyperparameter that controls the width of the kernel. The RBF kernel is commonly used in SVM because it is computationally efficient and can be used to transform the data into a general higher dimensional space (infinite dimensions). This latent space is also known as the SVM feature space.

2.1.4 Cross-Validation

In general, when building an ML model, you would want to verify whether the algorithm, hyperparameters and datasets you have chosen are sufficient to achieve satisfactory generalisable performance on future real world data. The standard way to verify this is to split the original raw dataset into three sets. A training set, a validation set, and a test set. Broadly speaking, the training set is used to train the model, the validation set is used to tune the hyperparameters - the model is also usually monitored during training with regular evaluations on this set every n iterations. And the test set is used to evaluate the performance of the model after training. The split is usually an 80/10/10 split with various splitting strategies depending on the data. One of the more common splitting strategies is random splitting, where the samples are split randomly into each of the three sets. One other common splitting method in cheminformatics

is scaffold splitting, where molecules within the dataset are separated into different sets if their core structure is different. This is done to ensure that the model is not overfitting to a particular core structure, which increases the challenge of the task for a high-bias model, whilst simultaneously reducing the probability of overfitting for a high-variance model.

In most cases, with large datasets, these sets can be pre-determined and static. If the dataset is small however, another technique known as k-fold cross-validation may be used. In k-fold cross-validation, the dataset is split into k folds, where k is a hyperparameter. The model is then trained k times, where each time, a different fold is used as the validation set, and the remaining $k - 1$ folds are used as the training set. The performance of the model is then averaged across the k folds. This technique is useful when the dataset is small, as it allows for more data to be used for training, and also allows for more data to be used for validation. The main drawback of this technique is that it is computationally expensive, as the model is trained k times.

Hyperparameter Tuning

Hyperparameters are parameters that are not directly learnt within the model, but are instead set by the user. These parameters are usually set before training the model, and are used to control the learning process. The specific hyperparameters vary by algorithm but some of the most common hyperparameters are the learning rate, the number of epochs, the batch size, the number of layers, the number of neurons per layer, the activation function, the loss function, the optimiser, and the kernel function. The most common hyperparameter tuning technique is grid search, where the user defines a grid of hyperparameters, and the model is trained for every combination of hyperparameters in the grid. The performance of the model is then evaluated on the validation set, and the hyperparameters that achieve the best performance are chosen. The main drawback of this technique is that it is computationally expensive, as the model is trained for every combination of hyperparameters in the grid. This will be discussed in more detail in Section 2.1.6.

2.1.5 Regularisation

Regularisation is a technique used to prevent overfitting in ML models. Overfitting is a common problem in ML models, where the model learns the training data too well, and is unable to generalise to new data. This is usually caused by the model having too many parameters, and thus being able to fit the training data too well. Regularisation is a technique used to prevent this from happening. There are two main types of regularisation, L1 and L2 regularisation. L1 regularisation is also known as Lasso regularisation, and L2 regularisation is also known as Ridge regularisation. Figure 2.2 shows an image visualising the effect of three regularisations. Lasso (L1 Norm), Ridge (L2 Norm), and Elastic Net regularisation (L1+L2 Norm) regressors (left to right).

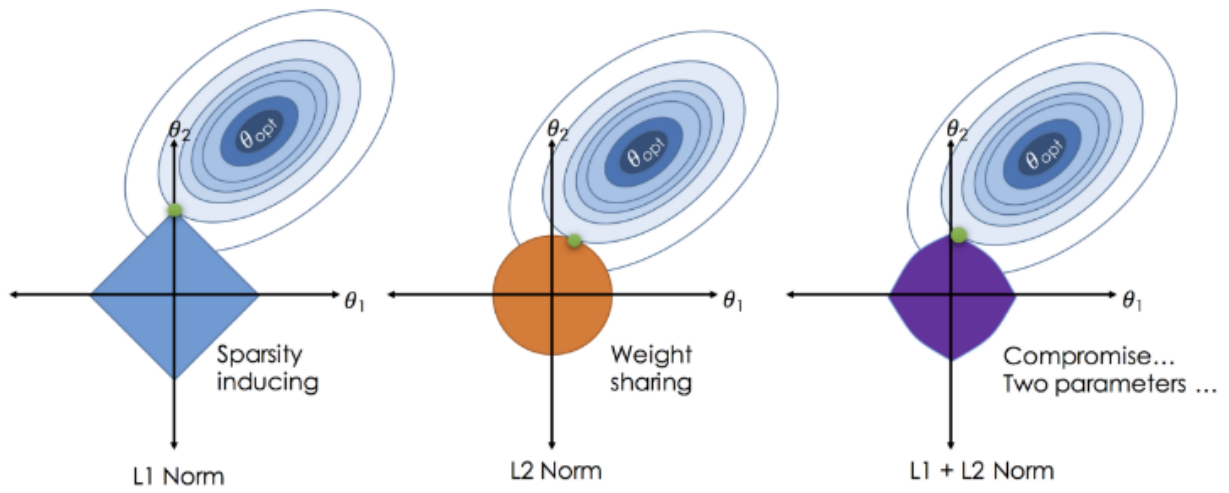


Figure 2.2: An image visualising how ordinary regression compares to the Lasso, the Ridge and the Elastic Net Regressors.⁸⁸ Image adapted from P. Mohan⁸⁹

L1 regularisation is defined as follows:

$$\text{L1 Regularisation} = \sum_{i=1}^n |w_i| \quad (2.25)$$

where w_i is the weight of the model. L2 regularisation is defined as follows:

$$\text{L2 Regularisation} = \sum_{i=1}^n w_i^2 \quad (2.26)$$

The main difference between L1 and L2 regularisation is that L1 regularisation can cause some of the weights to be zero because of the discontinuity at 0, whereas L2 regularisation is less likely to do so. This means that L1 regularisation can be used to perform feature selection, where some of the features are removed from the model. L2 regularisation is usually used to prevent overfitting, as it does not cause any of the weights to be zero. The regularisation term is usually added to the loss function, and is usually defined as follows:

$$\text{Regularisation} = \lambda \text{Regularisation Term} \quad (2.27)$$

where λ is a hyperparameter that controls the strength of the regularisation. The regularisation term is usually added to the loss function, and is usually defined as follows:

$$\text{Loss Function} = \text{Loss Function Without Regularisation} + \text{Regularisation} \quad (2.28)$$

So for example, if the loss function is the mean squared error, the regularised loss function with Ridge regularisation would be defined as follows:

$$\text{Ridge Regularised Loss} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^n w_i^2 \quad (2.29)$$

It is important to note that in equation 2.29, $\hat{y}$ is the predicted value of the model, and is therefore also a function of the weights w_i . This means that the regularisation term is also a function of the weights, and thus the regularisation term is also minimised during training. This also means that with MSE loss + L2 Regularisation, the same computation is performed twice, once for the MSE, and once for the regularisation term. This begs the question of the distinction between MSE and L2 regularisation. In short, both of these expressions use the L2 norm which is simply a norm defined as:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2} \quad (2.30)$$

This definition can be generalised to:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}} \quad (2.31)$$

where p is a hyperparameter. The L2 norm is thus a special case of the Lp norm, where $p = 2$.

The distinction therefore arises with the context in which the norm is used. In the loss function it is used to evaluate the model's performance, and in the regularisation term it is used to penalise the model for having large weights.

L1 regularisation is usually used in conjunction with L2 regularisation, where L1 regularisation is used to perform feature selection, and L2 regularisation is used to prevent overfitting. This is known as Elastic Net regularisation, and is defined as follows:

$$\text{Elastic Net Regularisation} = \lambda_1 \sum_{i=1}^n |w_i| + \lambda_2 \sum_{i=1}^n w_i^2 \quad (2.32)$$

Since these regularisations are applied to the weights of the model, this form of regularisation is also known as weight decay.⁹⁰ The main drawback of this (and most) regularisation techniques is that it can cause the model to underfit the data. This is usually solved by using a larger value of λ .

Other regularisation techniques can be applied to more complex deep learning models. These include dropout, where a random subset of the neurons in a layer are set to zero during training, and batch normalisation, where the input to each layer is normalised. These techniques will be discussed in more detail in Section 2.2.

2.1.6 Optimisation and Meta Learning

Optimisation is the process of finding the optimal values of the parameters of a model. This is usually done by minimising the loss function (which is a function of the model's output and hence the model parameters) with respect to the model's parameters. The most common optimisation algorithm is gradient descent - a first order iterative algorithm which, for any ML algorithm using the MSE loss function is defined as follows:

$$w_{i+1} = w_i - \eta \frac{\partial}{\partial w_i} \left(\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right) \quad (2.33)$$

where w_i is the weight of the model at iteration i , η is the learning rate, and $\hat{y}$ is the predicted value of the model. This algorithm is also known as the batch gradient descent algorithm, as it uses the entire training set to calculate the gradient. In most cases, loss is not directly a function of the model parameters, but rather a composite function of the output (which is a function of the parameters). The gradient of the loss function is thus usually calculated using the chain rule, which is defined as follows:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} \times \frac{\partial y}{\partial x} \quad (2.34)$$

where f is the function that is being differentiated, y is the intermediate variable, and x is the variable that is being differentiated with respect to.

Learning Rate

The learning rate is a hyperparameter that controls the size of the steps taken by a gradient descent algorithm. If the learning rate is too small, the algorithm will take too many iterations to converge, and if the learning rate is too large, the algorithm may skip the minima and diverge. The learning rate may be a constant or dynamic, depending on some function. One of the most common functions is the inverse square root function, which is defined as follows:

$$\eta = \frac{1}{\sqrt{t}} \quad (2.35)$$

where t is the iteration number. This function is usually used in conjunction with the Adam optimiser and decays the learning rate over time. This allows the optimiser to explore the parameter landscape by taking large steps at the beginning of training, and smaller steps towards the end of training to ensure convergence as well as to prevent getting stuck in local minima/saddles. This is also known as the learning rate schedule.

Gradient Descent Procedures

1. Initialise the weights of the model
2. Calculate the gradient of the loss function with respect to the weights
3. Update the weights in the opposite direction to the gradient (minimisation)
4. Repeat steps 2 and 3 until convergence or until the defined maximum number of iterations is reached.

Whilst gradient descent is the foundation algorithm for most other optimisers, it is usually not computationally feasible with larger datasets due to convergence times and memory limitations since the gradient must be computed for the entire training set. Instead other variant algorithms are used in practise. These include mini-batch stochastic gradient descent,⁹¹ RMSProp⁹² and Adam.⁹³ These algorithms usually converge faster than the batch gradient descent algorithm, but are less accurate. The most common optimisation algorithm used in deep learning is Adam, which is a combination of the RMSProp and Momentum optimisation algorithms. This algorithm is usually faster than the other optimisation algorithms, and is more accurate.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a variant of the gradient descent algorithm where the gradient is calculated using a subset of the training set. This reduces the computational complexity of the algorithm, by randomly sampling a subset of the training set. This is known as mini-batch stochastic gradient descent, and is usually done in batches of 16-512 samples. This algorithm is usually faster than the batch gradient descent algorithm, but is less accurate. As we deal with batches and mini-batches of data, it is useful to introduce contextual terms to represent the average loss over a batch of data, and the entire training dataset. These are known as the batch loss and the epoch loss or cost respectively.

An important addition to the SGD algorithm is the use of a momentum term. This term has two main uses: to increase the speed of convergence by taking larger steps in the direction of the gradient vector, and to allow better learning with noisy data. This is done by adding a fraction of the previous gradient to the current gradient - in essence, it can be thought of as an exponential moving average across the data samples. SGD with momentum can be defined as follows:

$$v_{i+1} = \beta v_i + \eta \frac{\partial}{\partial w_i} \left(\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right) \quad (2.36)$$

where v_i is the momentum term at iteration i , β is the momentum hyperparameter, and η is the learning rate, which would result in a weight update of:

$$w_{i+1} = w_i - v_{i+1} \quad (2.37)$$

where w_i is the weight of the model at iteration i .

Figure 2.3 shows a visualisation of the effect of momentum on the stochastic gradient descent algorithm. Momentum gives a push towards the centre of the valley, and allows the algorithm to converge faster. However, if the momentum hyperparameter is too large, the algorithm

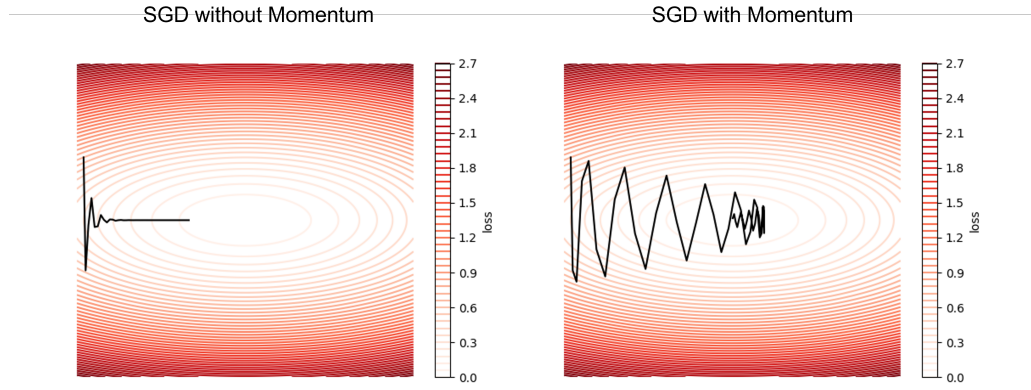


Figure 2.3: A visualisation of the effect of momentum on the stochastic gradient descent algorithm.

may skip the minima and either have to go back on itself, or diverge. This is known as the momentum trap. The momentum hyperparameter is usually set to 0.9, and is usually used in conjunction with the Adam optimiser.

RMSProp

Root Mean Square Prop (RMSProp) is a variant of the SGD where the learning rate is adjusted based on the average of the squared gradients of the previous iterations. This is usually done to reduce the learning rate as the algorithm converges, and is usually done by calculating the average of the squared gradients of the previous iterations. RMSProp can be defined as follows:

$$s_{i+1} = \beta s_i + (1 - \beta) \frac{\partial}{\partial w_i} \left(\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right)^2 \quad (2.38)$$

where s_i is the average of the squared gradients at iteration i , β is the momentum hyperparameter, and η is the learning rate, which would result in a weight update of:

$$w_{i+1} = w_i - \frac{\eta}{\sqrt{s_{i+1}}} \frac{\partial}{\partial w_i} \left(\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right) \quad (2.39)$$

where w_i is the weight of the model at iteration i . RMSProp is known as an adaptive learning rate algorithm, as the learning rate is adjusted based on the average of the squared gradients

of the previous iterations. Other adaptive learning rate algorithms include AdaGrad⁹⁴ and Adam.⁹³

Adam

Adam is a combination of the RMSProp and Momentum optimisation algorithms. This algorithm is usually faster than the other optimisation algorithms and finds good solutions, however recent works suggest that it may not find the best solution to the optimisation problems, and in some cases its solution is worse than SGD.⁹⁵ However for most problems, Adam is the most commonly used optimisation algorithm. For MSE loss, Adam can be defined as follows:

$$m_{i+1} = \beta_1 m_i + (1 - \beta_1) \frac{\partial}{\partial w_i} \left(\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right) \quad (2.40)$$

$$s_{i+1} = \beta_2 s_i + (1 - \beta_2) \frac{\partial}{\partial w_i} \left(\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \right)^2 \quad (2.41)$$

where m_i is the momentum term at iteration i , s_i is the average of the squared gradients at iteration i , β_1 and β_2 are the momentum hyperparameters, and η is the learning rate, which would result in a weight update of:

$$w_{i+1} = w_i - \frac{\eta}{\sqrt{s_{i+1}} + \epsilon} m_{i+1} \quad (2.42)$$

where w_i is the weight of the model at iteration i , ϵ is a small constant to prevent division by zero, and m_{i+1} and s_{i+1} are the updated momentum and average of the squared gradients respectively. Adam is known as an adaptive learning rate algorithm, as the learning rate is adjusted based on the average of the squared gradients of the previous iterations.

Meta Optimisation Algorithms

As with the optimisation algorithms, meta optimisation algorithms go a step further in terms of abstraction and can be used to find the best hyperparameters for the optimisation algorithms. These algorithms are usually used to find the best learning rate, momentum, and other hyperparameters for the optimisation algorithms and for the model. The most commonly used meta optimisation algorithm is grid search, which is a brute force algorithm that tries all possible combinations of hyperparameters. This algorithm is the most commonly used one as it is simple to implement, it is however very crude and offers no guarantee of any performance improvements. The next most commonly used meta optimisation technique is random search. This is also a brute force algorithm for hyperparameter optimisation. However unlike grid search, random search does not try all possible combinations of hyperparameters, instead, it tries a random set of combinations of hyperparameters.

More recently, with the acceleration of computational power, ML algorithms are being used to optimise the hyperparameters of other ML models. One example of such method is Bayesian optimisation. This method uses Gaussian Processes to optimise the hyperparameters of a model.⁹⁶ In Gaussian Process models, the dependent variable y is modelled as a function of the independent variable x :

$$y = f(x) + \epsilon \quad (2.43)$$

where $f(x)$ is the function that models the relationship between x and y , and ϵ is the noise term. The Gaussian Process model assumes that $f(x)$ is a Gaussian distribution, and that the noise term ϵ is also a Gaussian distribution. The Gaussian Process model can be defined as follows:

$$f(x) \sim \mathcal{N}(\mu(x), \Sigma(x)) \quad (2.44)$$

The kernel function $\kappa(x, x')$ is used to define the covariance between the Gaussian distributions.

One common kernel is the squared exponential kernel, which is defined as follows:

$$\kappa(x, x') = \sigma^2 \exp\left(-\frac{|x - x'|^2}{2l^2}\right) \quad (2.45)$$

where σ is the standard deviation of the Gaussian distribution, and l is the length scale of the Gaussian distribution. The Gaussian Process model can be used to model the relationship between the hyperparameters and the loss function. This is done by defining the hyperparameters as the independent variable x , and the loss function as the dependent variable y .

More recently, Reinforcement Learning has been used as a method for designing neural architectures. This is known as Neural Architecture Search (NAS)⁹⁷ where the neural architecture is modelled as a Reinforcement Learning problem. The neural architecture is modelled as an MDP where the state is the neural architecture, the action is the addition of a new layer, and the reward is the loss function. NAS is closely related to Meta hyperparameter optimisation and which are all part of a newly developed subfield of ML known as automated machine learning (AutoML).⁹⁸

2.1.7 Data Augmentation

Data Augmentation is the art of taking the training data you have, and somehow creating more data from your dataset to train your model. In most cases, training ML models on larger datasets results in better performance, so augmenting your dataset is a good way to improve the performance of your model. There are many ways to augment your dataset. One of the most common is by adding noise to your samples. For example, with images, one can rotate, flip, brighten, darken and add noise to the images. This can also make the model more robust to the noise in the real world. One can also go down the path of theoretically generating or simulating new data *in silico*. For example, for a self driving ML model, one can simulate different environments for a driverless car.^{99–101} Another more recent technique is to use GANs to generate new data.¹⁰² GANs are a type of ML model that can generate new data that is similar to the training data. This is done by training two models, a generator and a

discriminator. The generator is trained to generate new data that is similar to the training data, and the discriminator is trained to distinguish between the real data and the generated data. The generator is trained to fool the discriminator, and the discriminator is trained to distinguish between the real data and the generated data. This is done iteratively until the generator is able to generate data that is indistinguishable from the real data. This is a very powerful technique, and can be used to generate new data for many different applications.

One example of augmentation in cheminformatics is to exploit the redundancies given by the projection of a 2D molecular graph into a 1D SMILES string. This is achieved by selectively varying the graph traversal order, which is equivalent to varying the order of the SMILES string. Figure 2.4 shows an example of this augmentation applied to the 2D molecular graph of aspirin.

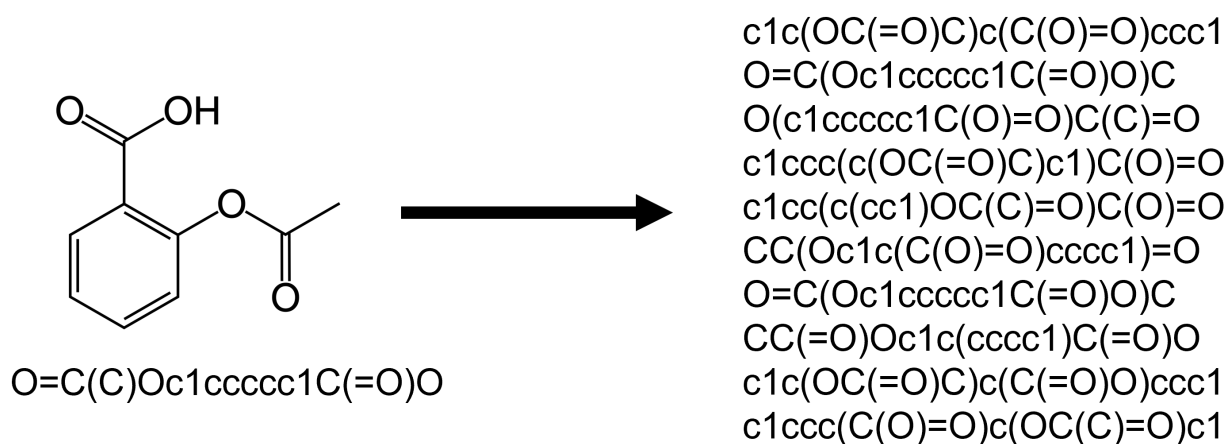


Figure 2.4: An example of SMILES augmentation. The canonical SMILES string of aspirin is shown on the left. The SMILES string is then augmented by varying the order of the 2D graph traversal.

Other forms of molecular augmentation exist, these also include molecular generator GANs which can predict a core scaffold change that is not likely to change the function of the molecule and generate variants based off that seed. These can also be further augmented using descriptors and representations such as SMILES, Morgan fingerprints, or RDKit fingerprints.

2.2 Deep Learning

In this section, various Deep Learning (DL) architectures will be discussed. Deep Learning is a subset of ML algorithms which contain one or more hidden layers. Where a hidden layer is a layer of nodes that are not the input or output layer. More generally it is some intermediate latent space within a model. Previously, DL models were generally unfavourable due to the high computational and data requirements needed for learning, however, now with the advent of GPUs and the availability of large datasets, DL models are now becoming of the most common model types due to their high performance.

2.2.1 Feed Forward Neural Networks

The first Deep Learning architecture was the Multilayer Perceptron (MLP), which was invented in 1969 by Frank Rosenblatt.⁴³ The MLP is a feedforward neural network, which is a type of neural network where the information flows in one direction, from the input layer to the output layer. The MLP was built upon initial works on the perceptron by Rosenblatt, whereby more affine layers were introduced in between the input and output layers.

Figure 2.5 shows an example of a Deep Feed Forward Neural Network (FFNN) with one input layer, two hidden layers, and one output layer. For this FFNN given an input feature vector $\mathbf{x}$, the output is given by the following equation:

$$\mathbf{y} = \sigma(\mathbf{W}_3\sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)) + \mathbf{b}_3 \quad (2.46)$$

Where $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{W}_3$ are the weight matrices of the first, second hidden layers and output layer respectively, $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{b}_3$ are the bias vectors of the first and second hidden layers and output layer respectively, and σ is the activation function. The activation function is a non-linear function that is applied to the weighted sum of the inputs to the node.

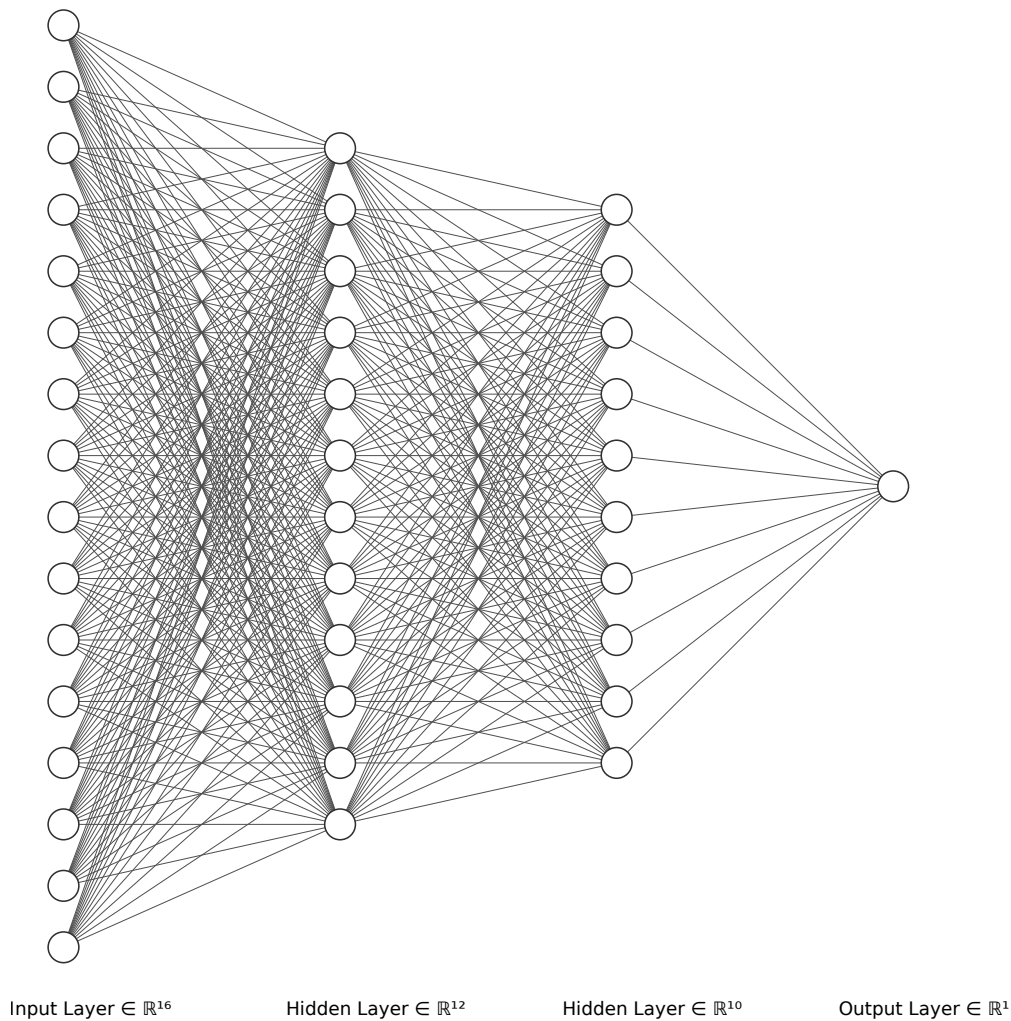


Figure 2.5: A Multilayer Perceptron with 1 input layer (16-dim), 2 hidden layers (12-dim and 10-dim) and 1 output layer (1-dim).

Activation Functions

Activation functions are non-linear functions that are applied to the weighted sum of the inputs to the node. The activation function is used to introduce non-linearity into the network, which is required for the network to be able to learn complex functions. It has been shown that the introduction of activations to the network introduce non-linearities and thus allows deep neural networks to be universal function approximators - that is they have been proven to be able to estimate any function.¹⁰³

Table 2.1 shows some of the most commonly used activation functions. The sigmoid function is a smooth function that is bounded between 0 and 1. The tanh function is similar to the sigmoid

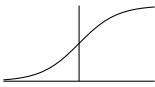
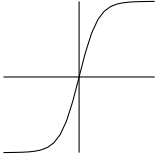
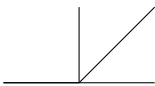
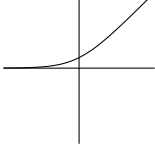
Name	Function	Derivative	Figure
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))^2$	
tanh	$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	
ReLU	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$	
Softmax	$f(x) = \frac{e^x}{\sum_i e^x}$	$f'(x) = \frac{e^x}{\sum_i e^x} - \frac{(e^x)^2}{(\sum_i e^x)^2}$	
Softplus	$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{e^x}{1+e^x}$	

Table 2.1: Non-linear activation functions.

function, but it is centred around 0 and is asymptotic at 1 and -1. The Rectified Linear Unit (ReLU) function is a piecewise linear function that is zero for negative values and the identity function for positive values. The softmax function is used in the output layer of a classification network. The softplus function is a smooth approximation of the ReLU function.

It is important to note that whilst all of these functions are non-linear, they are continuously differentiable (with the caveat that is ReLU) and thus can be used in backpropagation. However, there are other activation functions that are non-linear but are not continuously differentiable, such as the maxout function. Maxout is a generalisation of ReLU, however not all maxout functions are suitable for backpropagation, but most can be used in other algorithms such as the genetic algorithm.

Backpropagation

Backpropagation is an algorithm that is used to train neural networks. It is an iterative algorithm that is used to update the weights and biases of the network.⁴⁷ It utilises the chain rule to apply gradient descent and other optimisers to the network parameters. The algorithm

is as follows:

1. Perform a forward pass through the network to calculate the output of the network.
2. Calculate the cost of the batch - cost, C , refers to the average difference between the output of the network and the expected output for the round (i.e. the average loss over the batch or the training set).
3. Use an optimisation algorithm to calculate the gradient of the cost function with respect to the network parameters.
4. Backpropagate the cost gradient through the network to calculate the gradient of the cost function with respect to the weights and biases at each layer.
5. Update the weights and biases using the calculated gradients.

In order to backpropagate the errors through the network, the layers of the network are traversed in reverse order. We define a node k in layer l to have output y_k^l and input $z_k^l = y_j^{l-1}$, where j is the index of the connected node in layer $l - 1$. The error of the node is defined as δ_k^l . For each node in layer l , the weights connecting it to the nodes in layer $l - 1$ are defined as w_{jk}^l . The bias of the node is defined as b_k^l . The output of any node in any layer is thus:

$$y_k^l = \sigma\left(\sum_j w_{jk}^{l-1} a_j^{l-1} + b_k^l\right) \quad (2.47)$$

where σ is the activation function of the node. The activation can thus be applied to a set of nodes by considering a propagation matrix $p^l = (W^l)^T y^{l-1} + b^l$ making an entire layer's output:

$$y^l = \sigma(p^l) \quad (2.48)$$

The cost delta can thus be computed within the forward pass by computing the delta in the propagation matrix $\Delta y^l = \sigma(p^l) - \sigma(p^l + \Delta p^l)$.

The error gradient can then be computed for each parameter within the network:

$$\delta_k^l = \frac{\partial C}{\partial y_k^l} = \frac{\partial C}{\partial \sigma_k^l} \cdot \frac{\partial \sigma_k^l}{\partial (\sum_i W_{jk}^l y_j^{l-1} + b_k^l)} \quad (2.49)$$

Once the error gradient has been computed for each parameter, the weights and biases can be updated using an optimisation algorithm as discussed in Section 2.1.6. As each output is a composite function of the inputs, previous outputs and/or activation function, the chain rule is used for the gradient computation such that it can be backpropagated.

For all nodes in the output layer, L , the error is given by:

$$\delta^L = \nabla_y C \odot \sigma'(p^L) \quad (2.50)$$

Where $\nabla_y C$ is the vector of cost derivatives per output and $\odot$ denoting element-wise matrix multiplication. During Backpropagation the gradient traverses the network in reverse order, starting from the output layer. Therefore for any layer, l , the error is given by:

$$\delta^L = (W^{l+1} \delta^{l+1}) \odot \sigma'(p^L) \quad (2.51)$$

The gradients are then computed recursively for each layer, starting from the output layer. As the network gets deeper, so does the application of the chain rule. This can become problematic for really deep and recurrent networks.

Summary Of Backpropagation Equations

$$\delta_k^l = \frac{\partial C}{\partial y_k^l} = \frac{\partial C}{\partial \sigma_k^l} \cdot \frac{\partial \sigma_k^l}{\partial (\sum_i W_{jk}^l y_j^{l-1} + b_k^l)} \quad (2.52a)$$

$$\delta^L = \nabla_y C \odot \sigma'(p^L) \quad (2.52b)$$

$$\delta^L = (W^{l+1} \delta^{l+1}) \odot \sigma'(p^L) \quad (2.52c)$$

Whilst FFNNs provide the base framework for deep learning architectures, there are many applications for which their simplistic, fixed length architecture can not suffice. Some such applications include:

1. Spatial data - although images can be flattened into a single vector input for an FFNN, spatial information is lost in the flattening process. Images are also getting larger (higher resolution, colour depth, etc.) it can quickly become computationally infeasible to flatten and perform standard matrix multiplication as with an FFNN.
2. Temporal data - For each sample passed through an FFNN, the network considers information within the current sample only as it has no memory of previous samples. Thus if a prediction for a current sample is based on one or more past samples (e.g. in a time series, or any sequence such as a sentence), a FFNN would not be able to make a valid prediction.

2.2.2 Recurrent Neural Networks

Recurrent networks attempt to solve the temporal limitations of a FFNN by maintaining a “memory” of the previous sample provided to the network and combining that with the current sample input.^{48,104} One way to think about this is to consider a hidden layer output of a Deep Neural Network (DNN) being fed back as input to the same network along with the input of the next sample, resulting in a cyclical architecture as shown in Figure 2.6. To better visualise RNNs, an unrolled representation of the network (Figure 2.6) shows the inputs/outputs/memory at each time step. This represents the simplest form of an RNN.

Backpropagation Through Time (BPTT)

The backpropagation algorithm can be applied to RNNs by unrolling the network and applying the backpropagation algorithm to each time step. This is known as Backpropagation Through Time (BPTT). The unrolled network is shown in Figure 2.6.

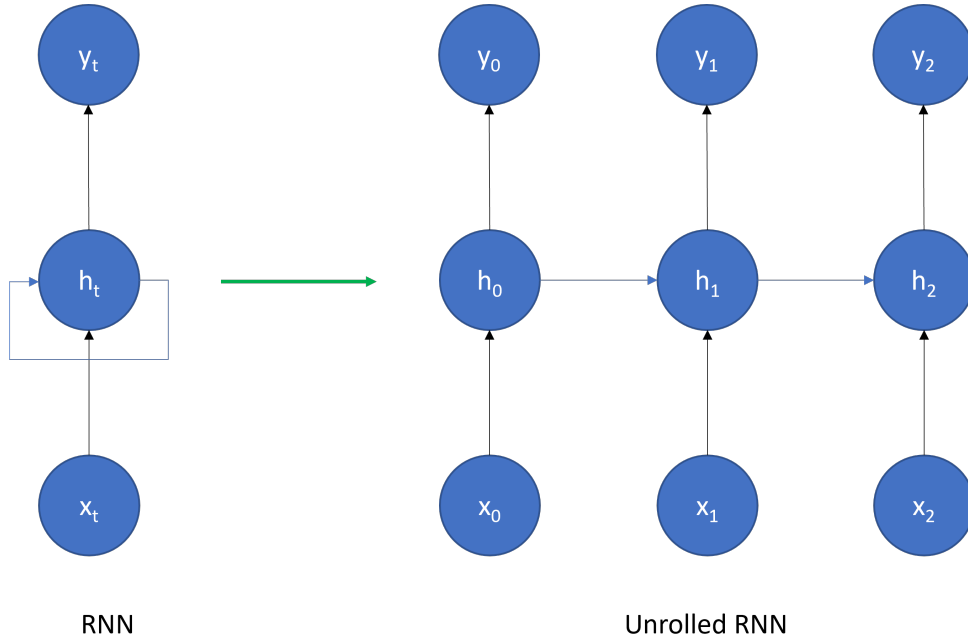


Figure 2.6: A simple RNN architecture

It is important to note that with RNNs there are two dimensions of “depth” within the network.

1. The depth of the network (i.e. the number of layers)
2. The depth of the unrolled network (i.e. the number of time steps)

BPTT therefore must backpropagate along both dimensions.³¹ This can be computationally expensive as the number of time steps increases. To reduce the computational cost of BPTT, the network can be truncated to a fixed number of time steps. This is known as Truncated Backpropagation Through Time (TBPTT).¹⁰⁵ The network is then unrolled for the number of time steps and backpropagated through. The network is then truncated and unrolled again for the next set of time steps. This process is repeated until the entire dataset has been processed.

For a sample propagation through an RNN, the hidden vector at time step t is given by:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \quad (2.53)$$

Where h_t is the hidden vector at time step t , W_{hh} is the weight matrix between the hidden vectors, W_{xh} is the weight matrix between the input vector and the hidden vector, b_h is the

bias vector for the hidden vector and σ is the activation function. The output vector at time step t is given by:

$$y_t = \sigma(W_{hy}h_t + b_y) \quad (2.54)$$

Where y_t is the output vector at time step t , W_{hy} is the weight matrix between the hidden vector and the output vector and b_y is the bias vector for the output vector.

It is important to note that the hidden vector at time step t is a function of the hidden vector at time step $t - 1$ and the input vector at time step t . This means that the hidden vector at time step t is a function of the entire input sequence up to time step t . This is the reason why RNNs are able to maintain a memory of the previous samples. For each timestep, the hidden state is weighted by the weight matrix W_{hh} , these weights are shared across all hidden layers in the temporal domain. The output of the network is then computed as the dot product between the output layer weight matrix W_{hy} and the hidden state vector h_t - thus there is a network output for every timestep. The gradients of the network are computed recursively for each time step, starting from the output layer. As the network gets deeper, or longer, so does the application of the chain rule. This can become problematic for really deep and recurrent networks.

Memory, Vanishing gradients and Exploding gradients

As an RNN gets longer, since the chain rule is used along with activation functions over many layers and timesteps, the gradients can become very small or very large. This is known as the vanishing/exploding gradient problem. A simple solution would be using a different activation function such as the hyperbolic tangent function, which has a range of $[-1, 1]$ and is thus less likely to suffer from the exploding gradient problem. This problem is also enhanced by the limited precision of computers to store floating point numbers.

As previously determined, an RNN network output at time t is dependent on the hidden state at time $t - 1$. Thus for the first timestep a hidden state for the -1 th step must be generated.

In most cases, this is a vector of zeros, or a beginning of sentence (BOS) token. In the context of RNNs for NLP, each input vector, or token embedding, is a member of the set of tokens known as the vocabulary. Each token within a vocabulary is a vector representation of either words, subwords, characters or auxiliary tokens.

Since the weight matrix W_{hh} is shared across all timesteps, when backpropagating, the gradient of the cost with respect to the weight matrix W_{hh} is the sum of the gradients of the loss function with respect to the weight matrix W_{hh} at each timestep.

$$\frac{\partial C}{\partial W_{hh}} = \sum_{t=1}^T \frac{\partial C}{\partial W_{hh}} \quad (2.55)$$

since W_{hh} is a function of the hidden states and inputs, at each timestep, the cost gradient with respect to the weights must be computed recursively across the composite functions within each timestep. This is what differentiates BPTT from standard backpropagation. For example, the second timestep gradient of a three time step deep network would be:

$$\frac{\partial C_2}{\partial W_{hh}} = \frac{\partial C_2}{\partial y_2} \frac{\partial y_2}{\partial h_2} \frac{\partial h_2}{\partial \sigma} \frac{\partial \sigma}{\partial a_2} \quad (2.56)$$

where C_2 is the cost at time step 2, y_2 is the output at time step 2, h_2 is the hidden state at time step 2, σ is the activation function and a_2 is the propagation vector at time step 2, $a_2 = (W_{xh}x_2 + W_{hh}h_1 + b_h)$. Which can be generalised to any timestep:

$$a_t = (W_{xh}x_t + W_{hh}h_{t-1} + b_h) \quad (2.57)$$

Due to the recurrent nature of RNNs and the number of repeated multiplications of composite functions, the scaling of the cost partial is exponential with timesteps, meaning that the gradients being very small or very large is a probable problem and does occur often.

$$\frac{\partial C_t}{\partial W_{hh}} \propto |W_{hh}|^t \cdot \left| \frac{\partial \sigma}{\partial a} \right|^t \quad (2.58)$$

As mentioned previously, bounded activation functions such as the hyperbolic tangent function help reduce the likelihood of exploding gradients compared to unbounded activations such as ReLU. Other simple solutions include gradient clipping and adaptive learning rates to ensure the gradients do not become too large or too small.

Truncated BPTT is also an option for both exploding and vanishing gradients, however longer term information is lost. This is a problem for tasks such as language modelling, where the context of the sentence is important. The Long Short-Term Memory (LSTM)¹⁰⁶ and Gated Recurrent Units (GRU)¹⁰⁷ networks were developed to address this problem.

Long Short-Term Memory (LSTM)

The main idea behind LSTM networks¹⁰⁶ is that they are able to selectively forget information, depending on what they deem important for future time steps. This is done by learned weights (gates) which are used to determine what information is important to keep and what information is not. In the LSTM architecture there are three gates:

- Input gate: determines what information is important to keep
- Forget gate: determines what information is important to forget
- Output gate: determines what information is important to output

For the output gate, the state of the memory cell, c_t , is multiplied by the output gate, o_t , to determine what information is important to output. The output gate is a function, akin to a FFNN and is defined as:

$$o_t = \sigma(W_{ho}h_{t-1} + W_{xo}x_t + b_o) \quad (2.59)$$

where W_{ho} is the weight matrix between the hidden state and the output gate, W_{xo} is the weight matrix between the input and the output gate and b_o is the bias vector for the output gate.

The output of the output gate is then multiplied by the activation function, σ , to determine what information is important to output. The activation function is a sigmoid function. The hidden state output then becomes:

$$h_t = o_t \odot \tanh(c_t) \quad (2.60)$$

The hidden state is then used to compute the output of the network, y_t .

The forget gate occurs within the hidden state of the unrolled network at each time step. It is defined as:

$$f_t = \sigma(W_{hf}h_{t-1} + W_{xf}x_t + b_f) \quad (2.61)$$

where W_{hf} is the weight matrix between the hidden state and the forget gate, W_{xf} is the weight matrix between the input and the forget gate and b_f is the bias vector for the forget gate. The output of the forget gate is then activated by activation function, σ , to determine what information is important to forget. The hidden state output then becomes:

$$h_t = f_t \odot c_t \quad (2.62)$$

The input gate processes the input before passing it to the cell state. In this case, the “input” is the input at the current time step, and the hidden state of the previous time step. As such, the input gate is two-fold and provides a new proposed cell state $\tilde{c}_t$ which is then modulated. The input gate modulation and new cell state are defined as:

$$i_t = \tanh(W_{hi}h_{t-1} + W_{xi}x_t + b_i) \quad (2.63)$$

$$\tilde{c}_t = \tanh(W_{hc}h_{t-1} + W_{xc}x_t + b_c) \quad (2.64)$$

where W_{hi} is the weight matrix between the hidden state and the input gate, W_{xi} is the weight matrix between the input and the input gate and b_i is the bias vector for the input gate. Combining these too, the actual cell state then becomes:

$$c_t = \tilde{c}_t \odot i_t + c_{t-1} \odot f_t \quad (2.65)$$

Figure 2.7 shows the unrolled LSTM architecture with the locations of each of the gates.

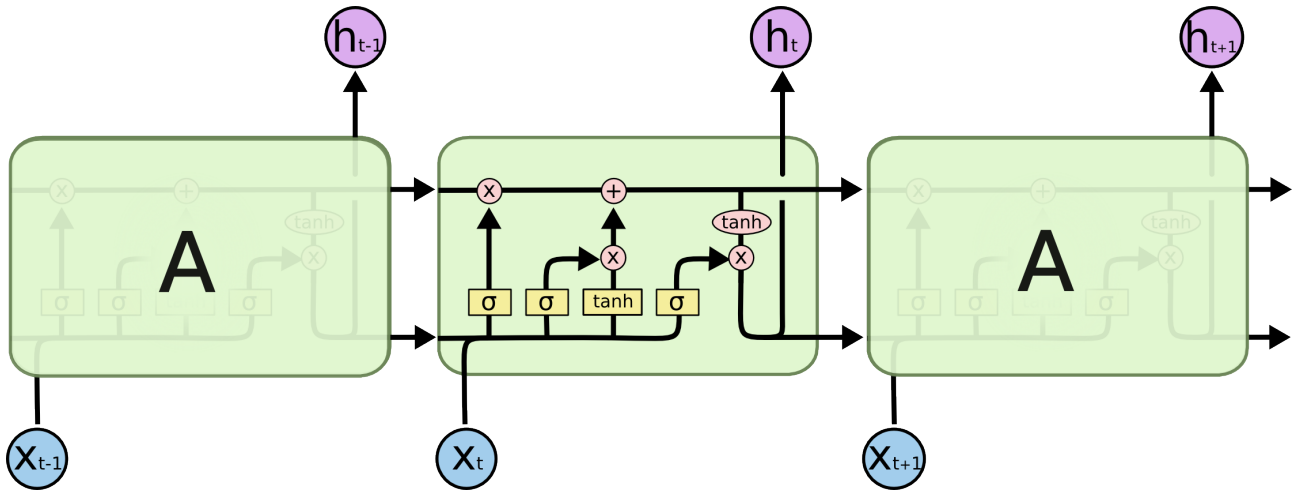


Figure 2.7: Unrolled LSTM architecture. Image source¹⁰⁸

2.2.3 Convolutional Neural Networks

One of the main initial goals of neural networks (the perceptron) was for visual tasks and image recognition. However, the perceptron was not able to achieve this goal due to the large number of parameters required to train the network. The convolutional neural network (CNN) was developed to address this problem.⁵⁰ CNNs are a type of feed-forward neural network, where the input is a 2D (or 3D) image. The CNN architecture is composed of convolutional layers, pooling layers, normalisation layers and affine layers. The convolutional layers are the main component of the CNN architecture and are responsible for extracting features from the input image. The pooling layers are used to reduce the dimensionality of the input image, which is important for reducing the number of parameters in the network. The fully connected layers are used to perform classification/regression on the input image.

Convolutional Layers

A convolution kernel is a small matrix that is used to extract features from the input image. Its size must be smaller than or equal to the size of the input image and its depth must be identical to that of the image as the dot product is computed. The convolution kernel is applied to the (usually padded) input image, and the output of the convolution is a 2D feature map. The convolution kernel is applied to the input image by sliding it across the input image, and performing element-wise multiplication and summing the result. The convolution kernel is then moved to the next position and the process is repeated. The convolution kernel is moved across the input image in a step size of 1 or more pixels. This is usually referred to as the stride. The size of the feature map is dependent on the size of the input image and the size of the convolution kernel. The size of the convolution kernel is a hyperparameter of the network and is usually an odd number, such as 3x3 or 5x5. The convolution kernel is usually initialised with random values, and is then trained during the backpropagation process. The convolution kernel is also known as a filter. In a CNN a kernel convolution layer can be defined by:

$$f(x, y) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} w_{i,j} x_{i,j} \quad (2.66)$$

where $f(x, y)$ is the output of the convolution, $w_{i,j}$ is the weight of the convolution kernel at position i, j and $x_{i,j}$ is the input image at position i, j .

At each convolutional layer, many differently weighted filters are applied to the input image. The output of each filter is a feature map. The output of the convolutional layer is then the concatenation of all the feature maps. Figure 2.8 shows a sequence of convolutional layers. The first convolutional layer has 6 5x5x3 filters, and the output of the convolutional layer is the concatenation of the 6 feature maps. The second convolutional layer has 10 5x5x6 filters, and the output of the convolutional layer is the concatenation of the 10 feature maps. The Z-dimension (depth) of the filter must match that of the input to the filter. Activation functions such as ReLU are also applied to the activation maps.

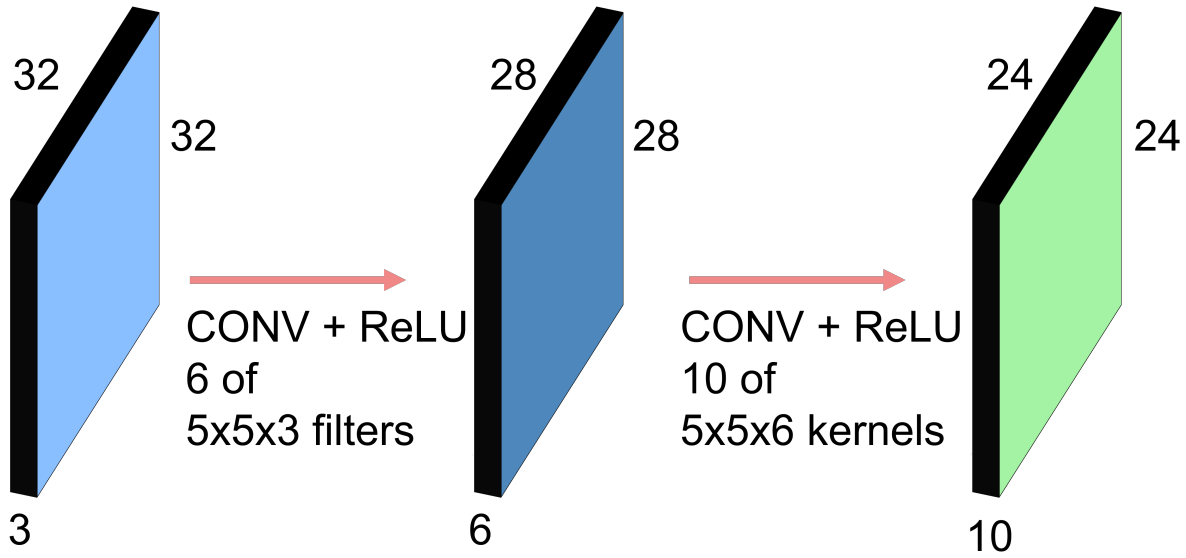


Figure 2.8: A sequence of depth-wise convolutional layers, with a stride of 1 and no padding.

The dimensionality of a convolution layer output depends on 4 variables: The input dimensions, the padding, the stride, the number of filters and the size of the filters. The output dimensions are calculated as:

$$\begin{aligned}
 \text{Output Width} &= \frac{\text{Input Width} - \text{Filter Width} + 2 \times \text{Padding}}{\text{Stride}} + 1 \\
 \text{Output Height} &= \frac{\text{Input Height} - \text{Filter Height} + 2 \times \text{Padding}}{\text{Stride}} + 1 \\
 \text{Output Depth} &= \text{Number of Filters}
 \end{aligned} \tag{2.67}$$

These convolutions are usually exploited to reduce the dimensionality along the x-y dimensions such that an Autoencoder like architecture can be used to reduce the dimensionality of the input image. This is useful for tasks such as image classification, where the input image is a high dimensional vector.

Pooling Layers

There are two common types of pooling usually applied to the feature maps post convolution. These are max pooling and mean pooling.

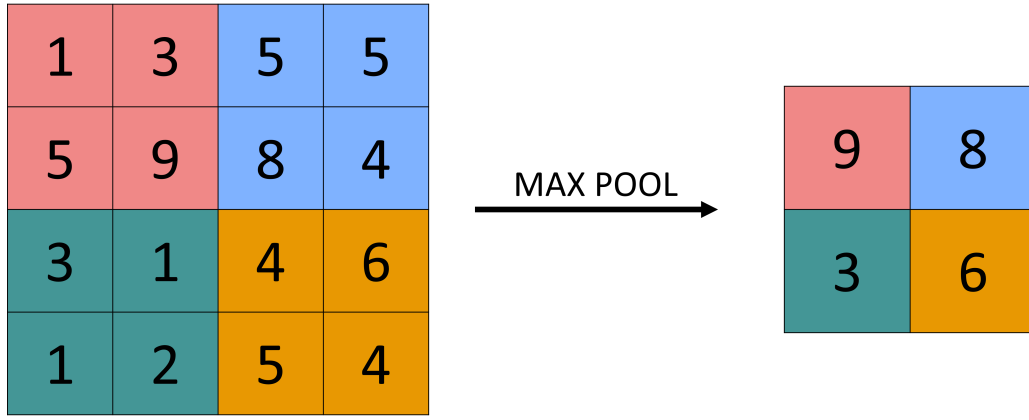


Figure 2.9: A sample of Max Pooling with a 2x2 filter and stride 2

Max pooling is the most common type of pooling and is defined as:

$$p(x, y) = \max_{i=0}^{k-1} \max_{j=0}^{k-1} x_{i,j} \quad (2.68)$$

where $p(x, y)$ is the output of the pooling, $x_{i,j}$ is the input image at position i, j and k is the size of the pooling kernel. The pooling kernel is usually a square matrix, such as 2x2 or 3x3 and is moved across the feature map at a given stride. Figure 2.9 shows max pooling with a 2x2 filter and a stride of 2. This kind of pooling is common because taking the max value of the feature map is more likely to trigger a signal in the activation function, thus clearly defining notable features within an image which would be useful when passed down to the FFNN for classification. Mean pooling is defined as:

$$p(x, y) = \frac{1}{k^2} \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} x_{i,j} \quad (2.69)$$

Mean pooling is less common than max pooling, but is used in some applications, usually when the input image is a 3D/4D image, such as a video. Both pooling techniques' main purpose is to make the convolution outputs more manageable.

2.2.4 Seq2Seq, Autoencoding and Autoregressive models

An Autoencoding model is a general class of neural models whereby some latent representation (an encoding) of the input is learned and this representation is then used for the tasks, whereas an Autoregressive model generates an output by sequentially predicting the next element in the sequence. The latent representation of the input may be learned, but is not usually used more than temporarily in this case. A Seq2Seq (Sequence to Sequence)¹⁰⁹ model is a combination of both whereby one sequence is transduced into another sequence that is of fixed length proportional to the input sequence. Therefore the main difference between an Autoencoding model and an Autoregressive model is based on the task and the training of the system, not on the architecture. The idea that each prediction is used cyclically for the next defines an Autoregressive model, whereas if the task is to learn encoded representations, then it is an autoencoder.

2.2.5 Message Passing Neural Networks

Message Passing Neural Networks (MPNNs) build on Graph Neural Networks (GNNs).⁵⁴ In a graph, each node is interdependent (i.e. each node may be interconnected with one or all other nodes). MPNNs have shown impressive performance in predicting molecular energies at the Density Functional Theory (DFT) level.⁵⁴ The MPNNs are based on the idea that the nodes in a graph can be represented as a vector, and the edges between nodes can be represented as a matrix. The nodes are then updated by passing messages between them. The messages are calculated by multiplying the edge matrix by the node vector. The messages are then summed and passed to the next node. This process is repeated until the nodes have converged. The nodes are then passed through a feed forward neural network to produce the output. The MPNNs are able to learn the structure of the graph and the interactions between the nodes. This is useful for tasks such as predicting molecular energies, where the interactions between the atoms are important. For an undirected graph, $G = (V, E, X, W)$, where V is the set of nodes, E is the set of edges, X is the set of node features and W is the set of edge features.

The MPNN is defined as:

$$\begin{aligned}
 \text{MPNN}(G) &= \text{FFNN}(\text{Update}(\text{Message}(\text{Read}(\text{Init}(G)))))) \\
 \text{Init}(G) &= \text{Concat}(X, I_d) \\
 \text{Read}(G) &= \text{Concat}(X, \text{Agg}(\text{Message}(\text{Init}(G)))) \\
 \text{Message}(G) &= \text{Concat}(X, \text{Agg}(\text{Read}(G))) \\
 \text{Update}(G) &= \text{Concat}(X, \text{Agg}(\text{Message}(\text{Read}(G)))) \\
 \text{Agg}(M) &= \text{Sum}(M)
 \end{aligned} \tag{2.70}$$

where d is the dimensionality of the node features, I_d is the identity matrix of dimension d and FFNN is a feed forward neural network. The MPNNs are able to learn the structure of the graph and the interactions between the nodes. This is useful for tasks such as predicting molecular energies, where the interactions between the atoms are important. This is known as the message passing algorithm, which involves: initialisation, aggregation and transformation steps. The aggregation step is what drives this algorithm to be suitable for molecular prediction as it makes the algorithm permutation invariant and node equivariant.

A node embedding (some relevant node-level feature representation which reflects the properties of each node in the graph) can be represented by a vector $h_v^{(t)}$, where v is the node and t is the iteration. The initialisation step is defined as:

$$h_v^{(0)} = x_v, \forall v \in V \tag{2.71}$$

A message is passed from node v to node u through aggregation and transformation steps. The aggregation step is defined as:

$$m_v^{(t)} = f_{agg}(h_v^{(t-1)}, \{h_u^{(t-1)} : u \in \mathcal{N}(v)\}) \quad 1 \leq t < T \tag{2.72}$$

and the transformation step:

$$h_v^{(t)} = f_{Update}(h_v^{(t)}, m_v^{(t)}) \quad (2.73)$$

Many variants of aggregation functions f_{agg} and transformation functions f_{Update} have been proposed. Some of the more common functions are a simple sum + multi-layer perceptron (MLP):

$$m_v^{(t)} = \sum_{u \in \mathcal{N}(v)} h_u^{(t)} \quad (2.74)$$

$$h_v^{(t+1)} = \sigma(W^{(t)} \cdot m_v^{(t)}) \quad (2.75)$$

Or a mean aggregate + MLP:

$$m_v^{(t)} = \frac{1}{|\mathcal{N}(v) + 1|} \sum_{u \in \mathcal{N}(v)} h_u^{(t)} \quad (2.76)$$

$$h_v^{(t+1)} = \sigma(W^{(t)} \cdot m_v^{(t)}) \quad (2.77)$$

In some MPNNs, there is a readout function which is used to produce the output, given all the graph information at iteration T . The readout function can be neural units which can be defined as:

$$R = \sum_{v \in V} \sigma(i(h_v^{(T)}, h_v^0)) \odot (j(h_v^{(T)})) \quad (2.78)$$

where i and j are neural units and $\odot$ is the element-wise multiplication.

This allows MPNNs to do a variety of tasks including: considering the entire graph state to make a prediction - usually used for chemical property or energy predictions. Or, predicting

features/properties about a specific node within a system. This is useful for tasks such as predicting the properties of a specific atom/functional group in a molecule.

2.2.6 The Transformer Model

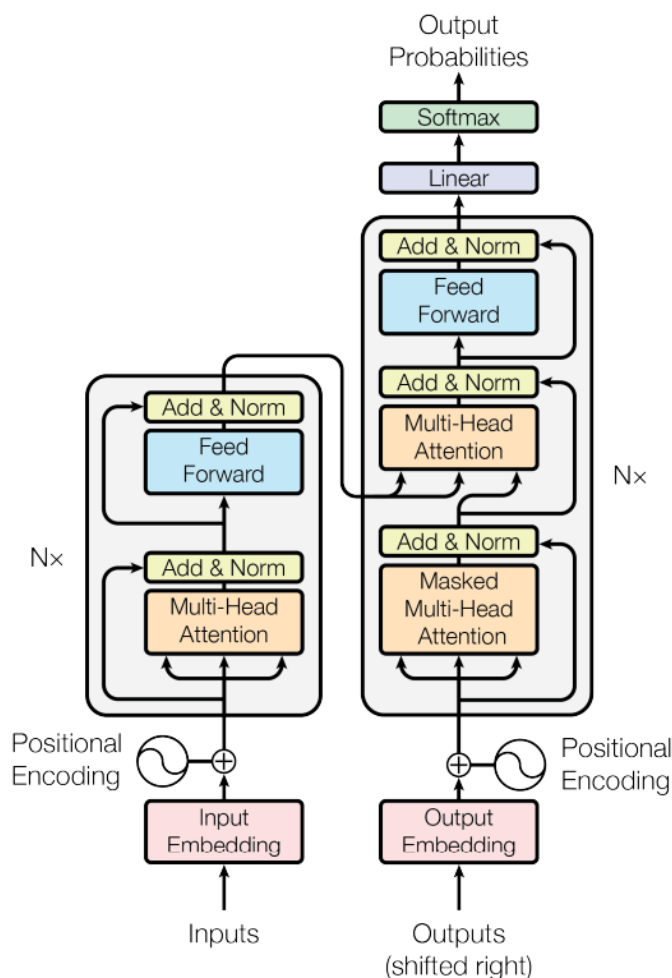


Figure 2.10: The Transformer Model Architecture.⁵

The Transformer model is a neural network architecture that is based on the attention mechanism. It was introduced in the paper *Attention is all you need*⁵ and has been shown to be very effective in a variety of tasks. The initial Transformer model is a sequence-to-sequence (Seq2Seq) model. The main initial task for this model was that of Neural Machine Translation (NMT). The Transformer model is able to learn long-range dependencies in the data, which is useful for tasks such as NMT. The Transformer model is made up of an encoder and a decoder.

The encoder takes a sequence of inputs and produces a sequence of embeddings for the input (an autoencoder). The decoder takes the sequence of outputs from the encoder and produces a sequence of outputs autoregressively.

The encoder and decoder are made up of a number of identical layers. Each layer is made up of multiple attention heads, a feed forward neural network, and a residual connection as well as various batch normalisation steps. The multi-head attention layer is used to calculate the attention between the inputs and outputs. The feed forward neural network is used to transform the inputs and outputs. The residual connection is used to allow the model to learn the identity function as well as to control gradient flow through backpropagation. Figure 2.10 shows the architecture of the transformer model.

On the left is the encoder, which is made up of a stack of N identical layers. The multi-head attention layer is made up of a number of attention heads. Each attention head is made up of a query, key and value matrix. These three matrices have learned weights and are dotted with the input embeddings to generate the three vectors for each input. This generates fixed length vectors for each element in a sequence.

Scaled Dot-Product Attention

The scaled dot-product attention is used to calculate the attention between the inputs and outputs. The single-headed attention is calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.79)$$

where Q is the query matrix, K is the key matrix, V is the value matrix and d_k is the dimensionality of the key matrix. The multi-headed attention is calculated as:

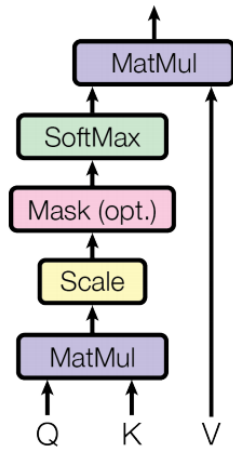
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)W^O \quad (2.80)$$

where h is the number of attention heads, W^O is the output weight matrix and Concat is the concatenation of the attention heads.

It is important to note that Q , K and V are matrices of the current input sequence which are generated by weight matrices W_Q , W_K and W_V which are learned during training.

Because a single attention head heavily compresses the sequence information by scoring based on the dot product of queries and keys, the feature extraction performed is limited. Multi-head attention is used to allow the model to learn different types of attention. The multi-head attention is able to learn different types of attention by using different weight matrices for each head. This allows the model to learn different types of attention such as: local attention, global attention and further attention to specific elements in the sequence. The operations for single and multi-head attention are shown in figure 2.11. Since each attention head has independent weights, multiple attentions can be computed in parallel.

Scaled Dot-Product Attention



Multi-Head Attention

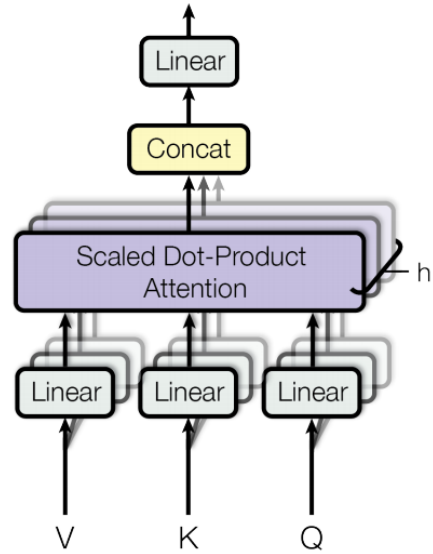


Figure 2.11: Scaled Dot-Product Attention and Multi-Head Attention.⁵

The normalisation scaling of the attention mechanism is based on the dimensionality of the key matrix, d_k . This is because the dot product of the query and key matrices is scaled by $\frac{1}{\sqrt{d_k}}$. This scaling is used to prevent the dot product from growing too large. The scaling is also used to prevent the softmax function from saturating. The softmax function is used to normalise

the attention scores into probabilities.

The computational operations scaling of attention is of the order $\mathcal{O}(n^2 \cdot d)$, where n is the length of the sequence and d is the model depth, with the scaling of activation operations being $\mathcal{O}(n^2 + n \cdot d)$. This is because the attention scores are calculated for each element in the sequence. This is computationally expensive and can be a bottleneck for the sequence length that can be given to the model. Whilst this is an issue, in NLP it is not high priority as sentences are rarely long enough for the n^2 dependency to become problematic. When applied to chemical data however, this issue is much more prominent.

One of the main benefits of multi-head attention is the ability for coreference resolution. This is the ability to resolve the same entity in a sequence. This is useful for tasks such as NMT where the same word can be translated into different words in different languages. This is also useful for tasks such as chemical property prediction where the same atom can have different properties in different molecules. It is useful to think about attention in the following way: from the perspective of each token in a sequence, the relevance/importance of every other token in that sequence to itself is computed and the highest scoring pairs output the highest attention score. In this context, a token is a vector representation (embedding) of either words, subwords, or characters as defined in the vocabulary.

Decoder/Masked Multi-Head Attention

Whilst the encoder performs scaled dot-product self attention where every token in the sequence is considered, the decoder must mask the yet unproduced tokens as it autoregressively generates them. This is done by setting Q , K and V to $-\infty$ such that they are 0 post softmax. The decoder then keeps going until a stop token is produced. The decoder is also able to use the encoder outputs to calculate the attention. This is done by setting Q to the decoder outputs and K and V to the encoder outputs. This is done to allow the decoder to learn the relationship between the encoder outputs and the decoder outputs. This is useful for tasks such as NMT where the decoder needs to learn the relationship between the input and output sequences.

Sinusoidal Positional Encoding

Positional encodings are added to the input embeddings to allow the model to learn the position of the tokens in the sequence. This is useful for tasks such as NMT where the order of the words in the sequence is important. The positional encodings are added to the input embeddings as follows:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}}) \quad (2.81)$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}}) \quad (2.82)$$

where pos is the position of the token in the sequence, i is the dimension of the embedding and d_{model} is the dimensionality of the model.

2.3 Chemical Feature Representation

For any computational task within chemistry, one must determine the optimal computational representation of the chemical system of interest. In the case of molecular dynamics, the representation is the atomic coordinates of the system. In the case of quantum chemistry, the representation is the molecular orbitals. In the case of machine learning, the representation is the features of the system. In this section, we will discuss the representations of chemical systems in the context of machine learning. We will first discuss classical molecular feature extraction methods such as molecular descriptors, and then discuss the representation of whole chemical systems in the context of neural networks.

2.3.1 Molecular Fingerprints and Descriptors

Molecular fingerprinting is one of the most common 2D feature extraction methods.^{60,72,110} These methods use predefined algorithms to extract a set of features from a molecule. The most common fingerprinting methods are based on the Morgan algorithm⁷² and the Daylight fingerprint.¹¹¹ The Morgan fingerprint is a circular based rotationally and translationally invariant fingerprinting algorithm.⁶⁰ The Daylight fingerprint is a path-based fingerprint, where each atom is represented by a node, and each bond is represented by an edge connecting the two atoms. These algorithms reduce the dimensionality of a 2D molecular structure to a 1D bit vector for use in computational tasks. Whilst Molecular fingerprints are themselves molecular descriptors, one of the main features of a fingerprint specifically is that the representation must be unique for each molecular structure. That is, the representation must be translationally, rotationally and permutationally invariant. This allows a fingerprint to be used to compare the similarity of two molecules through metrics such as the Tanimoto coefficient.¹¹² The Tanimoto coefficient is defined as the ratio of the number of bits that are set in both fingerprints to the number of bits that are set in either fingerprint. Tanimoto similarity is defined as:

$$T = \frac{A \cap B}{A \cup B} \quad (2.83)$$

This metric is used extensively in virtual screening as a way to search the functional chemical space within a large molecular library.¹¹³ Historically, topological fingerprinting was designed for this function, but more recent topological fingerprints now allow more structure-activity related searches.

Morgan and Extended Connectivity Fingerprints

Initially, the Morgan algorithm was a proposed solution to the molecular isomorphism problem (being able to identify the same molecule with different atom indexing). The algorithm was proposed by Morgan in 1965, and was later implemented by Daylight.⁷² The algorithm is based

on the concept of a circular fingerprint, where circular fragments are enumerated radially across the molecule. The Morgan algorithm acts as a base for a class of circular fingerprints known as the Extended Connectivity Fingerprints (ECFPs) or the extended-connectivity count vector (ECFC) - which usually are applications of the algorithms with varying parameters such as radius.⁶⁰ The Morgan algorithm is as follows:

1. For each atom in the molecule, generate a list of atoms that are within a certain radius of the atom (neighbouring atoms).
2. Assign each atom an invariant identifier. This invariant identifier is usually constructed from the following:
 - Number of neighbouring non-hydrogen atoms
 - Number of bonds to neighbouring non-hydrogen atoms
 - Atomic number
 - Atomic mass
 - Number of hydrogens
 - Is the atom in a ring?

These properties form the Daylight atomic invariants rule (DAIR).¹¹¹ The invariants are usually hashed (can be a simple sum) to a unique integer. This integer is the invariant identifier.

3. For each invariant, the invariants of the neighbouring atoms as well as the bond order to the neighbours is considered.
4. Update the current invariant to the hash of the neighbouring atom invariants and bond orders.
5. Repeat steps 3-4 until the number of unique invariants is unchanged (i.e. the invariants are stable due to expanding over the whole molecule).
6. The final fingerprint is a bit vector that is a hash of the invariants at each iteration.

It is useful to know that this algorithm may also be used as a method for canonical representation of molecules. One simple addition which can be made for SMILES canonicalisation for example, is to simply add the following two steps to the algorithm once the invariants are stable to determine the order of the atoms in the string:

- Assign priorities according to invariants.
- Disambiguate ties by atom type and bond order.

Whilst the invariants are by design constructed to be unique, ECFPs/Morgan FPs are not unique. Due to bit-collisions of the hash functions used, there is a many-to-one mapping between the invariants and the bit vector. This means that there are theoretically multiple molecules that will have the same fingerprint. This is not a problem for most applications, because the bit vector fingerprint is large (usually 2048 bits +) making them sparse such that the probability of a collision is very low. However, for applications where the probability of a collision is high, such as in polymers where the molecule is constructed from identical repeating monomer units, a different fingerprinting method must be used as a dimer would look identical to a trimer for example.

2.3.2 String Representations

SMILES

One of the most abundant representations of molecules in both public and private databases is the string representation SMILES (Simplified Molecular Input Line Entry System).⁶⁵ SMILES is a string representation of a molecule that is constructed from a depth-first search of the molecule. The algorithm is shown in figure 1.5.

The SMILES representation is linear and is constructed from a set of characters that represent the atoms, bonds and rings in the molecule. Whilst there are many extensions to SMILES (such as the Extended SMILES (ESMILES) and the Isomeric SMILES (ISMILES)), as well as

the SMILES arbitrary target specification (SMARTS).¹¹¹ The basic SMILES representation is as follows:

- Atoms are represented by their atomic symbol (e.g. C, H, O, N, etc.)
- Hydrogens are not explicitly represented, but are implied by the number of bonds to the atom. (Hydrogens with specified stereochemistry can be explicitly represented by the symbol H)
- Adjacent atoms in a SMILES string are connected by an implicit (single) bond. (e.g. CC is ethane)
- Explicit bonds are represented by a bond symbol (e.g. =, -, #, etc.) between the two atoms. (e.g. C=C is ethene)
- Branches are represented by parentheses. (e.g. CC(C)C(=O)O is isobutyric acid)
- Cyclic molecules are represented by a number (e.g. C1CC1 is cyclopropane)
- Stereochemistry is represented by the @ symbol (e.g. N[C@@H](C)C(=O)O is L-alanine)
- Isomeric information is represented by the \ or / symbols (e.g. F\C=C\F is trans-difluoroethene)
- Square brackets can be used to represent a specific atom/functional group (e.g. [CH3] is methyl)

It is important to note that whilst there have been efforts to try and standardise canonical SMILES, there is no standardised canonical SMILES representation. This is because there are many different ways to represent the same molecule, and there is no way to determine which is the correct representation. This is because the SMILES string is constructed from a depth-first search of the molecule, and the order of the atoms in the string is not consistent.⁶⁶ As such, whilst SMILES are commonly used to search and compare molecular databases, there is a trend to combine with more unique representations such as molecular identifiers or otherwise for database search.

In ML, this redundancy may be exploited for data augmentation, as we permute the SMILES string to generate new molecules. This is discussed in more detail later in this thesis.

2.3.3 InChI

The International Chemical Identifier (InChI)¹¹⁴ is a string representation of a molecule that is constructed from a breadth-first search of the molecule.

It describes molecules in layers, with each layer describing a different aspect of the molecule. The layers and their prefixes are as follows:

1. The chemical formula of the molecule. `no prefix`
2. The non-hydrogen atom connectivity of the molecule (topology). `/c`
3. Hydrogen connectivity of the molecule (bond orders). `/h`
4. The stereochemistry of the molecule. `/b /t /m /s`
5. The isotopic composition of the molecule. `/i /h /b /t /m /s`
6. The charge of the molecule. `/q /p`

The InChI string is constructed from the layers in the order above, with each layer separated by a forward slash (/) and the respective prefix letter. The chemical formula is constructed from the atomic symbols of the atoms in the molecule, with the number of atoms of each type in the molecule. The atom connectivity is constructed from the atomic symbols of the atoms in the molecule, with each atom numbered in sequence of bonding (excluding hydrogen). The hydrogen connectivity is determined by counting the number of hydrogens attached to each counted atom, bond order is therefore also defined in this sublayer. The stereochemistry is constructed by considering double bonds and cumulenes `/c`, tetrahedrals and allenes `/t /m` and other types `/s`. The charge is constructed from the atomic symbols of the atoms in the molecule, with the charge of each atom in the molecule `/p` is for protons.

Atoms are numbered according to molecular formula and connectivity is similar to SMILES in which each atom is defined by its number. There are no “ring closure” numbers for cyclic molecules, instead, numbers can be used.

It is important to note that unlike SMILES which have redundant representations, InChI strings are unique and can be used to uniquely identify a molecule.

2.3.4 Protein Structure Representations

Whilst proteins are simply polymers of amino acids, the structure of the protein is much more complex than the structure of a molecule. The primary structure of a protein is defined by the sequence of amino acids in the protein. The secondary, tertiary and quaternary structures of a protein are defined by the interactions between the amino acids in the protein. The secondary structure of a protein is determined by the hydrogen bonding between the amino and carboxyl groups of the amino acids in the protein. The tertiary structure of a protein depends on the secondary structure as well as the interactions between the side chains of the amino acids in the protein. The quaternary structure of a protein is defined by the interactions within multiple protein subunits.

The 3D structure of a protein is thus important to its function and it is therefore more common to use the 3D structure of proteins for computational tasks, rather than a 2D molecular graph, or a linear representation.

One of the most common 3D representations of proteins is the Protein Data Bank (PDB) format.⁵⁷

PDB

The PDB is a database of 3D structures of proteins, nucleic acids and small molecules.⁵⁷ A PDB file is a text file that contains the 3D coordinates of the atoms in the protein, as well as more recently, the connectivity of the atoms in the protein or protein-ligand complex.

Each PDB is split into sections or records. These include:

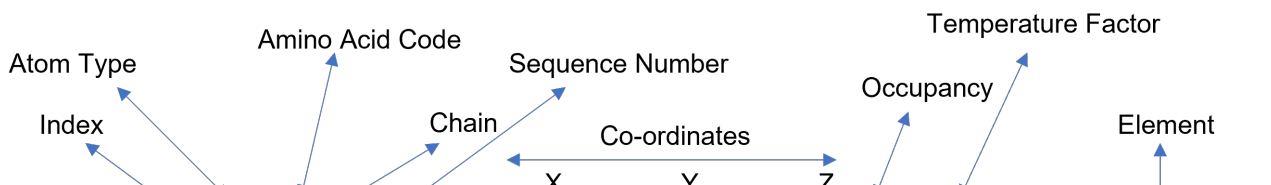
- The HEADER record, which contains information about the protein, such as the name of the protein, the date of deposition, the resolution of the structure, the experimental method used to determine the structure, etc.
- The TITLE record, which contains the title of the protein.
- The COMPND record, which contains the compound name of the protein.
- The SOURCE record, which contains information about the source of the protein, such as the organism from which the protein was isolated.
- The KEYWDS record, which contains key words that describe the protein.
- The EXP record, which contains information about the experimental method used to determine the structure of the protein.
- The AUTHOR record, which contains the name of the author of the protein.
- The JRNL record, which contains the journal reference for the protein.
- The REMARK record, which contains additional information about the protein.
- The SEQRES record, which contains the primary sequence of the protein.
- The HELIX record, which contains information about the alpha helices in the protein.
- The SHEET record, which contains information about the beta sheets in the protein.
- The TURN record, which contains information about the turns in the protein.
- The SITE record, which contains information about the sites in the protein.
- The CRYST1 record, which contains information about the crystallographic symmetry of the protein.
- The CONNECT record, which contains information about the connectivity of the atoms in the protein.

- The MASTER record, which contains information about the number of records in the PDB file.
- The ATOM record, which contains the 3D coordinates of the atoms in the protein.
- The HETATM record, which contains the 3D coordinates of the atoms in the protein-ligand complex.
- The END record, which contains the end of the PDB file.
- The MODEL record, which contains the start of a new model in the PDB file.
- The TER record, which contains the end of a model in the PDB file.

The ATOM and HETATM records are the most important records in the PDB file, as they contain the 3D coordinates of the atoms in the protein. The ATOM record contains the 3D coordinates of the atoms in the protein, whilst the HETATM record contains the 3D coordinates of the other atoms in the protein-ligand complex (ligand, metals etc.). The ATOM record is split into the following fields:

- The record name, which is always ATOM.
- The atom index, which is a unique integer that identifies the atom.
- The atom name, which is a string that identifies the atom.
- The alternate location indicator, which is a string that identifies the alternate location of the atom.
- The residue name, which is a string that identifies the residue that the atom is in.
- The chain identifier, which is a string that identifies the chain that the atom is in.
- The residue sequence number, which is an integer that identifies the residue that the atom is in.
- The code for insertion of residues, which is a string that identifies the insertion of residues.

- The x coordinate, which is a float (a non-integer number) that identifies the x coordinate of the atom.
- The y coordinate, which is a float that identifies the y coordinate of the atom.
- The z coordinate, which is a float that identifies the z coordinate of the atom.
- The occupancy, which is a float that identifies the occupancy of the atom.
- The temperature factor, which is a float that identifies the temperature factor of the atom.
- The segment identifier, which is a string that identifies the segment that the atom is in.
- The element symbol, which is a string that identifies the element of the atom.
- The charge on the atom, which is an integer that identifies the charge on the atom.



Atom Type	Index		Amino Acid Code	Chain	Sequence Number	Co-ordinates			Occupancy	Temperature Factor	Element
						X	Y	Z			
ATOM	9	N	TYR	A	2	4.475	21.533	64.814	1.00	16.08	N
ATOM	10	CA	TYR	A	2	3.469	22.329	64.124	1.00	16.57	C
ATOM	11	C	TYR	A	2	3.789	23.799	64.273	1.00	17.49	C
ATOM	12	O	TYR	A	2	3.330	24.625	63.481	1.00	17.72	O
ATOM	13	CB	TYR	A	2	2.079	22.044	64.692	1.00	16.58	C
ATOM	14	CG	TYR	A	2	1.706	20.594	64.649	1.00	16.16	C
ATOM	15	CD1	TYR	A	2	1.586	19.922	63.424	1.00	15.60	C
ATOM	16	CD2	TYR	A	2	1.478	19.868	65.837	1.00	16.06	C
ATOM	17	CE1	TYR	A	2	1.245	18.538	63.382	1.00	14.41	C

Figure 2.12: An example of PDB ATOM records.

It is important to note that the PDB format bonding information is based on residue-level resolution. Each standard residue has its own 3 letter code. With protein-ligand complexes, the ligand is also given a 3 letter code. Given that the PDB format is based upon residue level resolution, it is not possible to identify the bonding of the atoms of non-standard amino acids or ligands in a protein-ligand complex without further definitions within the PDB. This is a problem for computational tasks such as docking, where these atoms and their interactions in the protein-ligand complex are important. Figure 2.12 shows an example of how atoms are defined based on their type and the residue that they belong to in the PDB format. The example shown is of a tyrosine residue.

With the increase in crystallographic data output and protein-ligand complexes uploaded on the PDB, the PDB format will be retiring from 2024 out of necessity due to the limitations of the 3 letter codes which have nearly all been used up. The new format will be called the PDBx/mmCIF format, which is a more flexible format that will allow for more information to be stored in the file.¹¹⁵

2.4 Molecular Dynamics

The 3D structure of a protein is important to its function to varying degrees. However a protein (as with all molecules) is dynamic and fluctuates over time. One method of understanding the relationship between structure and function are quantitative structure-activity relationship

(QSAR) modelling techniques. These techniques are however, based upon the assumption that the structure of a protein is static.

Molecular Dynamics (MD) allows more in-depth understanding of the function of a protein through its atomic-level interactions over time. Whilst MD itself is a deterministic simulation, the constraints that keep a system from exploding (temperature and pressure regulators) are (usually) stochastic, meaning that the results of the simulation will vary with each simulation and thus repeat simulations and data analysis must be used to conclude simulation results.

Since MD simulates protein dynamics, given the right conditions, a primary sequence of amino acids will fold into its tertiary structure in an MD simulation. Theoretically, the energy of the system will be minimised through time with MD as it reaches the more stable, folded conformation.^{116,117}

2.4.1 Molecular Mechanics

In reality, the atomic interactions with a system are quantum. However, MD simulations are performed using classical mechanics. This is because the quantum effects are (usually) too small to be considered in the simulation, given the computational cost required. Hybrid Quantum Mechanics/Molecular Mechanics MD (QM/MM/MD) simulations are possible. They are computationally expensive and are not usually used outside of specific situations.^{116,117}

In MM/MD, the interactions between atoms in a system are modelled using a force field. A force field is a mathematical model that describes the interactions between atoms within a system. The force field is usually based upon the Lennard-Jones potential, which is a function of the distance between two atoms. However, many variants of force fields exist that allow for better alignment with experimental data depending on the system being modelled.

Force Fields

In this thesis, the AMBER forcefield was used, specifically ff19SB.¹¹⁸ The AMBER forcefield is a general purpose forcefield that is used for a wide range of systems. The ff19SB variant of the AMBER forcefield is a variant of the ff19 variant of the AMBER forcefield that is specifically designed for proteins.^{118–123} AMBER force fields have the general form:

$$E_{total} = \sum_{bonds} E_{bond} + \sum_{angles} E_{angle} + \sum_{dihedrals} E_{dihedral} + \sum_{impropers} E_{improper} + \sum_{non-bonded} E_{non-bonded} \quad (2.84)$$

Where E_{bond} is the bond energy, E_{angle} is the angle energy, $E_{dihedral}$ is the dihedral energy, $E_{improper}$ is the improper energy, and $E_{non-bonded}$ is the non-bonded energy. The bond energy is given by:

$$E_{bond} = \frac{1}{2}k_b(r - r_0)^2 \quad (2.85)$$

Where k_b is the bond force constant, r is the distance between the atoms, and r_0 is the equilibrium bond length. The angle energy is given by:

$$E_{angle} = \frac{1}{2}k_a(\theta - \theta_0)^2 \quad (2.86)$$

Where k_a is the angle force constant, θ is the angle between the atoms, and θ_0 is the equilibrium angle. The dihedral energy is given by:

$$E_{dihedral} = \sum_{i=1}^n \frac{V_i}{2} (1 + \cos(n_i\phi - \phi_i)) \quad (2.87)$$

Where V_i is the dihedral force constant, n_i is the dihedral multiplicity, ϕ is the dihedral angle, and ϕ_i is the dihedral phase. The improper energy is given by:

$$E_{improper} = \frac{1}{2}k_i(\phi - \phi_0)^2 \quad (2.88)$$

Where k_i is the improper force constant, ϕ is the improper angle, and ϕ_0 is the equilibrium improper angle. The non-bonded energy is given by:

$$E_{non-bonded} = \sum_{i=1}^N \sum_{j=i+1}^N \left(4\pi\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^6 \right] + \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} \right) \quad (2.89)$$

Where the non-bonded term is the sum of the 6-12 Lennard-Jones potential and the Coulomb potential. ϵ_{ij} is the Lennard-Jones well depth, σ_{ij} is the Lennard-Jones radius, q_i is the charge of atom i , q_j is the charge of atom j , r_{ij} is the distance between atoms i and j and ϵ_0 is the permittivity of free space.

It is important to note some of the limitations given by this representation of a biological system.

1. Given that the bond energy is simulated using Hooke's law, the bond energy is only valid for linear bonds. This means that bonds can not be broken or formed in the simulation.
2. Each atom is considered as a point charge, this introduces limitations in systems where there are anisotropic charge distributions as well as systems where electronic polarizations affect the environment.

It is these parameters that are optimised in the force field fitting/creation process. This general form with rudimentary parameterisation is known as the Generalised Amber Force Field (GAFF), this version of the force-field has distinct atom types and parameters and every AMBER force field is a variant and/or subset of GAFF.

2.4.2 Energy Landscapes

One way to view MD Simulations is through the lens of energy sampling as the conformational space is searched.

The energy landscape is a graph of the energy of a system as a function of the conformation of the system. It is a useful tool for understanding the behaviour of a system and can be visualised as a topographical map, where the energy is the height of the map and the conformation is the position on the map.

This view intuitively gives rise to the ergodic hypothesis which states that the time spent in a region of the energy landscape (conformational space) is proportional to the volume of the region. This means that the time spent in a region of the energy landscape is proportional to the number of times the system will visit that region (i.e. all accessible microstates are equiprobable). This is a useful tool for understanding the behaviour of a system as it allows MD simulations to explain experimental results given the simulation is long enough.^{116,117}

Energy Minimisation

Once the system's parameters are assigned with the forcefield, energy minimisation of the system is performed as a pre-processing step. This is done to ensure that the system is in a stable state before the simulation is started (i.e. there are no contacts and the proper molecular arrangement is defined in 3D space). This is done by iteratively applying forces to the system to minimise the energy of the system. The process is stopped when the energy of the system is below a certain threshold.

During energy minimisation iterations, the gradient descent algorithm is used. This process is analogous to error minimisation when learning neural network weights. Once the energy fluctuations are narrowed down to a minima, conjugate gradient descent is used to further fine-tune the energy minimisation. The conjugate gradient descent algorithm takes into account the previous gradient to determine the direction of the next gradient. This is done to ensure

that the energy minimisation converges very close to a local minima. The algorithm is defined as:

$$\mathbf{g} = \nabla E(\mathbf{x}) \quad (2.90)$$

$$\mathbf{d} = -\mathbf{g} + \beta \mathbf{d} \quad (2.91)$$

$$\beta = \frac{\mathbf{g}^T \nabla^2 E(\mathbf{x}) \mathbf{d}}{\mathbf{d}^T \nabla^2 E(\mathbf{x}) \mathbf{d}} \quad (2.92)$$

$$\mathbf{x} = \mathbf{x} + \alpha \mathbf{d} \quad (2.93)$$

$$\alpha = \frac{\mathbf{g}^T \mathbf{g}}{\mathbf{d}^T \nabla^2 E(\mathbf{x}) \mathbf{d}} \quad (2.94)$$

Where $\mathbf{g}$ is the gradient, $\mathbf{d}$ is the direction, α is the step size, β is the conjugate gradient factor, $\mathbf{x}$ is the position, and $\nabla^2 E(\mathbf{x})$ is the Hessian matrix.

2.4.3 Molecular Dynamics Simulations

The molecular dynamics simulation is the process of simulating the motion of the atoms in the system. This is done by numerically solving Newton's equations of motion. The equations of motion are given by:

$$\mathbf{F} = m \frac{d^2 \mathbf{r}}{dt^2} \quad (2.95)$$

Where $\mathbf{F}$ is the force, $\mathbf{r}$ is the position, and t is the time. The force is given by:

$$\mathbf{F} = -\nabla E(\mathbf{r}) \quad (2.96)$$

Where $\mathbf{r}$ is the position, $E(\mathbf{r})$ is the energy, and ∇ is the gradient operator. The simplest algorithm used to solve the equations of motion is the Verlet.¹²⁴ This algorithm is a numerical integration method and is defined as:

$$\mathbf{r}(t + \Delta t) = 2\mathbf{r}(t) - \mathbf{r}(t - \Delta t) + \left(\frac{\Delta t^2}{m}\right) \mathbf{F}(t) \quad (2.97)$$

Where $\mathbf{r}$ is the position, t is the time, Δt is the time step, m is the mass, and $\mathbf{F}$ is the force.

AMBER uses a variant of the Verlet algorithm called the leapfrog algorithm. This algorithm is defined as:

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \mathbf{v}\left(t + \frac{\Delta t}{2}\right) \Delta t \quad (2.98)$$

$$\mathbf{v}\left(t + \frac{\Delta t}{2}\right) = \mathbf{v}\left(t - \frac{\Delta t}{2}\right) + \mathbf{a}(t) \Delta t \quad (2.99)$$

Where $\mathbf{r}$ is the position, $\mathbf{v}$ is the velocity, t is the time, Δt is the time step and $\mathbf{a}$ is the acceleration.

2.4.4 Generalised Born Surface Area

The Generalised Born (GB) method is used to calculate the potentials along the surface area of a molecule, such that solvation energy could be approximated.¹²⁵ The GB method decomposes the free energy of solvation, ΔG_{solv} , into an electrostatic and a non-electrostatic components:

$$\Delta G_{solv} = \Delta G_{elec} + \Delta G_{non-elec} \quad (2.100)$$

The non-elec term is taken to be proportional to the solvent accessible surface area (SASA) of the molecule as it is a combination of favourable Van der Waals and unfavourable disruption of the current water structure. The GB model approximates ΔG_{elec} as:

$$\Delta G_{elec} = -\frac{1}{2} \sum_{ij} \frac{q_i q_j}{f_{GB}(r_{ij}, R_i, R_j)} \left(1 - \frac{\exp[-\kappa f_{GB}]}{\epsilon} \right) \quad (2.101)$$

where:

$$f_{GB} = \sqrt{\left[r_{ij}^2 + R_i R_j \exp\left(\frac{-r_{ij}^2}{4R_i R_j}\right) \right]} \quad (2.102)$$

r_{ij} is the distance between atoms i and j , R_i and R_j are the Born radii of atoms i and j , κ is the dielectric constant of the solvent, ϵ is the dielectric constant of the solute, and q_i and q_j are the charges of atoms i and j . Depending on the GB model variant used (the igb parameter in AMBER), the Born radii are calculated differently.^{125–133}

Chapter 3

Predicting Lipid Phase Behaviour from Small Angle X-ray Scattering (SAXS)

Lipid membranes are a ubiquitous component of biological systems. They are the primary component of cell membranes and are found in the membranes of eukaryotic organelles such as mitochondria and chloroplasts.¹³⁴

Like all classes of biomolecules, lipids are subject to a variety of forces that determine their structure and function.¹³⁵ These forces include hydrophobic interactions, electrostatic interactions, and van der Waals interactions.¹³⁶ The relative importance of these interactions varies depending on the lipid and the environment in which it is found. In general, lipids (specifically phospholipids) tend to aggregate in aqueous environments, forming micelles or bilayers. The formation of these structures is driven by the hydrophobic effect, which is the tendency of non-polar molecules to aggregate in aqueous environments to avoid water-non-polar contact.^{136,137}

The hydrophobic effect is a consequence of the fact that the non-polar tails of lipids are not soluble in water. This results in the tails of the lipids being excluded from the aqueous environment, and the hydrophobic tails of the lipids being forced to interact with each other. This interaction is mediated by van der Waals forces, which are the weakest of the three interactions mentioned above. The hydrophobic effect is also responsible for the formation of micelles and

bilayers, which are the two most common structures of lipids in aqueous environments.^{136,138}

In this chapter, an ML pipeline and method are developed to allow rapid analysis of experimental lipid structure data. The method is based on the use of a convolutional neural network (CNN) with a new computational representation to predict the phase of a lipid phase from its small angle X-ray scattering (SAXS) pattern. The model is trained on a large dataset of synthetic SAXS patterns generated for a variety of phases and is then tested and benchmarked on a dataset of experimentally acquired SAXS patterns.

Phospholipids

Phospholipids are a class of lipids that are composed of fatty acid chains, and a phosphate group. The phosphate group is negatively charged, and the fatty acid chains are non-polar. The phosphate group is responsible for the negative charge of the phospholipid, and the fatty acid chains are responsible for the hydrophobicity of the phospholipid. The combination of the phosphate group and the lipid chain is responsible for the amphipathic nature of the phospholipid as the phospholipid has both hydrophobic and hydrophilic regions.¹³⁹

Single phospholipids can be classified based on the proportions of their respective regions. Depending on the sizes of each region within the phospholipid, the curvature of the aggregated phospholipids is affected in order to minimise the structure's surface area and energy. This is known as the *curvature effect*.¹⁴⁰ The three types of phospholipids are shown in Figure 3.1.

1. Type 0 phospholipids have a hydrophilic region that is proportionally similar to the hydrophobic fatty acid chain cross sectional area. The molecular packing is cylindrical and the curvature is zero.
2. Type I phospholipids have a hydrophilic region that is proportionally larger than the hydrocarbon chain. The molecular packing is conical and the curvature is positive.¹³⁷
3. Type II phospholipids have a hydrophilic region that is proportionally smaller than the hydrophobic fatty acid chain. The packing is inverted conical with negative curvature.¹⁴¹

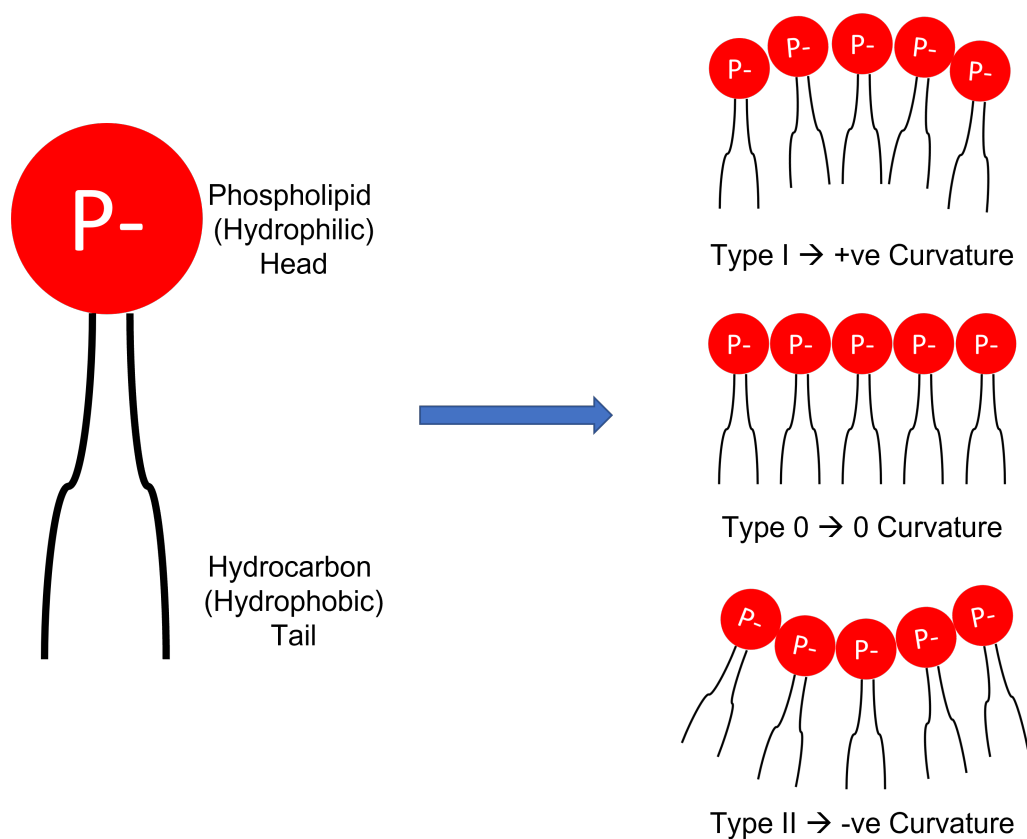


Figure 3.1: The three types of phospholipids and their curvature effects.

Whilst lipid structure is one of the main predictors of phase behaviour, environmental effects such as pH, temperature, pressure, concentration, etc. also have a huge impact on the phase behaviour of lipids.^{142–144}

3.1 Lipid Phase Behaviour

One of the most common aggregates of lipids within eukaryotes is the lipid bilayer aggregate. A lipid bilayer is a two-dimensional structure that is composed of two layers of phospholipids. The two layers of phospholipids are separated by a hydrophobic core, which is composed of the hydrophobic tails of the phospholipids. The hydrophobic tails of the phospholipids are not soluble in water, and therefore are excluded from the aqueous environment. The hydrophobic tails of the phospholipids are forced to interact with each other, and this interaction is mediated by van der Waals forces. Lipid bilayers are then stacked on top of each other, linearly to form the **lamellar phase**.¹⁴⁵

Where there is zero curvature, the bilayers are said to be **flat** (lamellar phase). Where the curvature is positive, the bilayers are said to be **curved**. Where the curvature is negative, the bilayers are said to be **inverted**.^{146,147} The lamellar phase is one of the most common phases of lipid bilayers in aqueous environments.¹⁴⁵

Curvature and Phase Behaviour

For a bilayer system, curvature is one of the most important factors that determines the phase behaviour of lipid bilayer as it determines the stacking.¹⁴⁷

One way to quantify thresholds between phases is to quantitatively assess a system's curvature. The mean curvature (H) is defined as:

$$H = \frac{1}{2}(c_1 + c_2) \quad (3.1)$$

where c_1 and c_2 are the principal curvatures of the bilayer. They are perpendicular to the bilayer normal radius.¹⁴⁸ The Gaussian curvature is defined as:

$$K = c_1 c_2 \quad (3.2)$$

In summary: $H > 0$ is a curved bilayer, $H < 0$ is an inverted bilayer.¹⁴⁹ $K > 0$ is an ellipsoidal surface, $K < 0$ is a hyperbolic surface and $K = 0$ is a parabolic surface.¹⁴⁸

Within a lamellar bilayer, lateral pressure caused by the hydrophobic tails of the phospholipids is balanced by the hydrophilic head groups of the phospholipids.¹⁵⁰ The pressure profile of a single monomer phospholipid is shown in figure 3.2.

The outward pressure observed in the head and the tail are due to sterics and electrostatic repulsions, with the inward interfacial tension at the bottom of the head acting as an inward pressure.¹⁴⁸ In a stable non-curved bilayer, the lateral pressure across the bilayer is symmetric. With an offset in the lateral pressure stimulating curvature within the membrane. The lateral

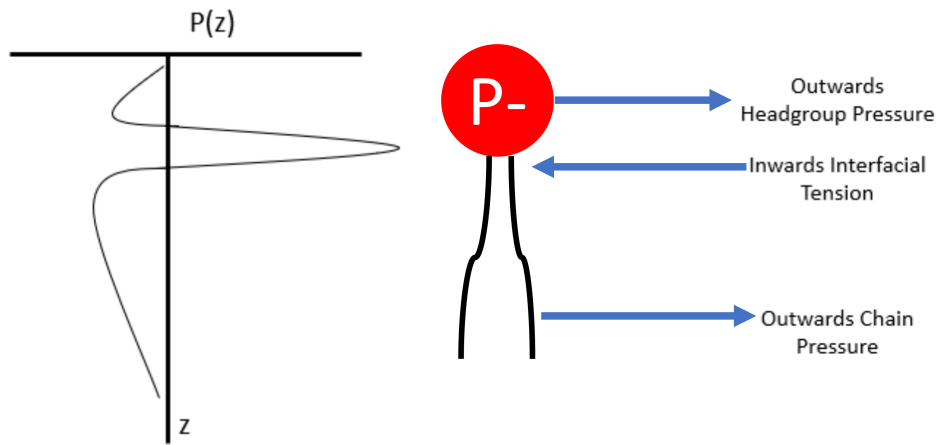


Figure 3.2: The lateral pressure profile of a single monomer phospholipid.

pressure profile can be manipulated by the environment (temperature, pH, etc.) and the lipid composition.

Helfrich showed that the energy of a lipid bilayer is a function of the curvature of the bilayer.¹⁴⁹ As such, the energy required to perturb the bilayer from its preferred state can be given by:

$$g_C = 2\kappa(H - H_0)^2 + \kappa_G K \quad (3.3)$$

where g_C is the curvature elastic energy per unit area, H_0 is the spontaneous curvature, κ is the bending rigidity and κ_G is the Gaussian curvature rigidity. The spontaneous curvature is the curvature of the bilayer at zero applied pressure. The bending rigidity is the energy required to bend the bilayer by one unit of curvature. The Gaussian curvature rigidity is the energy required to change the Gaussian curvature of the bilayer by one unit.

The spontaneous curvature of a bilayer is determined by its lateral pressure profile.

$$\kappa^b H_0 = \int p(z) dz \quad (3.4)$$

where κ^b is the bending rigidity of the bilayer, $p(z)$ is the lateral pressure profile¹⁴⁸ and z is the distance from the bilayer centre. For a symmetric bilayer $H = 0$ and $\kappa^b \neq 0$ since the integral must be 0.

Curvature frustration within the bilayer may occur when single monomer phospholipids are stimulated to change their curvature.¹⁴⁹ Initially, this change is resisted by the bilayer as the bilayer is in a state of equilibrium and so will expand to compensate. Once the expansion energy penalty becomes too large, the bilayer may invert with the lipids having a negative interfacial curvature.

3.1.1 Lamellar Phases

The lamellar phase is the one of the most common phases of lipid bilayers in aqueous environments. The lamellar phase is a two-dimensional structure that is composed of two layers of phospholipids. The two layers of phospholipids are separated by a hydrophobic core, which is composed of the hydrophobic tails of the phospholipids. The hydrophobic tails of the phospholipids are not soluble in water, and therefore are excluded from the aqueous environment. The hydrophobic tails of the phospholipids are forced to interact with each other, and this interaction is mediated by van der Waals forces.¹⁴⁵ Lipid bilayers are then stacked on top of each other, linearly to form the **lamellar phase**

Depending on the lipids and their environment (temperature, pressure, etc.) the lamellar phase can be further divided into subphases. Fluid Lamellar, L_α , Gel Phase, L_β , and Crystalline Lamellar, L_c . Figure 3.3 shows these subphases.

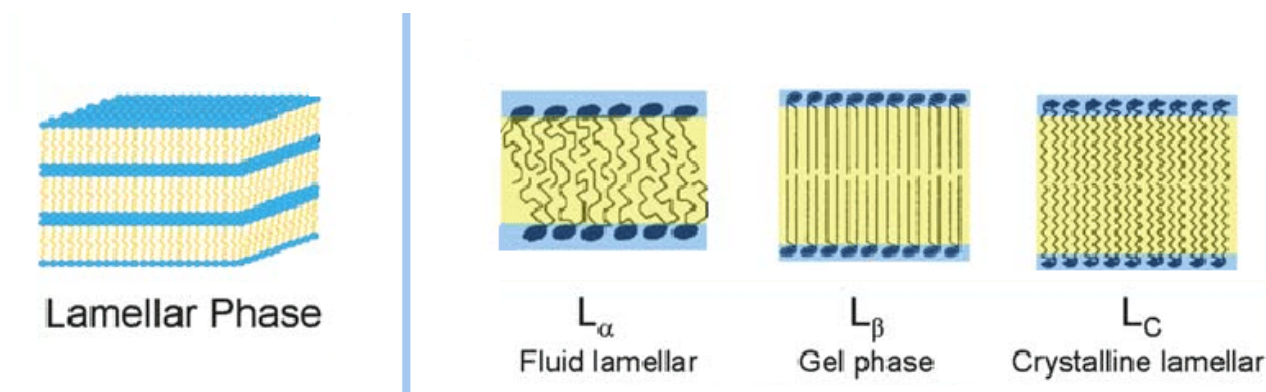


Figure 3.3: The lamellar phases of lipid bilayers, figure adapted from Seddon and Templer's work¹⁵¹

In the fluid lamellar, L_α , phase, the lipids diffuse freely within the bilayer structure and there is

hydrocarbon chain splay.¹⁵² In the gel L_β phase, the lipids are more ordered and the hydrocarbon chains are less freely able to move, however the heads can rotate about their axis.¹⁵³ In the crystalline lamellar, L_c phase, the lipids are ordered and the hydrocarbon chains are not able to move. The heads are also ordered and cannot rotate about their axis. The lipids are fixed in a lattice structure.¹⁵⁴ Temperature and pressure can be used to control the phase transition between the lamellar phases, from high to low temperatures (or the inverse for pressure). The lipid lamellar phases go from L_α to L_β to L_c respectively.¹⁵⁵ Figure 3.4 shows an example of a temperature-pressure phase diagram of a 1,2-Dioleoyl-sn-glycero-3-phosphocholine (DOPC) lipid bilayer.

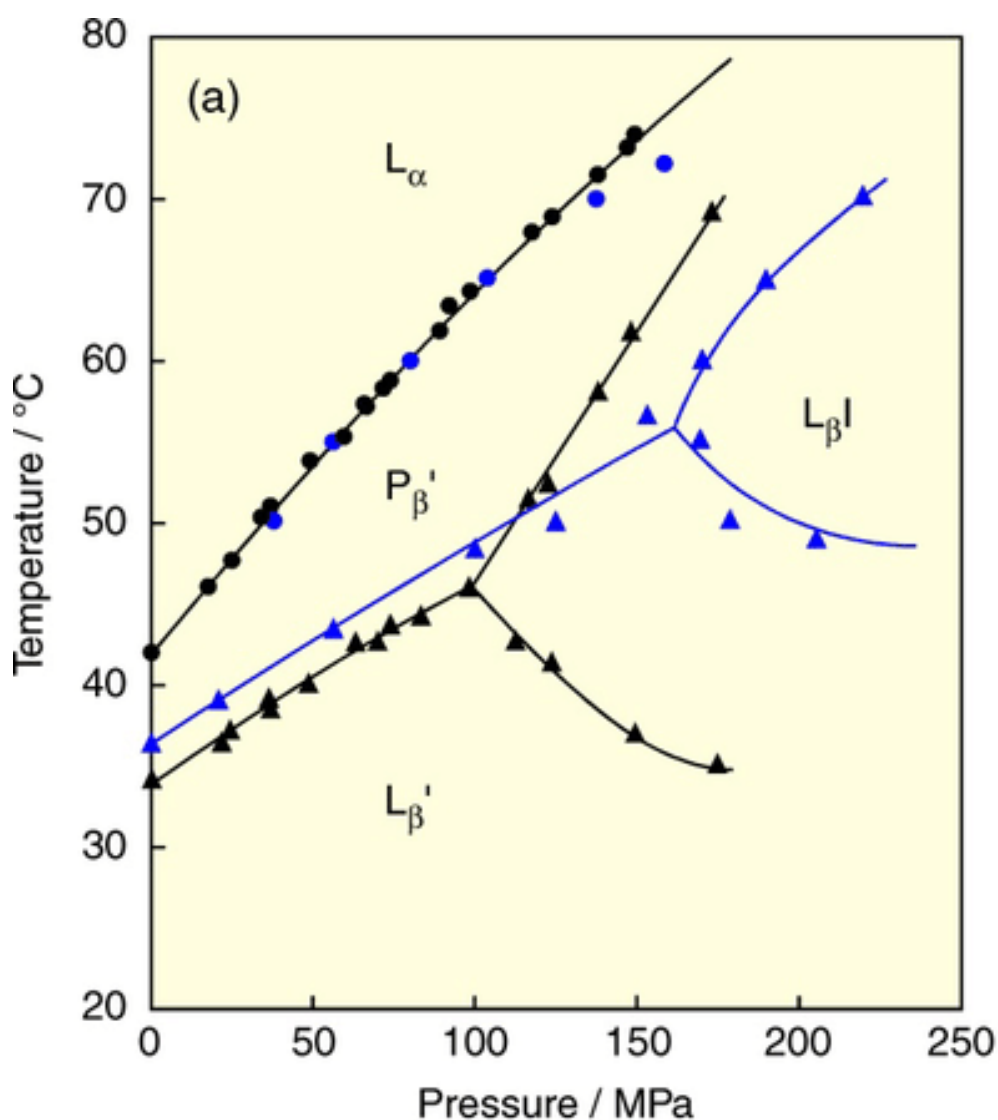


Figure 3.4: Temperature-pressure phase diagram of a DOPC lipid bilayer in D_2O (blue) and H_2O (black). L_β' , $L_\beta I$ and P_β' are the tilted, inter-digitated and rippled phases respectively - all subsets of the L_β gel phase. Figure adapted from Matsuki et al.¹⁵⁶

3.1.2 Hexagonal Phases

The Hexagonal phase consists of multiple cylindrical micelles of lipids which pack together to form a 2D hexagonal lattice.

There are two variants of the Hexagonal Phase (shown in figure 3.5), H_I (normal hexagonal) where the micelles are aligned such that the headgroups are on the outer surface and H_{II} (inverse hexagonal) where the micelles are aligned such that the tails are on the outer surface.¹⁵⁷

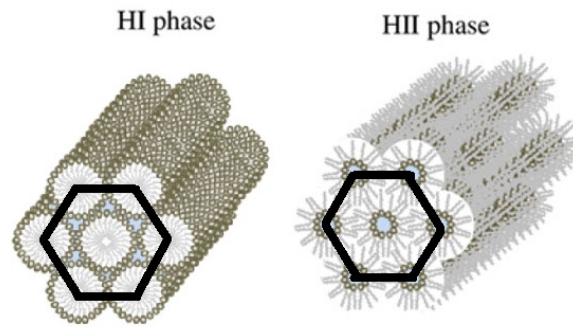


Figure 3.5: The normal and inverse hexagonal phase packing of cylindrical lipid micelles, figure adapted from Juliette Jouhet's work¹⁴¹

Inverse Hexagonal (H_{II}) is much more common than normal hexagonal (H_I) as it can be transitioned into from Lamellar phases in biological systems^{158,159} and plays important roles in membrane fusion and enzyme activity within eukaryotes.^{141,160}

3.1.3 Inverse Bicontinuous Cubic Phases

The final set of phases we will discuss are the much more rare cubic phases. These phases are geometrically 3D minimal surfaces which have a mean curvature of 0 and are therefore usually parameterised as $\mathbf{X} = (u, v, h(u, v))$ and satisfy Lagrange's equation:

$$(1 + h_v^2)h_{uu} - 2h_u h_v h_{uv} + (1 + h_u^2)h_{vv} = 0 \quad (3.5)$$

where h is the mean curvature, h_u and h_v are the partial derivatives of h with respect to u and v respectively.¹⁶¹

The three cubic phases we will discuss have space groups $Im\bar{3}m$ (Primitive, P), $Pn\bar{3}m$ (Diamond, D) and $Ia\bar{3}d$ (Gyroid, G).¹⁶² Figure 3.6 shows the three cubic phases.

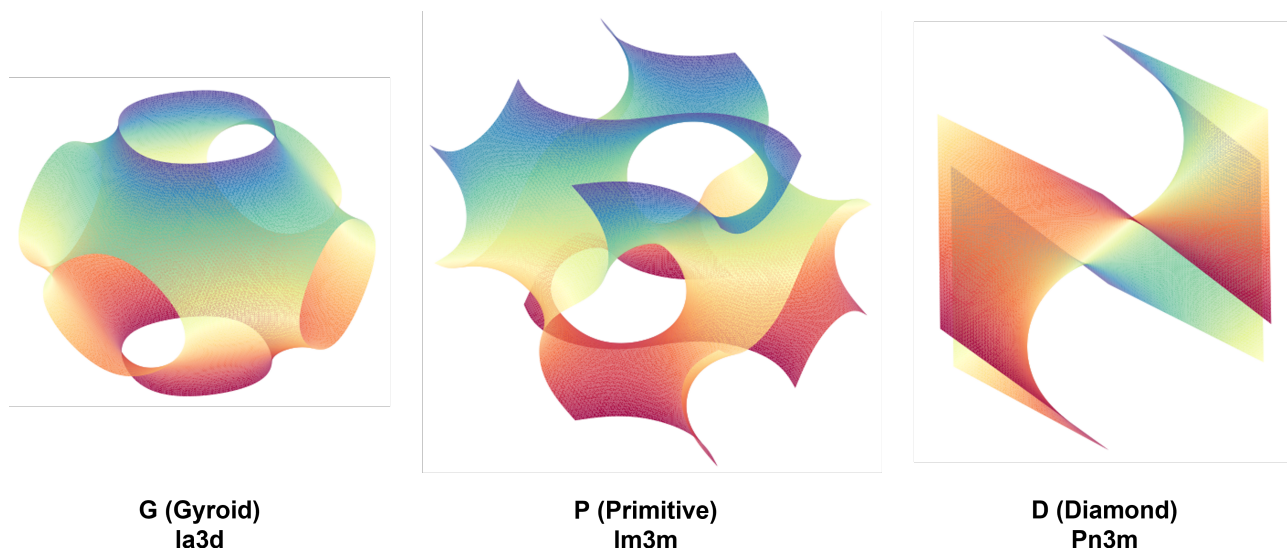


Figure 3.6: The Inverse Bicontinuous Cubic Phases of Lipid Bilayers

These structures are Triply Periodic Minimal Surfaces (TPMS) as the structure is repeated in 3D space.^{163,164} Their properties allow the lipids to aggregate in such a way to avoid curvature frustration as their curvature magnitude is greater than lamellar, but less than hexagonal. They are thought to be of importance due to their potential applications as nano-particle drug delivery systems.

3.2 Small Angle X-ray Scattering Patterns

Lipid macrostructures are often of the scale of Ångstrom to nanometers. This is within the range of the wavelength of X-rays (0.1-10 nm). As such, X-rays will scatter off the lipid bilayer and the resulting pattern can be used to determine the structure of the lipid system.^{73,165–167}

3.2.1 X-ray Scattering

Bragg scattering is the scattering of X-rays off a crystal lattice. The scattering is dependent on the wavelength of the X-rays and the lattice spacing of the crystal. The scattering is at a fixed angle, θ , and the intensity of the scattering is proportional to the square of the scattering vector, q , which is related to the wavelength of the X-rays and the angle of scattering, θ .^{168–170}

For an incident X-ray of wavelength λ the scattering angle is given by:

$$n\lambda = 2d_{hkl} \sin \theta_{hkl} \quad (3.6)$$

where n is an integer, d_{hkl} is the lattice spacing of the crystal and θ_{hkl} is the scattering angle. This is known as Bragg's Law.¹⁶⁹

In essence, for a pair of in-phase, parallel waves separated by a distance d_{hkl} , the waves will interfere constructively when the distance travelled by the waves is an integer multiple of the wavelength of the waves. This is known as Bragg's condition. The signal given on the detector is therefore a function of the wave phase and the angle of scattering.

An Ewald sphere is a sphere of radius q which is centred on the origin of the reciprocal space. Considering the basis vectors in Euclidean space, $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, the reciprocal basis vectors are given by:

$$\mathbf{a}^* = \frac{2\pi}{a} \mathbf{b} \times \mathbf{c} \quad (3.7)$$

$$\mathbf{b}^* = \frac{2\pi}{b} \mathbf{c} \times \mathbf{a} \quad (3.8)$$

$$\mathbf{c}^* = \frac{2\pi}{c} \mathbf{a} \times \mathbf{b} \quad (3.9)$$

where a , b and c are the lattice constants of the crystal. The reciprocal space is therefore the space of wave vectors, q , which are related to the scattering angle, θ .^{171,172}

An Ewald sphere can be constructed by considering fully elastic scattering, where the wave vector magnitude $|\mathbf{k}_1|$ is equal to the incident wave vector magnitude $|\mathbf{k}_0|$.

$$|\mathbf{k}_1| = |\mathbf{k}_0| = \frac{2\pi}{\lambda} \quad (3.10)$$

where λ is the wavelength of the X-rays and the diffracted wavevector being 2θ . Cycling the diffracted wavevector through different angles, θ , will give a 3D representation of the Ewald sphere of radius $q = \frac{2\pi}{\lambda}$. Diffraction will thus only be seen for wavevectors which lie on the Ewald sphere surface. Figure 3.7 shows a representation of the Ewald sphere for X-ray scattering.

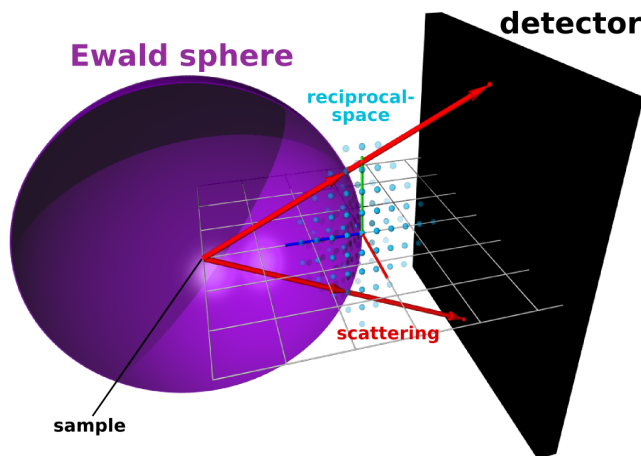


Figure 3.7: The Ewald sphere for X-ray scattering, figure from Kevin Yager¹⁷³

Powder Diffraction Patterns of Lipids

We have considered the scattering of X-rays off a crystal lattice. However, whilst some lipid phases are crystalline, most are not. As such, the scattering of X-rays off a lipid bilayer is more akin to powder diffraction. The scattering of X-rays off a powder is dependent on the size of the particles in the powder and the wavelength of the X-rays. As the entropy of a powder is larger than that of a crystal, individual domains are randomly oriented and the scattering is therefore isotropic (the same in all directions). As such, all reciprocal lattice points are equally likely to be scattered. The resulting scattering pattern is therefore a sum of the scattering from all reciprocal lattice points, which forms a ring pattern⁷³ as shown in Figure 1.6. These

rings are then usually integrated to give the scattering intensity as a function of q using the butterfly method.⁶¹ This results in a 1D scattering pattern, with the scattering intensity being proportional to the square of the scattering vector, q , where $q = 4\pi \sin \theta / \lambda = 2\pi / d_{hkl}$.

From this 1D scattering pattern, the features of the bilayer can be determined. The peak positions give insight into the phase of the bilayer as they are dependent on the shape of the unit cell and the spacing between the layers. The peak positions are entirely based on the lattice plane spacing which can be derived through the Miller indices of the lattice (hkl).¹⁷²

The peak intensities are dependent on structural factors like atom positions and type (electron density) within the unit cell as well as the effect of environmental factors such as thermal fluctuations which are accounted for in a dampening effect known as the Debye-Waller factor (DWF). The DWF effect increases for larger q or θ values. The DWF is dependent on the temperature of the sample and the atomic displacement parameters (ADP) of the atoms in the unit cell. The ADP is a measure of the thermal motion of the atoms in the unit cell.¹⁷⁴ The DWF is given by:

$$DWF = \exp \left(-\frac{q^2 \sum_{i=1}^3 \beta_i^2}{3} \right) \quad (3.11)$$

where β_i is the ADP of the i^{th} atom in the unit cell. Therefore the effective intensity is given by:

$$I(\mathbf{q})_{eff} = I_0 \exp(-\beta \mathbf{q}^2) \quad (3.12)$$

where I_0 is the initial peak intensity and β is the DWF component based on thermal fluctuations, known as the dampening factor.¹⁵⁰

The peak profile is dependent on the size of the crystal lattice. For an infinitely large crystal, the peak profile is a sharp dirac-delta function. However for a finite crystal with non-monochromatic X-rays, the peak profile is a Gaussian/Lorentzian/voigt function mixture, which gives a peak more width/breadth.^{172,175}

3.3 Simulating SAXS Patterns by Modelling Lipids

In this section, the methods used to simulate SAXS patterns for lipids are described. Since the aim of this chapter is to develop a model that predicts the phase of a lipid bilayer from its SAXS pattern, a large dataset of SAXS patterns for lipids is required to train the model.

As there are no large public datasets of SAXS patterns for lipids, the following methods were developed to simulate SAXS patterns for lipids such that the ML model could be pre-trained. This is done by modelling the lipid bilayer as a crystal lattice. The lattice is then populated with atoms from the lipid molecules. The scattering of X-rays off the atoms is then calculated and the scattering pattern is generated.

In general, for each phase, meshed electron density models were built in both the real and Fourier space. The general model equation describing the intensity of SAXS at a given wavevector is given by:

$$I(\mathbf{q}) = |F(\mathbf{q})|^2 S(\mathbf{q}) \quad (3.13)$$

where $F(\mathbf{q})$ is the Fourier transform of the electron density scattering model ($\rho(x)$) of a single unit cell (this is also known as the form factor), and $S(\mathbf{q})$ is the structure factor of the lattice. The structure factor is the real space lattice $s(x)$ in reciprocal space $S(\mathbf{q})$, and therefore consists of discrete, well-defined peaks constructed from delta functions.

3.3.1 Lamellar Phase Modelling

The lamellar phase Bravais lattice was modelled in 1D, with the unit cell along x being modelled by 3 Gaussians decorating the bilayer head group positions $\pm \frac{L}{2}$, where L is the distance of the headgroups from the centre of the bilayer. This model, with a lattice parameter α , for n points gives a lattice defined by:

$$s(x) = \sum_{n=-\infty}^{\infty} \delta(x - \alpha n) \quad (3.14)$$

where $\delta(x)$ is the delta function. The Fourier transform of this lattice, $\mathcal{F}(s) = S(q_x)$ gives the structure factor in one frequency dimension, k_x . The structure factor is then given by:

$$S(q_x) = \frac{2\pi}{\alpha} \sum_{m=-\infty}^{\infty} \delta(q_x - m \frac{2\pi}{\alpha}) \quad (3.15)$$

where $\delta(q_x)$ is the delta function in reciprocal space.

Next, the unit cell electron density was modelled as the first step to simulating the form factor. As mentioned previously this was modelled by generating 3 Gaussians decorating the bilayer head groups. This gave a unit cell electron density model of:

$$\rho(x) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2} \left(\frac{x \pm \frac{L}{2}}{\sigma_1}\right)^2\right) - \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2} \left(\frac{x}{\sigma_2}\right)^2\right) \quad (3.16)$$

where σ_1 and σ_2 are the standard deviations of the Gaussians. The Fourier transform of a Gaussian is the identity matrix product therefore, $\mathcal{F}(\rho(x)) = F(\mathbf{q})$ gives the form factor in one frequency dimension, k_x . The form factor is then given by:

$$F(q_x) = \exp(-2\pi^2\sigma_1^2q_x^2 + i\pi L) - \exp(-2\pi^2\sigma_2^2q_x^2) \quad (3.17)$$

combining the structure factor and form factor as per equation 3.13 gives the intensity of the lamellar phase as:

$$I(q_x) = 2\pi\alpha^* \sum_{m=-\infty}^{\infty} \delta(q_x - 2\pi m\alpha^*) \left(\exp(-2\pi^2\sigma_1^2q_x^2 \pm i\pi L) - \exp(-2\pi^2\sigma_2^2q_x^2) \right) \quad (3.18)$$

where α^* is the reciprocal lattice parameter. Figure 3.8 visualises the synthetic model structure and its parameters for a one-dimensional lamellar phase.

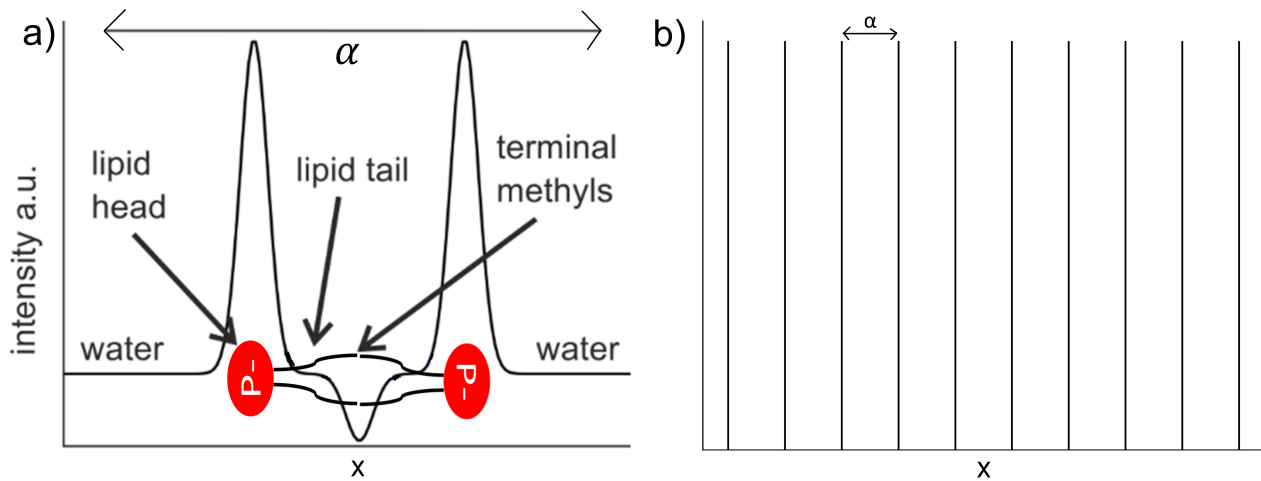


Figure 3.8: The synthetic model for a one-dimensional lamellar phase was created by convolving the reciprocal unit cell with the lattice. The lattice, b) is composed of a delta array with a lattice parameter of α . The electron density of the unit cell, a) was modelled using three Gaussian functions, which consider the relative positions of the headgroup and terminal methyl lipid ends in relation to the electron density of water.

Whilst this model now accurately describes the intensity of the lamellar phase, it does not account for the width of the peaks, or for dampening effects such as the Debye-Waller factor. The width of the peaks is due to the finite size of the crystal lattice, and the Debye-Waller factor is due to thermal fluctuations. The former was accounted for in the data processing step by convolving the simulated pattern with a Gaussian/Lorentzian/pseudo-voigt function mixture (see section 3.4), and the latter was accounted for by multiplying the simulated pattern by the Debye-Waller factor.

$$I_{DWF}(q_x) = I(q_x) \exp(-q_x^2 \Delta) \quad (3.19)$$

where Δ is the Debye-Waller factor. The amplitude and lattice peaks for the 1D lamellar phase model are shown in figure 3.9.

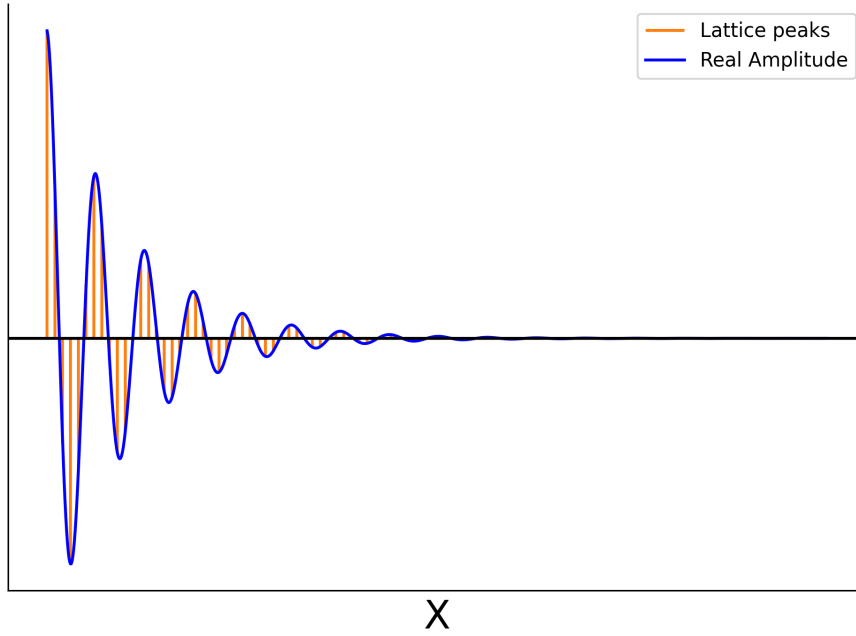


Figure 3.9: A 1D lamellar phase SAXS model was created by convolving the reciprocal unit cell with the lattice peaks and multiplying their power spectra with reciprocal lattice parameter α^* . This is demonstrated by overlaying the real space amplitudes of the unit cell with the lattice peaks.

3.3.2 Hexagonal Phase Modelling

The hexagonal phase consists of cylindrical micelles packed hexagonally along a 3rd dimension, z . Due to its symmetrical, cylindrical nature, the hexagonal phase can be modelled in 2D. This is shown in figure 3.10.

The 2D basis vector of the hexagonal lattice is given by:

$$H(\mathbf{x}) = \alpha(n + m\frac{1}{2})\hat{x} + \alpha m\frac{\sqrt{3}}{2}\hat{y} \quad (3.20)$$

where α is the lattice parameter, and $\mathbf{x}$ is the position vector. $\hat{x}$ and $\hat{y}$ are the unit vectors in the x and y directions respectively. The reciprocal lattice vector is given by:

$$S(q_x, q_y | \alpha^*) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \delta\left(q_x - \alpha^* \left(n + m\frac{1}{2}\right)\right) \delta\left(q_y - \alpha^* \left(m\frac{\sqrt{3}}{2}\right)\right) \quad (3.21)$$

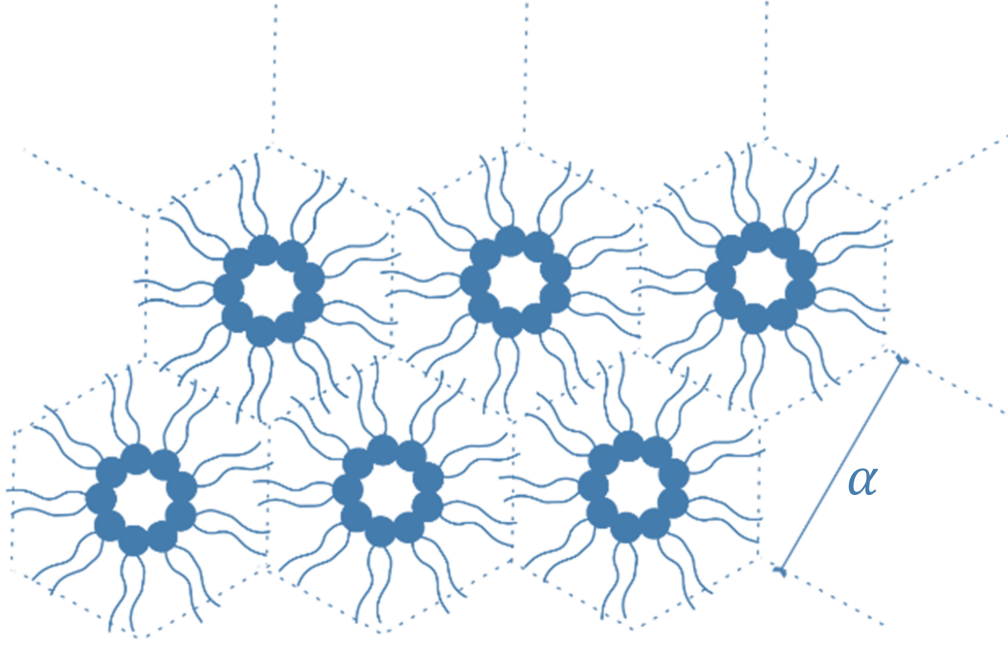


Figure 3.10: A 2-dimensional representation of the hexagonal lipid phase. The lattice consists of cylindrical micelles packed hexagonally along the z axis. The lattice parameter α is the distance between the centres of the micelles.

This reciprocal lattice can be collapsed onto the radial scattering vector, q_r , by taking the Euclidean norm of the reciprocal lattice vector, q_x and q_y :

$$S(q_r) = |S(q_x, q_y)|^2 \quad (3.22)$$

This way we have a convenient 1D representation of the hexagonal phase by taking the set of reciprocal lattice points along the radius from the origin.

The unit cell of the hexagonal phase is modelled in polar coordinates with two Gaussians:

$$\rho(r, \theta) = I_1 \frac{1}{\sqrt{2\pi\sigma_h^2}} \exp\left(-\frac{1}{2} \left(\frac{r - r_h}{\sigma_h}\right)^2\right) - I_2 \frac{1}{\sqrt{2\pi\sigma_l^2}} \exp\left(-\frac{1}{2} \left(\frac{r - r_l}{\sigma_l}\right)^2\right) \quad (3.23)$$

where r is the radial distance from the origin, θ is the angle from the x -axis, I_1 and I_2 are the intensities/weights of the two Gaussians, and σ_h and σ_l are the standard deviations of the two Gaussians. The two Gaussians are separated by a distance r_h and r_l . In this model $I_1 = 1$ and $I_2 = 0.2$ respectively.

Baddour¹⁷⁶ states that the fourier transform of the electron density $\mathcal{F}(\rho)$ is a function of the frequency radius k and angle ψ :

$$F(k) = \int_{-\pi}^{\pi} \int_0^{\infty} \rho(r, \theta) \exp(-ikr \cos(\psi - \theta)) r dr d\theta \quad (3.24)$$

This function is radially symmetric as $\rho(r, \theta) = \rho(r)\rho(\theta)$, and therefore the Fourier transform is also radially symmetric. Therefore:

$$F(k) = \int_0^{\infty} r \rho(r) \int_{-\pi}^{\pi} \exp(-ikr \cos(\psi - \theta)) d\theta \quad (3.25)$$

This function is very similar to the zeroth order Bessel function of the first kind, $J_0(kr)$. Therefore we can use it to simplify the Fourier transform. The zeroth order Bessel function¹⁷⁷ of the first kind is defined as:

$$J_0(x) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \exp(ix \cos(\psi - \theta)) d\theta = \frac{1}{2\pi} \int_{-\pi}^{\pi} \exp(-ix \cos(\alpha)) d\alpha \quad (3.26)$$

The fourier transform of the electron density is therefore simplified to:

$$F(k) = \int_0^{\infty} r \rho(r) J_0(kr) dr \quad (3.27)$$

This integral is known as the Hankel transform¹⁷⁸ of the electron density. The Hankel transform is defined as:

$$\mathcal{H}(f)(k) = \int_0^{\infty} r f(r) J_0(kr) dr \quad (3.28)$$

where $f(r)$ is the function to be transformed. The Hankel transform is a special case of the Fourier transform, and can also be radially symmetric. This leads $F(k)$ to be:

$$F(k) = 2\pi\mathcal{H}_0(\rho) \quad (3.29)$$

This can be solved numerically to give the scattering intensity $F(k)$ and therefore hexagonal intensity in accordance with equation 3.13.

3.3.3 Inverse Bi-continuous Cubic Phase Modelling

As mentioned previously, the cubic phases are based on the TPMS Schwarz minimal surfaces. The three specific cubic phases are: Primitive (P), Gyroid (G) and Diamond (D). Figure 3.6 shows the three surfaces which are rare and somewhat difficult to observe in nature with SAXS.^{164,179}

The electron density of these cubic phases is geometrically complex due to their shape. Works by Harper et al.,¹⁶⁴ Gandy et al.¹⁸⁰ and Garstecki et al.^{181–184} have shown that the reciprocal electron density of the cubic phases (the scattering amplitude) can be theoretically modelled by:

$$F(q) = \int \rho(x) \exp[iq \cdot x] dx \quad (3.30)$$

which gives intensity, $I(\mathbf{q})$ of:

$$I(\mathbf{q}) = \left[\sum_{j=1}^N s_j \cos(\mathbf{k}\mathbf{r}_j) \exp\left(-\frac{1}{2}(\mathbf{k}\mathbf{n}_j)\langle u^2 \rangle\right) \frac{\sin(\mathbf{k}\mathbf{n}_j \frac{L}{2})}{\mathbf{k}\mathbf{n}_j \frac{L}{2}} \right]^2 + \left[\sum_{j=1}^N s_j \sin(\mathbf{k}\mathbf{r}_j) \exp\left(-\frac{1}{2}(\mathbf{k}\mathbf{n}_j)\langle u^2 \rangle\right) \frac{\sin(\mathbf{k}\mathbf{n}_j \frac{L}{2})}{\mathbf{k}\mathbf{n}_j \frac{L}{2}} \right]^2 \quad (3.31)$$

where L is the distance between the lattice parameter from $-\alpha$ and α , $\mathbf{n}$ is the normal vector to the surface, $\mathbf{r}$ is the position vector, s_j is the surface area of the triangle along the meshed cubic surface, and $\langle u^2 \rangle$ is the mean square displacement of the surface. This displacement is a

way to account for fluctuations within the surface and is calculated from the theory of elasticity in membranes by Bruinsma.¹⁸⁵ The equation for the mean square displacement is:

$$\langle u^2 \rangle = \frac{k_B T d^3}{\sqrt{\kappa \bar{\kappa}}} \left(\frac{\Gamma_1}{\sqrt{1 + \Gamma_2 d^2 k_B T / \bar{\kappa}}} + \frac{\Gamma_3}{1 - \Gamma_4 d^2 k_B T / \bar{\kappa}} \right) \quad (3.32)$$

where k_B is the Boltzmann constant, T is the temperature, d is the thickness of the membrane, κ is the bending rigidity, Γ_1 - and Γ_4 are constants. $\bar{\kappa}$ is the Helfrich bending energy where $\bar{\kappa} \sim \kappa L^2$.

Two modifications are made to the model of the TPMS surface intensity:

1. It is assumed that there are no fluctuations in the surface
2. A Gaussian function is convolved with two Kronecker delta functions to accent the surface as opposed to Garstecki et al.'s model¹⁸³ which assumes a rectangular bilayer function.

The Kronecker delta functions are used to accent the surface and are defined as:

$$f_k d(x) = \delta(x - \alpha) + \delta(x - \alpha) \quad (3.33)$$

With a distance L between the lattice parameter from $-\alpha$ and α , the Fourier transform of the Kronecker delta functions is:

$$F_{kd}(\mathbf{q} \cdot \mathbf{n}_j) = 2 \cos \left(\mathbf{q} \cdot \mathbf{n}_j \frac{L}{2} \right) \quad (3.34)$$

The Fourier transform of the Gaussian part is:

$$F_g(\mathbf{q} \cdot \mathbf{n}_j) = \exp \left(-2\pi^2 \sigma^2 (\mathbf{q} \cdot \mathbf{n}_j)^2 \right) \quad (3.35)$$

where σ is the standard deviation of the Gaussian function. The final intensity after assuming no fluctuations and with the Kronecker delta-Gaussian decoration is therefore derived by substituting the Fourier transforms into equation 3.30 and using Euler's formula ($e^{ix} = \cos(x) + i \sin(x)$) to give:

$$I(\mathbf{q}) = \left[2 \sum_{j=1}^N s_j \cos(\mathbf{q} \cdot \mathbf{r}_j) \cos\left(\mathbf{q} \cdot \mathbf{n}_j \frac{L}{2}\right) \exp(-2\pi^2 \sigma^2 (\mathbf{q} \cdot \mathbf{n}_j)^2) \right]^2 + \left[2 \sum_{j=1}^N s_j \sin(\mathbf{q} \cdot \mathbf{r}_j) \sin\left(\mathbf{q} \cdot \mathbf{n}_j \frac{L}{2}\right) \exp(-2\pi^2 \sigma^2 (\mathbf{q} \cdot \mathbf{n}_j)^2) \right]^2 S(\mathbf{q}) \quad (3.36)$$

Computationally Generating the TPMS surfaces

Whilst the Schwarz TPMS minimal surfaces are mathematically defined, they are not computationally tractable. In order to optimise the data generation process for creating the simulated data of the cubic phases, the trigonometric volume approximations of the surfaces are used to generate them. For each shape the trigonometric volume approximation is defined as:

- **Primitive:** $V(x, y, z) = \cos(x) + \cos(y) + \cos(z)$
- **Gyroid :** $V(x, y, z) = \sin(x) \cos(y) + \sin(y) \cos(z) + \sin(z) \cos(x)$
- **Diamond :** $V(x, y, z) = \sin(x) \sin(y) \sin(z) + \sin(x) \cos(y) \cos(z) + \cos(x) \sin(y) \cos(z) + \cos(x) \cos(y) \cos(z)$

The Grid array consisting of the coordinates of the vertices of the volume were generated by NumPy's *meshgrid* function.⁹ The volume was then computed by evaluating the trigonometric volume approximation at each vertex of the volume.

With the volumes of the surfaces computed, the Marching Cubes algorithm is used to generate the surface meshes. The Marching Cubes algorithm is a surface reconstruction algorithm that generates a polygonal mesh from a 3D scalar field. The algorithm is based on the idea of marching through the volume and interpolating the surface between the vertices of the volume. The

algorithm is described in detail in Lorensen and Cline's work.¹⁸⁶ The algorithm is implemented in the Python package *scikit-image*¹³ which is based off Lewiner's C++ implementation.¹⁸⁷

3.4 SAXS Data Analysis and Processing

Using the above three models for SAXS pattern generation, the model's parameters are determined by matching the patterns with the experimental data using the Levenberg-Marquardt algorithm, a non-linear least squares algorithm that is used to find the minimum of a function. The algorithm is described in detail in Levenberg and Marquardt's work.^{188,189} The algorithm is implemented in the Python package *scipy*.⁸

Figure 3.11 shows a sample of the raw theoretically modelled SAXS peaks for the 5 phases as modelled above, with the selected optimised parameters for matching the sparsity of the common experimentally observed patterns.

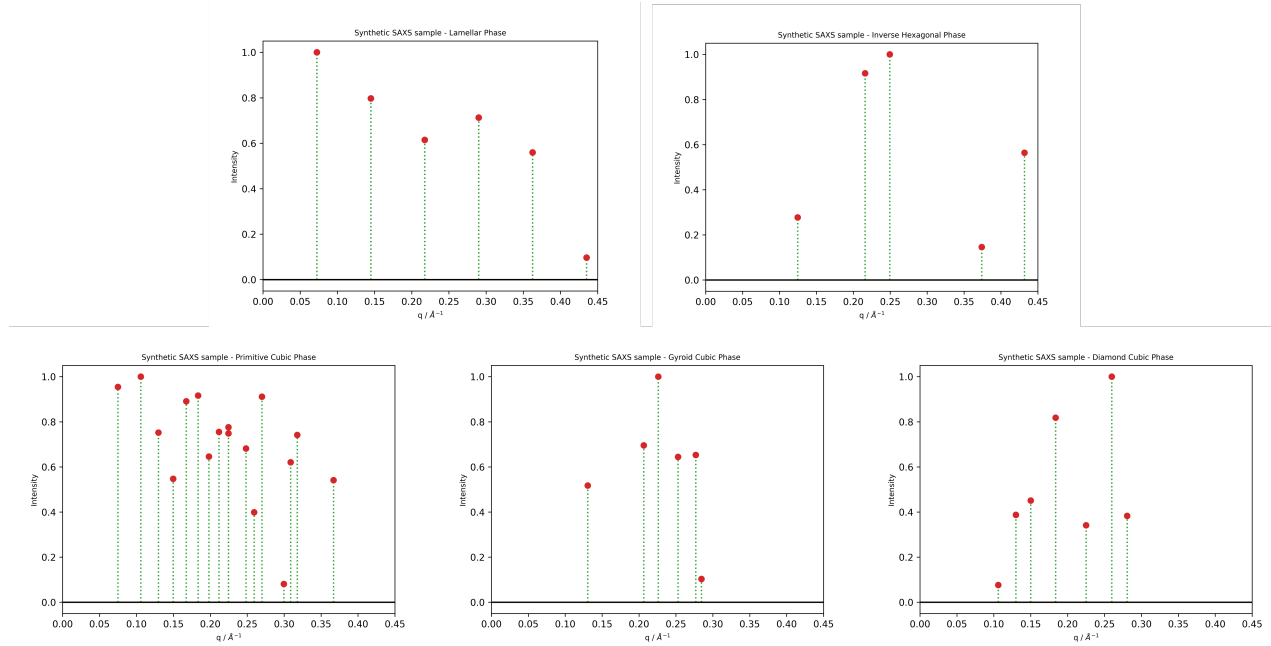


Figure 3.11: Raw SAXS peaks for the 5 phases - Lamellar, Hexagonal, Primitive, Gyroid and Diamond

SAXS Model parameters

The parameter ranges used to generate the SAXS patterns are shown in Table 3.1.

Phase	Lattice Parameter, α	Lipid Length/Head Position ($\text{\AA}$)	Head σ	Tail σ
Lamellar	40-104	25-35	1-3	2-4
Hexagonal	20-78	5-30	0.5-3	0.5-5
Primitive	5-30	1-5	0.0001-0.01	0.5
Gyroid	5-30	1-5	0.0001-0.01	0.5
Diamond	5-30	1-5	0.0001-0.01	0.5

Table 3.1: SAXS model parameter ranges used to generate the Synthetic datasets

These parameter ranges were cycled through in equal steps of 10 along the range. These ranges were chosen such that the space of the synthetically generated data covered that of the experimentally observed SAXS patterns, whilst limiting the space covered by the synthetic data such that the ML experimental data available for fine-tuning was not sparsely populating the training data, but rather the ML model capacity was focused on the regions of the parameter space where the experimental data was common.

In total, 20,000 patterns were generated for each Lamellar and Hexagonal phases, and 11,127 patterns were generated for each the Primitive, Gyroid and Diamond phases. The total number of patterns generated for the synthetic dataset was 73,381.

Synthetic Data Processing and Featurisation

Before the generated synthetic patterns could be used, additional experimental features were added such that they were similar enough to the experimental data to be used in the ML model. The additional features added were, a Voigt profile to each peak such that the peak breadth was similar to that of the experimental data; and the DWF exponential decay effect.

The Voigt signal is a convolution of a Gaussian and a Cauchy-Lorentzian signal. The Voigt signal is given by:

$$V(q) = \frac{\text{Re}[w(z)]}{\sigma\sqrt{2\pi}} \quad (3.37)$$

where $w(z)$ is the Faddeeva function, $z = \frac{q+i\gamma}{\sigma\sqrt{2}}$ and σ is the standard deviation of the Gaussian signal. The Faddeeva function is a complex function that is used to model the Voigt signal.¹⁹⁰ The Faddeeva function is given by:

$$w(z) = \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-t^2} e^{(z-t)^2} dt \quad (3.38)$$

The Voigt profile is implemented in the Python package *SciPy*.⁸

Whilst this accounts for most experimental features, it does not account for some of the noise features from experimental data. This includes the distinct broad hump usually found at around $q = 0.4 \text{ \AA}^{-1}$, due to the presence of poly-carbonate capillaries which are used as the container for the lipid samples. This is resolved by convolving the synthetic data with 'blank' experimental samples and adding Gaussian noise. These blank samples are real SAXS samples taken with an empty poly-carbonate capillary. Figure 3.12 shows a sample of the blank samples used for convolution.

With this, we now have the final synthetic data generated and processed, ready for training the ML model. Some samples of the fully featured synthetic data are shown in Figure 3.13.

These samples include the Voigt profile, the DWF exponential decay effect, the broad hump at $q = 0.4 \text{ \AA}^{-1}$ from experimental blanks and some Gaussian noise.

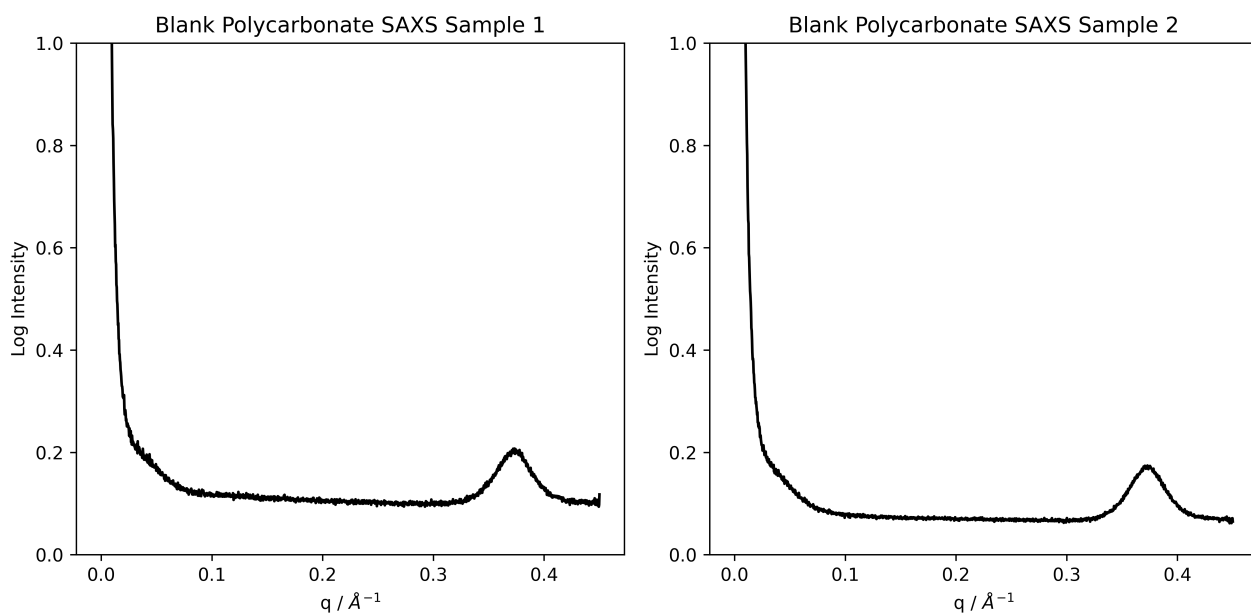


Figure 3.12: Sample of blank samples used for convolution

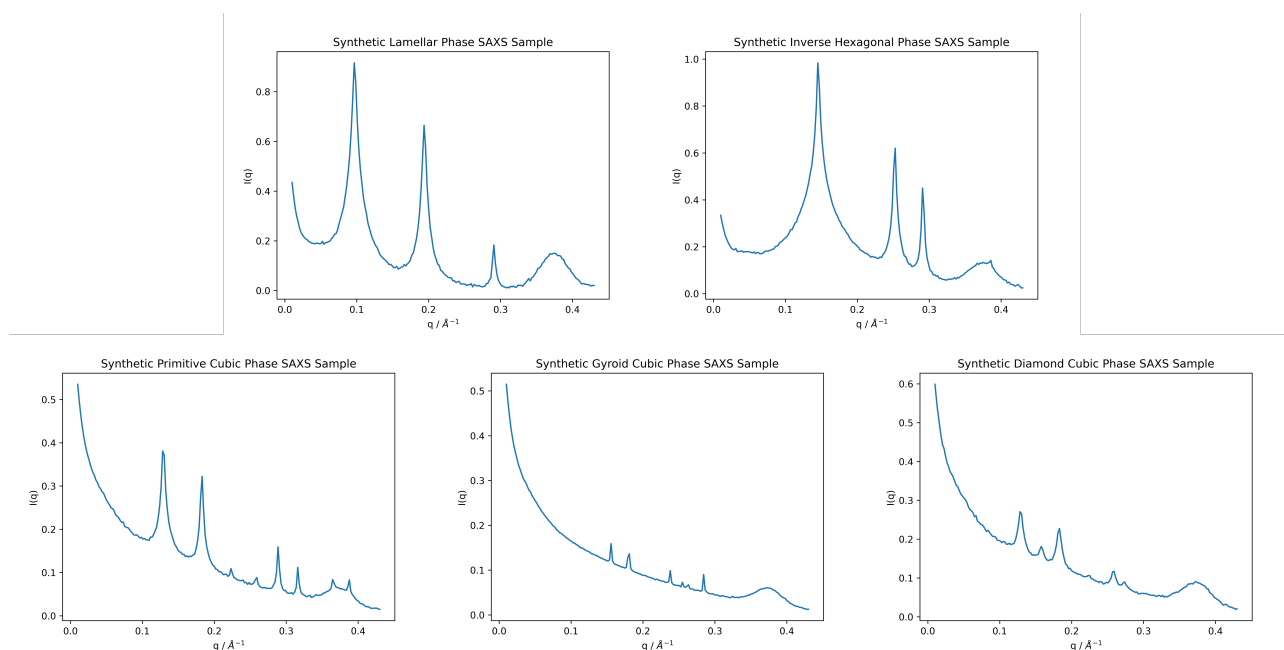


Figure 3.13: Sample of the final synthetic data

In order to understand the effect of the synthetic data on the ML model, the feature space of the synthetic data was compared to that of the experimental data. The feature space of the synthetic data was generated by using the same featurisation method as that used for the experimental data. The feature space of the experimental and synthetic data for lamellar and hexagonal is shown in Figure 3.14.

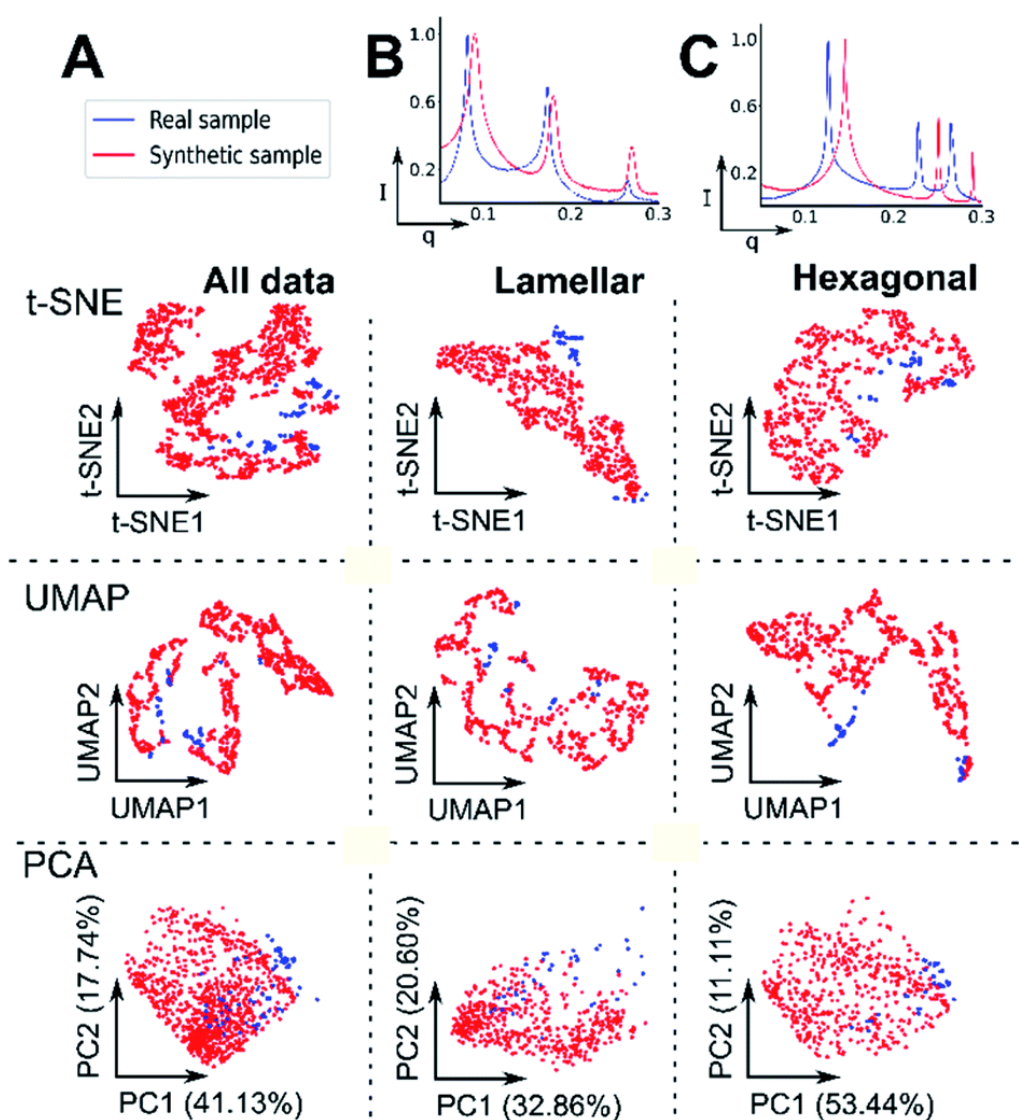


Figure 3.14: Feature space of the synthetic data compared to that of the experimental data¹⁹¹

Three dimensionality reduction techniques were used to visualise the feature space between the two sets: Principal Component Analysis (PCA), t-distributed Stochastic Neighbour Embedding (t-SNE) and Uniform Manifold Approximation and Projection (UMAP). PCA is a linear dimensionality reduction technique that is used to reduce the dimensionality of the data whilst preserving the variance of the data. t-SNE is a non-linear dimensionality reduction technique that minimises the Kullback-Leibler divergence between the probability distributions of the high-dimensional data and the low-dimensional data. UMAP is a non-linear dimensionality reduction technique that uses a manifold learning approach to embed the data into a low-dimensional space.

In all three cases the synthetic data feature space is found to cover that of the experimental

samples, as well as an extended area of the feature space that is not covered by the experimental data. There are roughly two clusters created by the real data (hexagonal and lamellar) in the t-SNE plot, however there are more than two clusters formed by the synthetic data. This suggests that there is significant intraphase variation in the synthetic data, which is not present in the experimental data. This is likely due to the fact that the synthetic data is generated from a ranged parameter space, whereas the experimental data is generated from a fixed parameter space. This is further supported by the fact that the synthetic data feature space is found to cover a larger area of the feature space than the experimental data. This suggests that the synthetic data is more representative of the total feature space of SAXS spectra than the more sparse experimental data, whilst still being representative of the experimental data as it is within the convex hull of the experimental data.

Data Preparation and Featurisation for ML

Now that the synthetic data has been generated and processed, it is almost ready to be used to pre-train ML model. Firstly, both the synthetic and experimental samples needed further processing before feeding into the model architecture.

As the experimental data was largely unlabelled, manual labelling of the experimental spectra was necessary. In order to aid the experimental scientists with this, a web app was built with Flask¹⁹² and Bokeh¹⁹³ to allow experimentalists to easily label the lipid phase observed in their SAXS spectra. This allowed the creation of the experimental classification training and test set, as well as identifying the blank poly-carbonate peak only spectra for synthetic feature decoration. Figure 3.15 shows a screenshot of the web app. The SAXS spectrum is displayed along with phase options or a pass option which takes the user to the next unlabelled spectrum. Hovering over the spectrum gave the q value at the mouse cursor's current position.

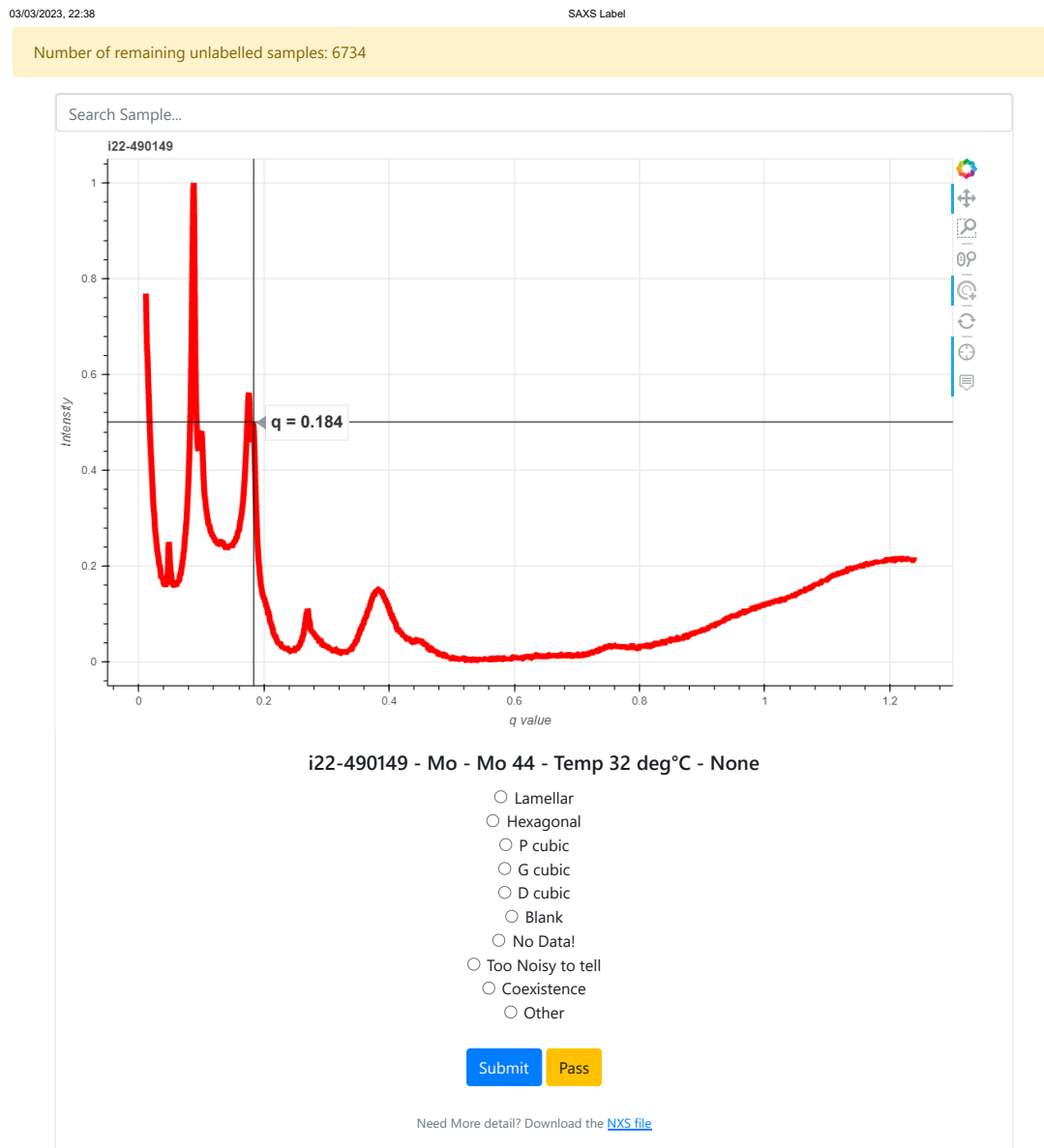


Figure 3.15: A screenshot of the experimental SAXS labelling web app

The data needed to be of fixed length such that it could be fed into the model. Whilst the synthetic data was already of fixed length, the experimental data was not, since each sample

had a different resolution and therefore a different number of data points for the same q range. To resolve this, Cubic spline interpolation¹⁹⁴ was used to interpolate the experimental data to a fixed length. Cubic spline interpolation is a method of interpolation that is used to fit a curve to a set of data points. The curve is a piecewise cubic polynomial, which is constructed by joining the data points with a series of cubic polynomials. The curve is given by:

$$S(x) = \sum_{i=1}^n a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \quad (3.39)$$

where a_i , b_i , c_i and d_i are the coefficients of the cubic polynomial, and x_i are the data points. The coefficients are given by:

$$a_i = \frac{1}{6h_i}(y_{i+1} - y_i) - \frac{1}{3h_i}(2x_{i+1} - x_i) \quad (3.40)$$

$$b_i = \frac{1}{2h_i}(y_i - y_{i+1}) + \frac{1}{6h_i}(x_{i+1} - x_i) \quad (3.41)$$

$$c_i = \frac{1}{2h_i}(x_{i+1} - x_i) \quad (3.42)$$

$$d_i = y_i \quad (3.43)$$

where $h_i = x_{i+1} - x_i$. The data was interpolated to a fixed length of 200 data points. Which was the maximum number of data points for the lowest resolution experimental sample. The Cubic spline interpolation implementation used was the one in the Python package *SciPy*.⁸

The data intensity was also normalised to avoid weight spikes such that the model could be trained effectively. Min-Max normalisation was used to normalise the data. Min-Max normalisation is a method of normalisation that is used to scale the data to a fixed range. The data is normalised to the range $[0, 1]$ such that the minimum value is 0 and the maximum is 1.

The normalised data is given by:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (3.44)$$

where x_{norm} is the normalised data, x is the original data, x_{min} is the minimum value of the original data and x_{max} is the maximum value of the original data.

Finally, the data is split into training, validation and test sets, with the training sets being used to pre-train (synthetic) and fine-tune (experimental) the ML model and the validation/test sets being used to validate the ML model. The data splits are shown in Table 3.2.

Dataset	Phase				
	Lamellar	Hexagonal	Primitive	Gyroid	Diamond
Pre-train	10,000	10,000	6,040	6,040	6,040
Fine-tune	40	40	40	40	40
Valid	10	16	40	40	40
Test	20	17	7	7	7
Synthetic test	10,000	10,000	5,000	5,000	5,000

Table 3.2: Data splits for the SAXS data

Whilst Lamellar and Hexagonal experimental data is relatively abundant, the experimental data for cubic phases is scarce. In general the synthetic data in the pre-training steps gives the ML model some background and context to the data, which helps the ML model to learn the features of the data. However, the synthetic data is not a perfect representation of the experimental data, and overfitting to the synthetic data becomes a real issue. This is further discussed in Section 3.5. It is also important to note, that due to the scarcity of experimental cubic phase data, both the pre-train and fine-tune sets for the cubic phases are synthetic, with the validation sets also being synthetic data.

Furthermore, the feature vector of each sample, $\mathbf{i}$, is upscaled to a 2D matrix by taking the outer product of the feature vector with itself. This multi-dimensional spatial separation is done to allow the ML model to learn better correlations between the features. The 2D matrix is given by:

$$\mathbf{M} = \mathbf{i} \otimes \mathbf{i} = \mathbf{ii}^T = \begin{bmatrix} i_1^2 & i_1 i_2 & \dots & i_1 i_n \\ i_2 i_1 & i_2^2 & \dots & i_2 i_n \\ \vdots & \vdots & \ddots & \vdots \\ i_n i_1 & i_n i_2 & \dots & i_n^2 \end{bmatrix} \quad (3.45)$$

where $n = 200$ is the number of data points in the feature vector. This also scales the values of the feature vector to the range $[0, 1]$ such that the convolutional ML model is more easily able to abstract the nuanced peak within the data. This representation can also be visualised as a 2D image, where it shows peak pattern relationships as squares, with shoulder peaks and broad peaks as an aura. A sample for each phase is shown in Figure 3.16.

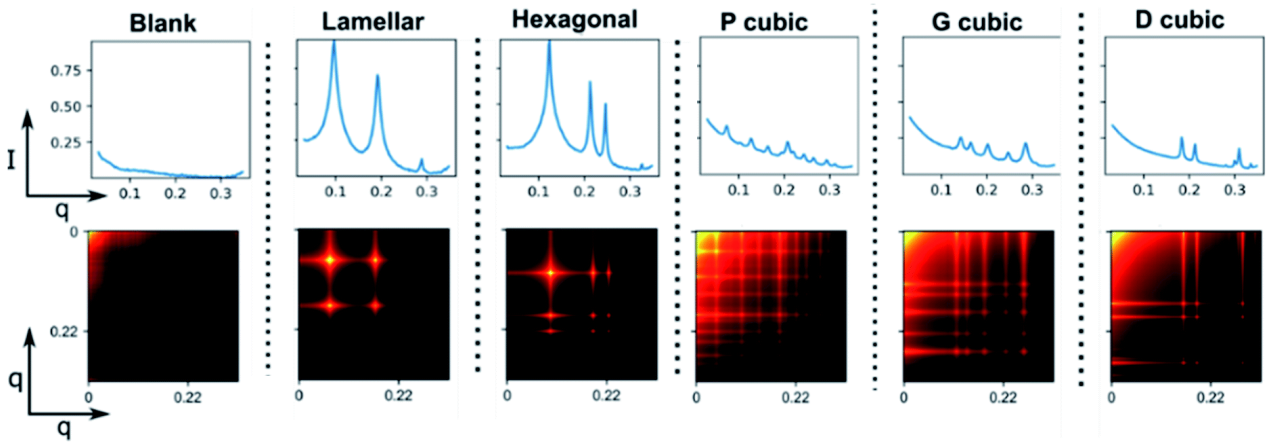


Figure 3.16: Feature matrix for each phase, adapted from own work¹⁹¹

In essence, phases with sharp, well defined peaks, have well-defined square/rectangle like shapes, whereas phases with low intensity peaks, have a more blurred shape. This is evident in both

the synthetic and experimental data with this representation.

Furthermore, given that lipid phases in nature are usually a continuous and dynamic system which can change as mentioned previously, this representation is flexible enough to be used to represent the coexistence of multiple phases as it duplicates the data across 2 dimensions. Figure 3.17 shows an example of the feature matrix for a coexistent phase, where the feature matrix is a combination of the feature matrices of an inverse hexagonal and lamellar phase. Future works should explore the use of this representation for coexistent phases for ML purposes or otherwise.

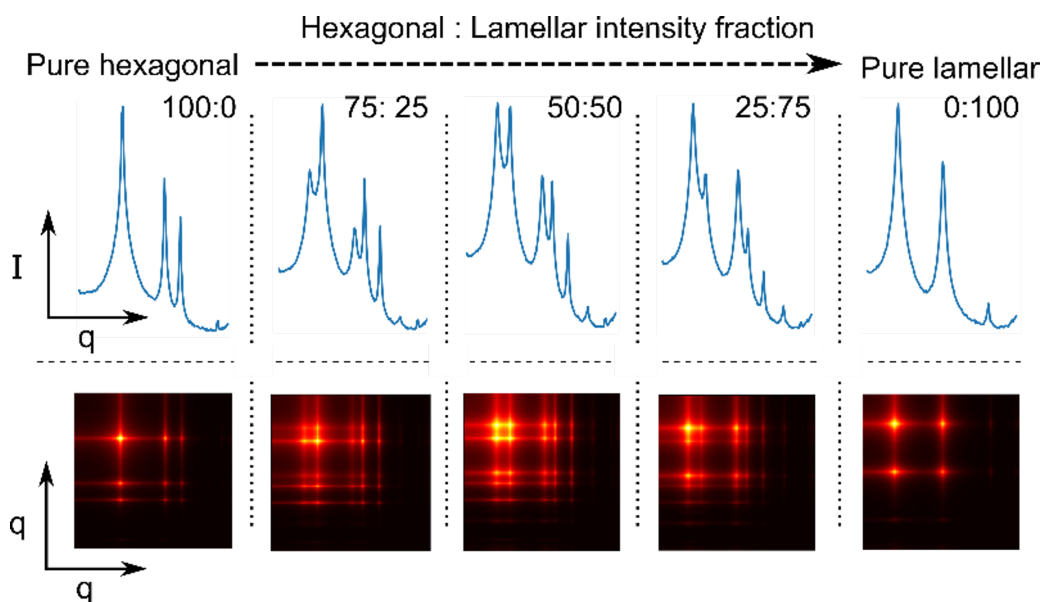


Figure 3.17: Feature matrix for coexistent lipid phases - lamellar and inverse hexagonal, adapted from own work¹⁹¹

3.5 Pipeline Workflow

Now that we have the synthetic data and the experimental data featurised and in a format that can be used by the ML model, the model architecture and the training workflow can be considered. This pipeline will architect the CNN by pre-training the model on the synthetic data, fine-tuning the model on the experimental data and evaluating the model's performance on predicting lipid phase behaviour from SAXS spectra, which is the aim of this chapter. Figure 3.18 shows the overall pipeline from data generation to fine-tuning.

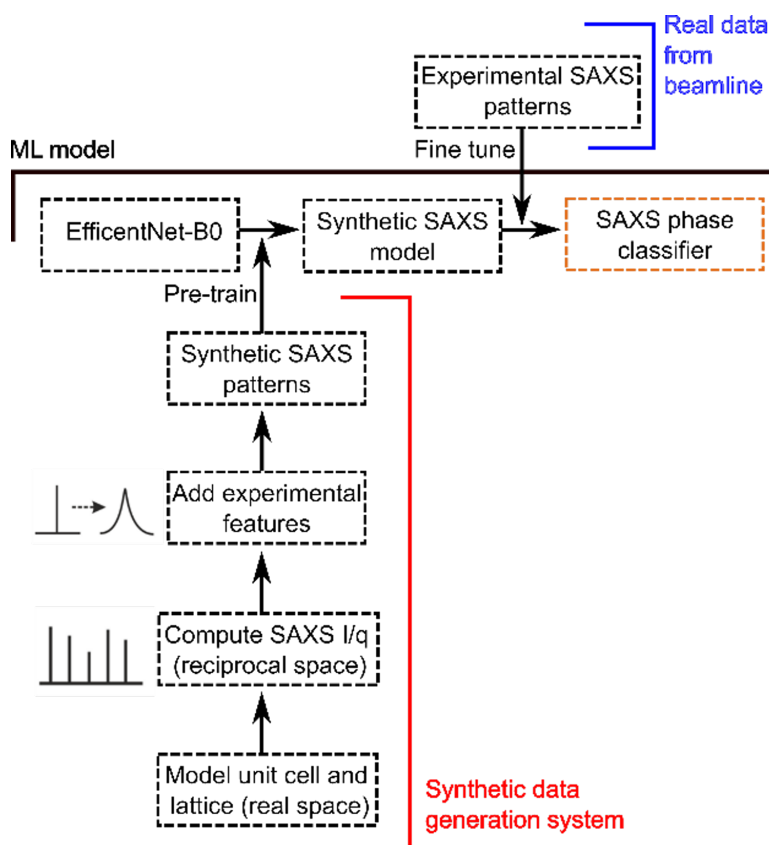


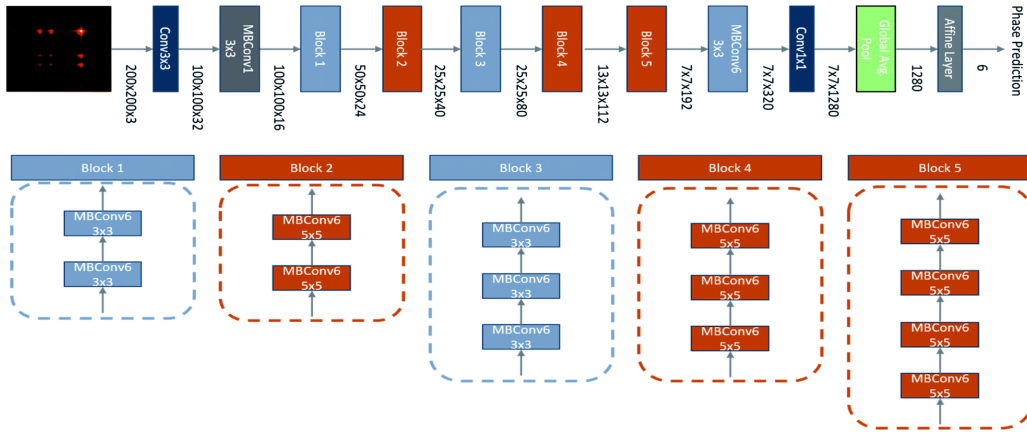
Figure 3.18: Full SAXS pipeline workflow, from synthetic data generation and experimental data acquisition/labelling to model pre-training and fine-tuning.¹⁹¹

3.5.1 Model Architecture

One of the main considerations when deciding on the model architecture, was to minimise inference time such that scientists can use the model to make predictions that will aid their experimental plan on-site at the beamline.

As such, EfficientNet-B0 was used as our basis model.⁴ It is a convolutional neural network that is designed to be efficient in terms of both memory and computational resources. It is the smallest model in the EfficientNet family, and is designed to be used for mobile and fast-paced applications. Its architecture was determined by a combination of neural architecture search and a large-scale empirical study. The model architecture and data flow is shown in Figure 3.19.

This model utilises mobile inverted bottleneck convolution blocks (MBConv).^{195,196} These blocks perform a combination of depthwise separable convolutions and standard convolutions. Depth-

Figure 3.19: EfficientNet-B0 model architecture¹⁹¹

wise separable convolutions are a combination of a depthwise convolution and a pointwise convolution. The depthwise convolution is a convolution that is applied to each channel of the input independently, and the pointwise convolution is a convolution that is applied to the output of the depthwise convolution. This combination of convolutions allows the model to reduce the number of parameters and computations, whilst maintaining its accuracy. Figure 3.20 shows the depthwise separable convolution as a combination of a depthwise convolution and a pointwise convolution. The model also uses a combination of inverted residual connections and squeeze-and-excitation blocks.^{196,197} Residual connections are used to allow the model to learn the identity function, and squeeze-and-excitation blocks are used to allow the model to learn the importance of each channel in the feature map. In this case, the inverted residual connections go from narrow to wide to narrow convolution kernels ($1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$) as opposed to standard residuals. The model also utilises a combination of global average pooling (GAP)¹⁹⁸ and dropout layers. Global average pooling is used to reduce the dimensionality of the feature map, and dropout layers are used to regularise the model. A variant of the activation function ReLU was also used along with the swish activation function. This variant was ReLU6 which is defined as:

$$ReLU6(x) = \min(\max(0, x), 6) \quad (3.46)$$

This variant of the activation function was used to reduce the number of computations as it limits the range of the activation function to $[0, 6]$ and therefore any resultant float is limited

to 3 bits beyond the decimal. The swish activation function is defined as:

$$\text{swish}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1 + e^{-x}} \quad (3.47)$$

where $\sigma(x)$ is the sigmoid function. The swish activation function was shown to be more efficient than the standard ReLU activation function as it was shown to have a lower number of operations and a lower memory footprint. The model also utilises a combination of batch normalisation and stochastic depth. Batch normalisation is used to normalise the output of each layer across batches, and stochastic depth is used to regularise the model. The model also utilises a combination of label smoothing and mixup. They are both used to regularise the model and to reduce overfitting. Training the model involved a combination of weight decay and gradient clipping. Weight decay is used to regularise the model,⁹⁰ and gradient clipping is used to account for exploding gradients.

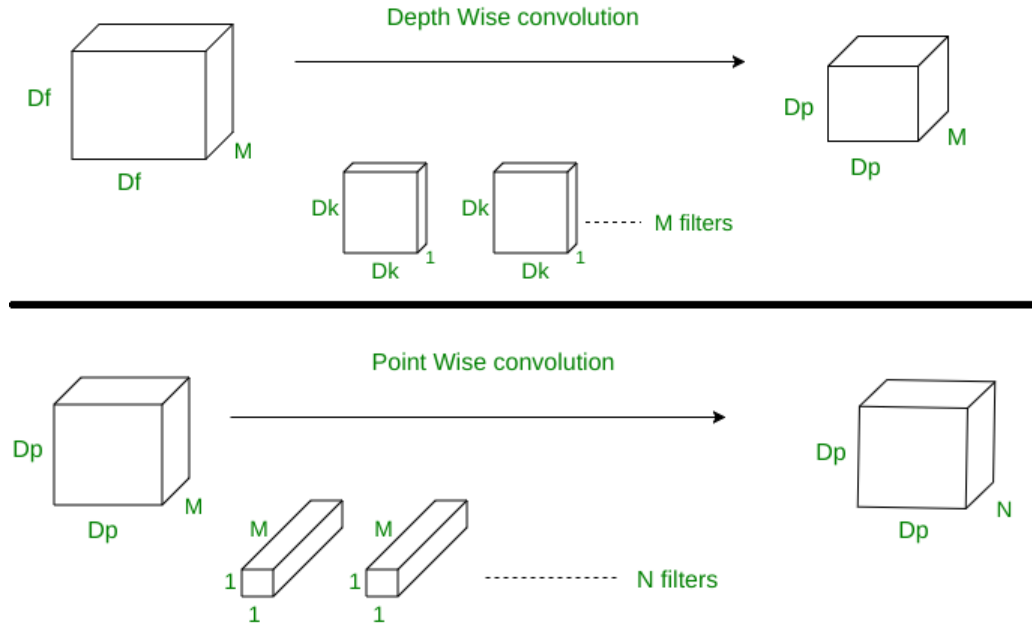


Figure 3.20: Depthwise and pointwise convolutions, together, form a depthwise separable convolution¹⁹⁹

The MBConv blocks are then stacked on top of each other, with the output of each block being the input to the next block. The output of the last block is then passed through a GAP layer, which averages the values of the feature maps across the spatial dimensions. The output of the GAP layer is then passed through a fully connected layer, which is then passed through a

softmax activation function. The softmax activation function is used to normalise the output of the fully connected layer, such that the output is a probability distribution.

3.5.2 Training Workflow

The pipeline is shown in Figure 3.18. The workflow is split into two parts, the pre-training and fine-tuning. The pre-training is done using synthetic data, and the fine-tuning is done using experimental data. The pre-training is done to give the ML model some context and background to the data, and the fine-tuning is done to fine-tune the ML model to better analyse experimental data.

The model weights were first initialised using the noisy-student EfficientNet-B0 pre-trained weights. The pre-training process was then started by training on the synthetic training set as split above. Each sample feature map was broadcast to three channels and fed through the model. A dropout of 0.1 was used on the classification affine layer for regularisation. Categorical cross-entropy loss was used with a batch size of 8. The Adam optimiser⁹³ was used to minimise the loss with a learning rate of 0.1. A linear learning rate decay was applied such that the learning rate was scaled by 0.1 per epoch with loss plateau.

The training and test loss was monitored during both pre-training and fine-tuning to determine the optimal number of epochs. All training was performed on an Nvidia GTX 1080 Ti GPU. Each pre-training epoch took approximately 5 minutes to train and around 50 seconds for each fine-tuning epoch. During pre-training on the synthetic set, the validation loss began fluctuating after 2 epochs, suggesting further training would lead to overfitting. As such, pre-training was halted after 2 epochs. The effects of longer/shorter pretraining on experimental SAXS lipid phase prediction is shown in table 3.3

For fine-tuning, the model was trained for 20 epochs, with snapshots taken in steps of 5 epochs for evaluation. The same learning rate decay was applied but this time a patience of 10 epochs was applied to the scheme, as the dataset is smaller and the number of trained epochs is larger.

1 Synthetic Pre-training Epoch		
Fine-tune Epochs	Valid/test accuracy (%)	Synthetic test accuracy (%)
0	54.9	98.3
5	95.5	97.7
10	92.4	94.8
15	96.9	98.8
20	96.9	96.3
2 Synthetic Pre-training Epochs		
Fine-tune Epochs	Valid/test accuracy (%)	Synthetic test accuracy (%)
0	62.5	99.6
5	96.9	97.8
10	98.2	98.0
15	92.4	99.2
20	99.6	99.5

Table 3.3: Model Results for predicting lipid phase behaviour on both the experimental and synthetic test sets for both 1 and 2 pre-training epochs over 0-20 fine-tuning epochs

From Table 3.3, it can be seen that pre-training for 2 epochs on the synthetic data, significantly improves the model’s downstream performance on predicting the lipid phases of the experimental samples ($\Delta + 3\%$). This confirms that the pre-training procedure and the synthetically generated SAXS patterns are accurate enough to give the model a better understanding of the features presented and their relationship with the lipid phase. It can also be seen that there is a general upwards trend in experimental test accuracy with an increase in fine-tune epochs, however this increase plateaus after around 10-15 epochs. This suggests that further fine-tuning may lead to overfitting (which can already be insinuated from the slight reduction in synthetic test set accuracy) and that the potential for larger fine-tune datasets may improve the model’s

predictability. Figure 3.21 shows the fluctuations of both the experimental and synthetic test set accuracies with the number of pre-train/fine-tune epochs.

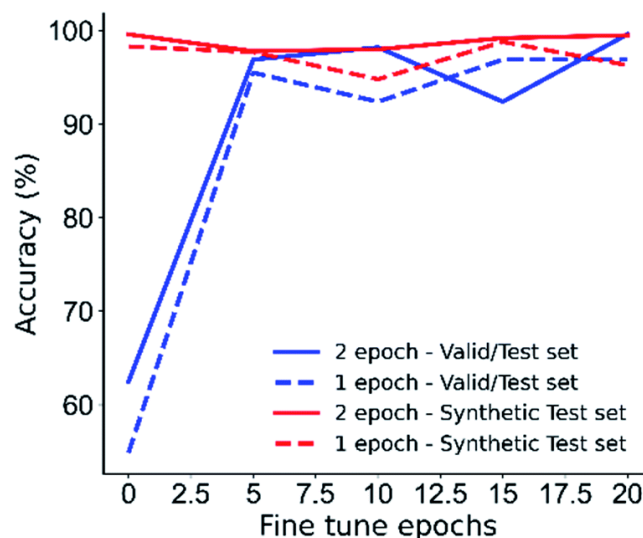


Figure 3.21: Accuracy trajectories with varying pre-training and fine-tuning epochs on both experimental and synthetic test sets¹⁹¹

Finally, another web app was built to allow experimental scientists to upload their raw SAXS spectra and the phase prediction probability distribution is displayed. Figure 3.22 shows a screenshot of the app.

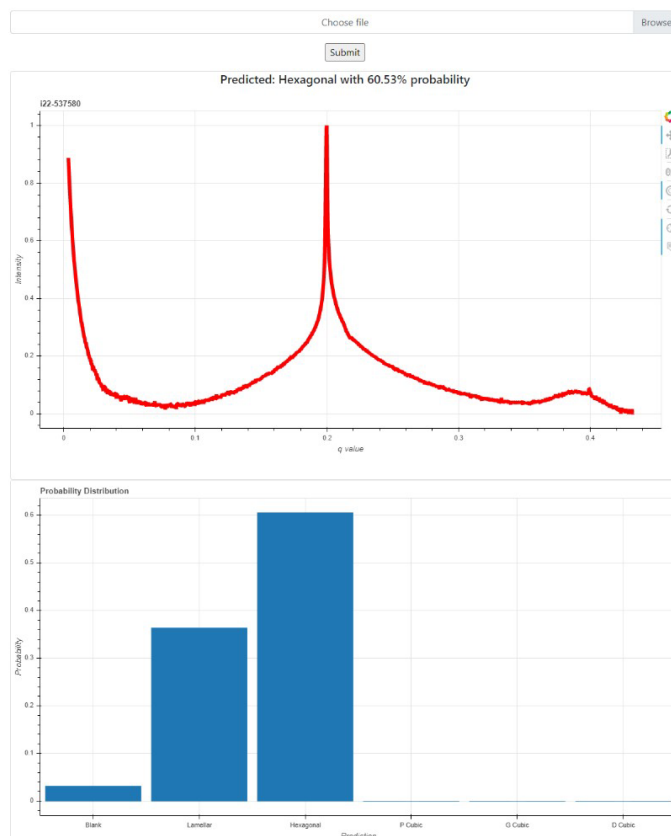


Figure 3.22: A screenshot of the model’s final web app that can be used by experimental scientists at the beamline facility

3.6 Conclusion

In this chapter, an entire ML pipeline was built to predict the lipid phase behaviour of an unknown sample from its experimental SAXS spectrum. An efficient web app was built to allow the easy data labelling of previously collected experimental spectra. A theoretical model of each phase was created and implemented to generate synthetic SAXS spectra of the lipid phases in order to augment the collected experimental data for training of a CNN (EfficientNet-B0) through a pre-training/fine-tuning scheme. Finally, another web app was built to allow for easy access to the model instantaneously.

Chapter 4

Distributed Pre-training of Transformers for Property Prediction

We have now explored both experimental and computed representations of aggregate phospholipids, which are structures on the 5-10 nm scale in thickness; and used ML to analyse both representations to accelerate experimental discovery.

In this chapter, a more fundamental approach to chemical representations will be explored using the Transformer architecture,⁵ a neural network architecture that has been shown to be very effective at learning representations of language, to generate learned representations of (small) molecules (which are usually of the 1 nm scale).^{6,37,200} A Transformer is pre-trained on a large corpus of text based chemical structures (SMILES), and then fine-tuned to predict specific properties of new molecules. This approach is then tested on multiple benchmark platforms (RDKit’s Benchmarking platform^{62,69} and MoleculeNet²⁹). The model’s latent space is then explored to better understand a more general chemical space. This approach is similar to the pre-training of language models and then using the pre-trained language models to perform downstream tasks such as machine translation and question answering.

Recent works within the field of chemistry and drug discovery have utilised the Transformer architecture to learn representations of molecules.^{6,37,200} Transfer learning, with pre-training

and fine-tuning has been shown to be effective for molecular property prediction and reaction prediction with models like ChemBERTa³⁷ and The Molecular Transformer.⁶ Further work was done to explore more abstract molecular representations with transformers for drug discovery with MolBERT.²⁰⁰ Other NLP techniques are also used in chemistry, for example, a common text-mining approach, topic modelling has been used to explore large molecular datasets and aggregate molecules within them.³⁹

In this work, the use of a Transformer architecture to learn representations of molecules is explored at scale, and these representations are used to predict molecular properties. The latent space of the model is explored and is benchmarked across a standard suite of datasets to better understand the chemical space that the model has learned. Data ablation studies are also performed to give insights into the learned features and biases.

4.1 Molecular Representations

Currently, in order to represent molecules for use in downstream tasks (such as property prediction), a variety of methods are used.^{6,201–205} These methods include the use of fingerprints (Morgan,^{72,110} RDKit,²⁰⁶ etc.), which are a vector representation of the presence of substructures in a molecule, and the use of (usually) graph neural networks that operate directly on the molecular graphs.^{202,204} These methods are effective, but they are not able to capture the full chemical space, and are not able to capture the chemical space in a way that is amenable to a variety of downstream tasks. In other words, different architectures and/or fingerprints must be used to achieve good downstream performance for each task. In this section, we will explore large scale pre-training of a Transformer architecture on a large corpus of SMILES. Feature vectors from the latent space of the encoder section of the model are extracted and used as molecular fingerprints for fine-tuning to predict different molecular properties agnostically.

4.2 Large Language Models and their Pre-training

One of the most prominent advances in the field of natural language processing (NLP) in the last few years has been the development of large language models (LLMs).^{2,55,81,82,207–211} These models are able to learn representations of language that are able to capture the syntactic and semantic structure of language. They have been shown to be effective at a variety of downstream tasks, such as machine translation, question answering, and text generation. These models are able to achieve state-of-the-art performance on these tasks, and are able to achieve this performance without any task-specific architecture or feature engineering. Some of these language models are Autoencoders (BERT,⁵⁵ ALBERT,²⁰⁷ RoBERTa⁸¹), some are Autoregressive (GPT-3,² GPT-J⁸²) and some are a combination of both (T5,²¹⁰ BART²⁰⁸).

Whilst the pre-training procedure for each of these language models varies significantly, the general procedure for the preparation of the training data is the same. The training data is a large corpus of text, which is then tokenized and converted into a sequence of tokens. A token, in this context is defined as a single unit of text. This can be a single character, a word, or a subword. These text units are represented as (mapped to) standard vectors (embeddings) which are read by the model. The tokens are then fed into the model, and the model is trained on a particular pre-training task. The model can then be fine-tuned on a downstream task. In many cases, the objective of the model is to generate meaningful embeddings for the tokens that better represent their sentiment, meaning, or context.

4.2.1 Pre-training Tasks

The pre-training tasks for the autoencoding language models are usually a masked language model (MLM) or a next sentence prediction (NSP) task. The MLM task is a task where a random subset of the tokens in the input sequence are masked, and the model is trained to predict the masked tokens. The NSP task is a task where the model is trained to predict whether two sentences are consecutive in a text. The MLM and NSP tasks are usually combined to

form a single pre-training task.

The pre-training tasks for the autoregressive language models are usually a causal language model (CLM) task. The CLM task is a task where the model is trained to predict the next token in a sequence of tokens. The CLM task is usually combined with the MLM task to form a single pre-training task.

A CLM task predicts the probability of the next token given the previous tokens in the sequence. This is mathematically defined as:

$$P(w_{t+1}|w_1, \dots, w_t) \quad (4.1)$$

Where w_t is the token at position t in the sequence of tokens.

A MLM task predicts the probability of a token given the entire sequence of tokens. The MLM task is a more complex task than the CLM task, and the MLM task is usually used as the pre-training task for the autoencoding language models. An example of a masked token is shown in Figure 4.1. This is mathematically defined as:

$$P(w_t|w_1, \dots, w_{t-1}, w_{t+1}, \dots, w_n) \quad (4.2)$$

Where w_t is the token at position t in the sequence of tokens.

This type of language modelling receives contextual information from the entire sequence of tokens, and is able to learn representations of language that are able to capture the syntactic and semantic structure of language. This is in contrast to the bag-of-words approach, which only receives contextual information from the immediate surrounding tokens, or from CLM tasks, which only receive contextual information from the previous tokens in the sequence. This modelling is therefore theoretically better able to capture the syntactic and semantic structure of language than the bag-of-words approach or the CLM approach. As it considers the structure of the language more holistically.

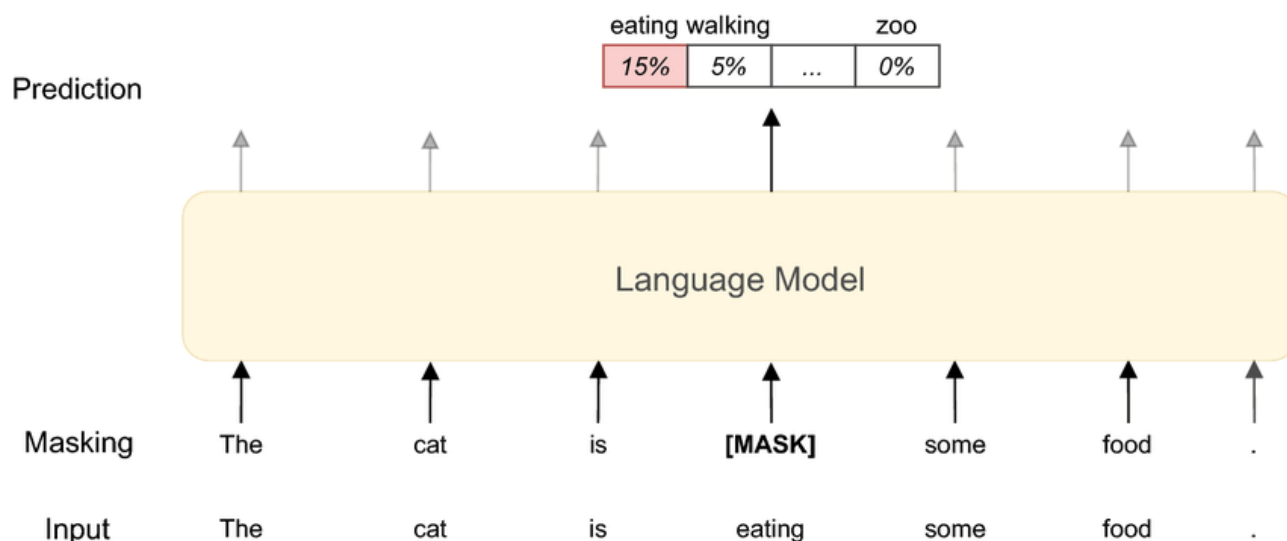


Figure 4.1: An example of a single masked language model (MLM) task. The model is being trained to predict the masked token given its context and vocabulary. Adapted from Weyssow et al.²¹²

Another benefit to this pre-training task is the ability to easily synthetically generate labelled data from an unlabelled corpus of text. This framing of an unsupervised learning task as a supervised learning task (known as self-supervised learning) is what enables these models to learn good representations from the available large unlabelled datasets.

It is important to note that before the MLM task is performed, the text must be tokenized. This is because the model is only able to learn representations of the tokens, and not the text itself. The tokenization process is the process of converting the text into a sequence of tokens. This can be as simple as separating the text on the spaces (into words) or through more elaborate tokenization methods.^{10,213} There are many considerations necessary to determine the best tokenization method for a particular language/space. These considerations include the size of the vocabulary, the size of the training data, the size of the model, and the downstream task. The tokenization process is discussed in more detail in Section 4.2.2.

4.2.2 Tokenization

There are three main classes of tokenization used today: word tokenization, character tokenization, and subword tokenization. The word tokenization algorithms are the simplest, and

are able to tokenize the text into a sequence of words. The character tokenization algorithms tokenize the text into a sequence of characters. The subword tokenization algorithms tokenize the text into a sequence of part words (subwords).

One of the most common set of tokenization algorithms today are the subword tokenization algorithms. These algorithms are able to learn a vocabulary of subwords from the training data. This is in contrast to the bag-of-words approach, which tokenizes the text into a sequence of words.

A subword is a sequence of characters that is a part of a word. For example, the word "tokenization" can be tokenized into the subwords "token", "##ize", "##ation". The "##" symbol is used to denote that the subword is a part of a word. This allows the generated vocabulary to be smaller than the vocabulary of words, and therefore allows the model to either be smaller, or to be able to process more tokens as the subwords can apply to multiple words. It also makes it easier for the model to gain a semantic understanding of the subwords as they are often used in multiple words. In this section we will discuss some of the most common subword tokenization algorithms, Byte Pair Encoding (BPE)²¹³ and Unigram.²¹⁴ We will also discuss the SentencePiece algorithm,¹⁰ which is a statistical method of creating an ML tokenization model.

Byte Pair Encoding (BPE)

Byte Pair Encoding (BPE) is a subword tokenization algorithm that is able to learn a vocabulary of subwords from the training corpus. The algorithm is as follows:

1. Pre-tokenize the training corpus into a sequence of words via a word tokenization algorithm.
2. Create a vocabulary of all the characters in the pre-tokenized corpus.
3. While the number of subwords in the vocabulary is less than the desired vocabulary size:
 - (a) Find the most frequent pair of consecutive characters in the training corpus.

- (b) Replace the most frequent pair of consecutive characters with a new subword of them merged together.

This can be mathematically defined as the union:

$$\text{Vocabulary} = \text{Vocabulary} \cup \text{Most Frequent Pair of Consecutive Characters} \quad (4.3)$$

It is important to note that vocabulary size is a hyperparameter and therefore the model designer chooses it. It becomes the stopping signal for BPE (or until no more merges are possible). It is also important to note that with tokenization comes auxiliary tokens such as: $\langle \text{unk} \rangle$, $\langle \text{pad} \rangle$, $\langle \text{cls} \rangle$, $\langle \text{sep} \rangle$, $\langle \text{mask} \rangle$, and $\langle \text{eos} \rangle$. These tokens are used to denote the beginning and end of a sequence, and to denote padding and unknown tokens. These tokens are added to the vocabulary of the model.

Unigram Language Model

The unigram language model is a statistical language model that is able to learn a vocabulary of subwords from the training corpus.

By definition, a unigram model considers each word (or subword) in the vocabulary to be independent of each other. This is in contrast to the n-gram model, which considers the previous $n - 1$ words to be dependent on the current word. This approximation is defined as:

$$p(\mathbf{x}|X) \approx \prod_{i=1}^n p(x_i) \quad (4.4)$$

where $\mathbf{x}$ is the sequence of words, X is the training corpus, x_i is the current word, and n is the length of the sequence. This unigram model is constrained by the normalisation constraint:

$$\sum_{x \in V} p(x) = 1 \quad (4.5)$$

where V is the vocabulary. This constraint is necessary to ensure that the probability distribution is valid.

SentencePiece

SentencePiece is a tokenization package written in C++.¹⁰ It allows fast tokenization of a large text corpus without the need for pre-tokenization. It is also language independent as it does not rely on whitespace for tokenization. By default, SentencePiece uses a Unigram language model in combination with the subword regularisation algorithm to learn a vocabulary of subwords from the training corpus.²¹⁴ Other algorithms such as BPE²¹³ are also implemented in the SentencePiece package.

The subword regularisation algorithm is an iteratively trained algorithm that works with a tokenization model to select the optimal subwords. It begins by initialising the base vocabulary. Unlike BPE, the base vocabulary starts large (usually all pre-tokenized words, characters and frequent substrings) and is then iteratively pruned. The algorithm defines a loss (like log-likelihood) and iteratively minimises it.

The objective of the subword regularisation algorithm is to find a way to assess the best subwords given the training corpus. This is done by defining a loss function that is able to measure the quality of the subwords.

Given a sequence of unigrams $w_1, \dots, w_n$, the probability distribution of the entire sequence is defined as:

$$P(w_1, \dots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \dots P(w_n|w_1, \dots, w_{n-1}) \quad (4.6)$$

therefore, considering an NMT task, given a pair of sequences of possible segmentations for a word (different subword combinations) X and Y , the probability of one sequence existing given the other and the model parameters, θ , is:

$$P(X|Y; \theta) = \prod_{i=1}^n P(y_i | \mathbf{x}, y_{<i}; \theta) \quad (4.7)$$

where $\mathbf{x}$ is the proposed sequence of subwords (the proposed segmentation) $y_{<i}$ is the sequence of subwords before the current subword y_i . This mathematical model is limited however, due to the exponential growth of the number of possible segmentations. This is because the number of possible segmentations is equal to the number of possible combinations of the subwords in the vocabulary.

To address this, expectation maximisation is used to find the optimal segmentation. This is embedded in the total cost of the model, which is the marginalised log-likelihood of the model. This is defined as:

$$\mathcal{L}_{\text{marginal}}(\theta) = \sum_{s=1}^{|D|} \mathbb{E}_{\substack{x \sim P(\mathbf{x}|X^{(s)}) \\ y \sim P(\mathbf{y}|Y^{(s)})}} [\log P(\mathbf{y}|\mathbf{x}; \theta)] \quad (4.8)$$

where D is the total number of all possible segmentations, $X^{(s)}$ is the set of possible subwords for the x_{th} sequence of characters, $\mathbb{E}$ is the expectation, and θ is the (unigram) model parameters. This is the objective function that is optimised by the subword regularisation algorithm.

In practice, due to the lack of computational feasibility given that the number of possible segmentations increases exponentially with sequence length, the expectation is approximated by sampling. This is done by sampling k sequences of subwords from the model and then taking the average of the log-likelihood of the model. This is defined as:

$$\mathcal{L}(\theta) \cong \frac{1}{k^2} \sum_{s=1}^{|D|} \sum_{i=1}^k \sum_{j=1}^k \log P(\mathbf{y}_j | \mathbf{x}_i; \theta) \quad (4.9)$$

$$y \sim P(\mathbf{y}|Y^{(s)}), x \sim P(\mathbf{x}|X^{(s)}) \quad (4.10)$$

where k is the number of samples taken from the model. For simplicity, SentencePiece uses

$k = 1$. In other words, the expectation is approximated by taking the log-likelihood of the model for a single random sample of the segmentation. This is therefore simplified to:

$$\mathcal{L}(\theta) \cong \frac{1}{|D|} \sum_{s=1}^{|D|} \log P(\mathbf{y}|\mathbf{x}; \theta) \quad (4.11)$$

This simplification works best with a greater number of iterations as it increases subword sampling. This is similar to Variational Autoencoders where expectation of the hidden state is approximated by one sample.

SentencePiece Training

SentencePiece is trained using the subword regularisation algorithm. Once we have a model to give the unigram probabilities and the corpus, we can train the model using the subword regularisation algorithm with the unigram model. Given a set of proposed unigram segmentation candidates, $S(x)$, the most probable segmentation $\mathbf{x}^*$ for an input X is given by:

$$\mathbf{x}^* = \arg \max_{\mathbf{x} \in S(X)} P(\mathbf{x}) \quad (4.12)$$

The marginal likelihood is which is again maximised by the EM algorithm is given by:

$$\mathcal{L} = \sum_{s=1}^{|D|} \log(P(X^{(s)})) = \sum_{s=1}^{|D|} \log\left(\sum_{\mathbf{x} \in S(X^{(s)})} P(\mathbf{x})\right) \quad (4.13)$$

The algorithm is trained using the following steps:

1. Initialise the vocabulary heuristically from the training corpus.
2. Repeat the following steps until convergence of the vocabulary to the desired size, or the maximum number of iterations is reached:

- (a) Using Expectation-Maximisation, optimise the unigram probabilities, $p(x)$, given the current vocabulary.
- (b) Sample each subword and compute its loss. The loss of each subword represents the probability of likelihood reduction on subword removal.
- (c) Sort the subwords by their loss, removing the bottom 20% of the subwords, excluding single character subwords. (i.e. minimise the loss)

Further details of the EM step and how it is used to find the most optimal subwords are given in Appendix A.1. Once EM is complete, the most optimal subwords are selected using a variant of the Viterbi algorithm. This is given in Appendix A.2.

4.3 Learning Representations of the Chemical Space

We have now considered a full NLP LLM training pipeline, from getting a training corpus, tokenizing the text, performing pre-training tasks with a transformer model, and finally fine-tuning the model on a downstream task. We now consider how we can apply this pipeline to learn representations of the chemical space. Figure 4.2 shows the pipeline used in this chapter.

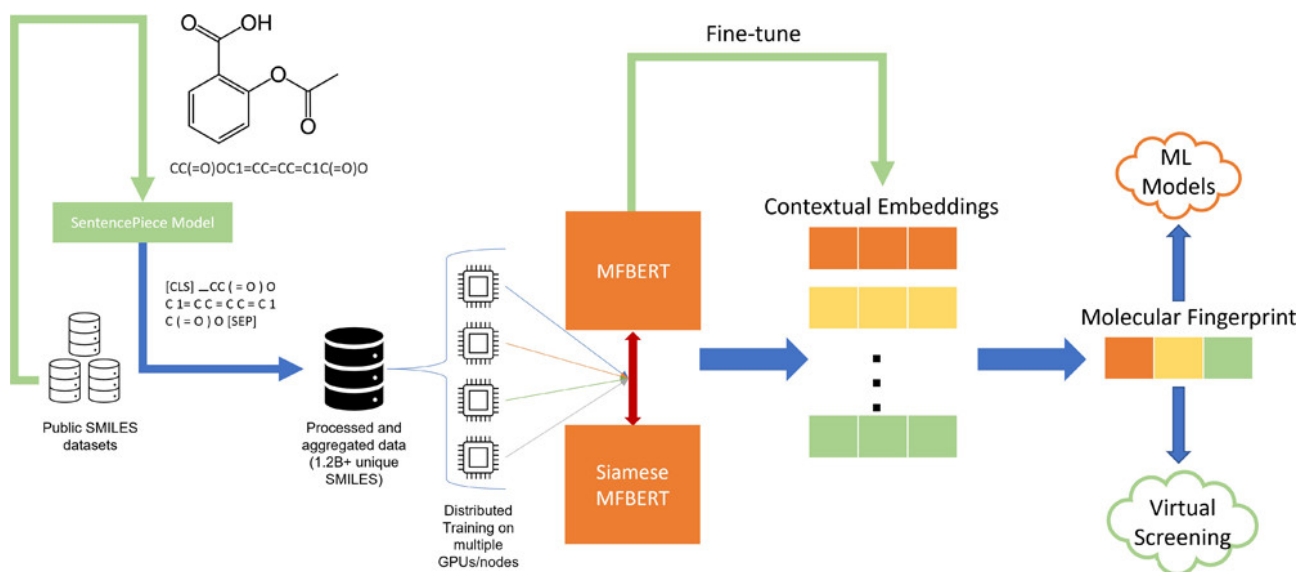


Figure 4.2: The workflow for learning representations of the chemical space²¹⁵

4.3.1 Data and Analysis

The main goal of this section is to be able to accurately and practically represent a wide variety of chemical compounds in a way that is able to capture their chemical properties, whilst maintaining the flexibility of being able to embed new chemical features, classes and representations. A wide variety of molecules that span across different chemical classes and features are therefore considered.

Data Aggregation and Preparation

In order to cover a variety of molecular features, many publicly available datasets were collected, filtered and aggregated. The datasets used were all large databases of SMILES⁶⁵ or SMARTS²¹⁶ strings, which were then filtered to remove any duplicates (more about this later), and any molecules that were not valid SMILES strings. The datasets used were:

- **ChEMBL**²¹⁷ - A database of bioactive molecules with known bioactivity or target. This database contains 1,000,000+ molecules.
- **GDB-13**²¹⁸ - A computationally enumerated database of 977M+ molecular graphs with constraints.
- **PubChem**²¹⁹ - A database of chemical compounds with associated metadata. This database contains 200,000,000+ molecules.
- **ZINC 15**²²⁰ - A database of commercially available compounds. This database contains 300,000,000+ molecules.
- **USPTO**²²¹ - A database of US patents with associated chemical reactions. This database contains 1,000,000+ molecules.

These datasets were chosen as they covered many areas of interest to chemists within the chemical space, whilst some of the larger, more broad datasets, like GDB-13,²¹⁸ were used more as a

means to enable the model to contextualize the space and be able to better abstract molecular structure limitations. The USPTO dataset²²¹ was also chosen for two main reasons. 1) It contains reaction SMARTS which are needed for the model to be flexible enough such that reaction tokens are in its vocabulary, so a user is able to fine-tune for a reaction related downstream task. 2) It contains reactions that are regularly used in industry, and the molecules used in these reactions are often repeatedly used for various products, thus, it allows an opportunity to accurately represent a frequency analysis of the practical chemical space which we can use to selectively bias our model. Table 4.1 shows the number of molecules in each raw dataset.

Dataset	Number of Molecules	Proportion of Aggregate (%)
GDB-13	977,468,301	81
ZINC 15	389,000,000	10
PubChem	206,550,222	9
ChEMBL	1,920,027	0.2
USPTO	994,838	0.00825

Table 4.1: The number of molecules in each raw dataset. The total number of molecules in the aggregate dataset is 1,575,933,388.

The datasets were then aggregated into a single dataset, which acted as the training corpus for the SentencePiece Tokenizer. For auxiliary tokens, [CLS] and [SEP] were used to denote the start (classifier token) and end (or separate for multi-SMILES input) respectively. The SentencePiece tokenizer was then trained on this dataset, and the resulting vocabulary was used to train the transformer model. The final vocabulary size was 2,417 tokens, which was chosen by the tokenizer once it reached the maximum number of EM sub-iterations which was set to 2 in agreement with the recommended default. The tokenizer was trained on the raw SMILES datasets prior to validation and filtration to ensure that all possible tokens were accounted for in the vocabulary such that out-of-vocabulary (OOV) errors are minimised for inference or downstream tasks. The OOV token used was [UNK].

Data Validation and Filtering

The datasets were then validated and firstly filtered to remove any molecules that were not valid SMILES strings. This validation was performed using the RDKit library,²⁰⁶ which is a cheminformatics library that is used to perform many chemical related tasks. Secondly, duplicates were removed. The validation and removal of duplicates was performed using the following steps:

1. Convert the SMILES string to a RDKit molecule object.
2. Check if the molecule object is valid and sanitize it.
3. If the molecule object is valid, traverse the molecular graph to re-generate the Canonical SMILES string.
4. Given the re-generated canonical SMILES, if:
 - (a) The Canonical SMILES string matches the original SMILES string, add it to the set of unique molecules.
 - (b) The Canonical SMILES string does not match the original SMILES string, assert that the original SMILES is a valid permutation of the Canonical SMILES string, then add the Canonical SMILES string to the set of unique molecules.

This first validation step removes any molecules with incorrect syntax, valencies, bonding, etc. The second validation step removes any duplicates, whilst also ensuring that the molecules are not just duplicates, but are in fact valid permutations of each other. This is important as it ensures that the model is not biased towards any particular SMILES string, and that the model is able to learn the chemical space in a more general way. This validation protocol was applied to the single molecule datasets (All bar USPTO). The reaction dataset was validated similarly with the RDKit SMARTS reaction parser. The molecules used within the reaction SMARTS were then extracted and validated as described above. The extraction process accounted for salts and other multi-molecule SMILES separated by a period. This was achieved through

a SMARTS substructure search through the reaction SMARTS, analysing molecules with a period given valency/charge constraints to assess whether they should be considered as a single molecule or not.

With duplicates and invalid molecules removed, the dataset now contained 1,264,754,823 molecules.

Data Analysis

As mentioned previously, whilst the chemical space is large, dense and varied, the actual molecules used practically in the real world only sparsely populate the chemical space. Thus, in order to effectively bias our model to the practical chemical space, an assessment of the frequency of molecules used in industry was required. Since the model was now required to place practically used molecules together (global closeness) in the latent space, it was also required to place functionally different molecules apart (local separation). As such, using SMARTS for a substructure search, the aggregate dataset is scanned for functional groups/fragments of molecules. The frequency of each available group in the dataset is assessed, and used to bias the model to place molecules with similar functional groups together locally in the latent space. Figure 4.3 shows the frequency of 85 molecular groups in a 1 million molecule random sample of the aggregate dataset, the 85 groups are considered according to RDKit’s fragment module.

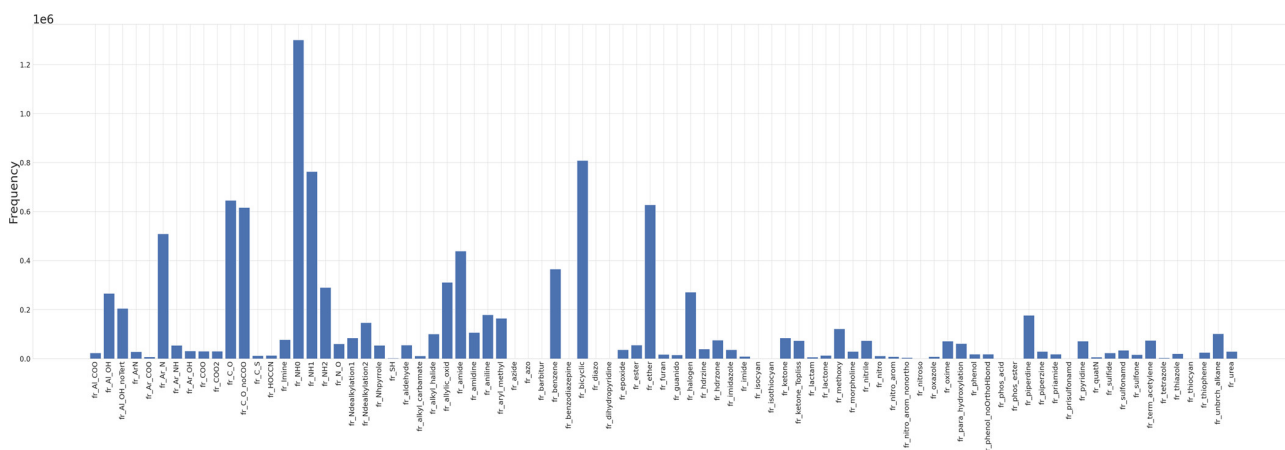


Figure 4.3: The frequency of 85 molecular groups in a 1 million molecule random sample of the aggregate dataset.²¹⁵

Assuming the 1M random sample is representative of the entire dataset, then the top most frequent groups are: *Tertiary amines*, *Secondary amines*, and *Bicyclic*, followed closely by *Car-*

bonyls and *Ethers*. This uneven distribution meant that the model would be biased towards some groups and would not be able to learn effective general representations of SMILES/S-MARTS during pre-training. To counter this, oversampling and augmentation techniques were used to a) normalise the frequency of each group, b) optimise the model’s learnings to more practical molecules by biasing the oversampling towards the frequencies observed in the USPTO reaction dataset and c) further increase the number of molecules in the total aggregate dataset.

Oversampling and Augmentation

The oversampling and augmentation techniques used were as follows:

1. For each group, the frequencies of the group in the aggregate dataset were computed.
2. The frequencies of the group were then considered within the USPTO dataset.
3. The ratio of the USPTO frequency to the aggregate frequency was calculated.
4. The number of molecules to oversample for each group was determined by multiplying the ratio by the number of molecules in the aggregate dataset.
5. This ratio is weighted such that there is a balance between oversampling the groups that are under-represented in the aggregate dataset, and oversampling the groups that are over-represented in the top 20% most frequent molecules the USPTO dataset. This is achieved by using the following formula: $ratio = \frac{1}{\frac{1}{USPTO-freq} + \frac{1}{agg-freq}}$. This formula ensures that the ratio is always greater than 1, and that the ratio is closer to 1 for groups that are under-represented in the aggregate dataset, and closer to the USPTO frequency for groups that are over-represented in the aggregate dataset.
6. Rather than oversampling with the canonical SMILES, the data was oversampled with permutations of the canonical SMILES. This was achieved using different graph traversals offered by the RDKit library.

Once the oversampling and augmentation was complete, the full, aggregate data was shuffled and binarized in order to set-up for model pre-training. The binarization process was performed using the following steps:

1. The full, aggregate dataset was shuffled.
2. The shuffled dataset was then split into 1000 chunks.
3. Each chunk was then split into 1000 sub-chunks.
4. Each sub-chunk was then binarized.
5. Each sub-chunk was then saved as a separate file.

The resultant binarized dataset was over 1.2 TB in size, and was used to pre-train our model.

Data Shuffling

In order to ensure model generalisability and robustness, the data was shuffled before being binarized. Shuffling reduces the likelihood that the model will get stuck in one area of the parameter space in which the optimiser is able to find a local minimum. This is achieved by ensuring that the model is exposed to a wide range of molecules during pre-training, and thus is able to learn a more general representation of the chemical space.

Because of the pre-training pipeline, GDB-13 molecules were shuffled separately from the rest of the dataset. The reason for this is discussed below. Because the augmented dataset was so large, it was not possible to shuffle the entire dataset at once, as it would not fit into memory. Therefore, an implementation of the Fisher-Yates algorithm²²² was used. This implementation was `terashuf`.²²³ It is provably fair shuffling and shuffles according to this algorithm:

1. N SMILES were divided into K chunks each containing N/K SMILES (and some left over).
2. Shuffle each of the K chunks.

3. Sample one of the K chunks where the probability of drawing chunk i is proportional to the number of SMILES in the chunk.
4. Pop the first line off the selected chunk and write it to the output file.
5. Repeat 3-4 until all SMILES have been written to the output file.

Data Binarization

The binarization process was performed to process and compress the dataset, such that it can be more easily accessible by the CPU and GPU. It also allowed more efficient distributed training where the master node is able to index the entire dataset and distribute the data to the workers without having to load the entire dataset into memory. This was achieved by memory mapping the dataset.²²⁴

4.3.2 Model

Once the data has been shuffled and binarized, the model and distributed training pipeline needed to be prepared. The model was based on the BERT architecture⁵⁵ and trained using the robust training methodology of the RoBERTa architecture.⁸¹ The model was trained using fairseq¹¹ and the PyTorch framework.¹² The model is named Molecular Fingerprints through Bidirectional Encoder Representations from Transformers (MFBERT)

MFBERT Architecture

MFBERT is based on the RoBERTa architecture⁸¹ and is shown in Figure 4.4. The Model is a 12-layer, 768-hidden, 12-heads, 88M parameter model. The input length is limited to 514 tokens (512 excluding auxiliary tokens) due to memory and parameter limitations. This is identical to the encoder part of the transformer architecture given in figure 2.10 but at a different scale.

During pre-training the base architecture was used. MFBERT was then duplicated and through weight sharing, used to create both variants of the Siamese MFBERT models. These Siamese

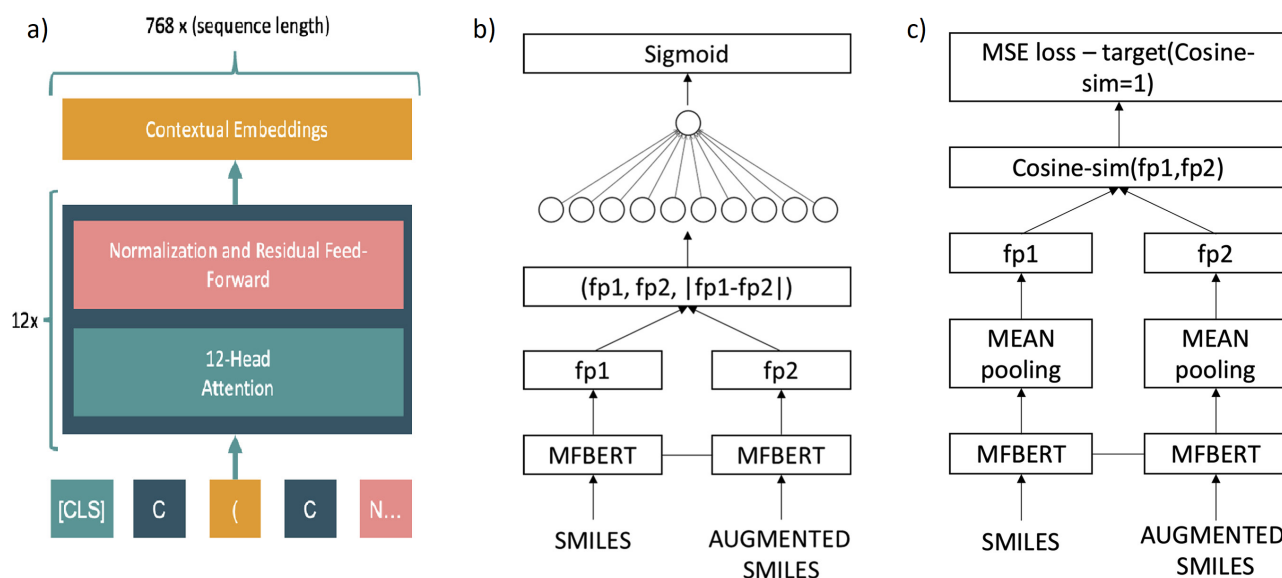


Figure 4.4: Three MFBERT architectures. a) is the base model, b) is a Siamese classifier and c) is the Siamese regressor, adapted from own work²¹⁵

models, combined with SMILES augmentation through graph traversal permutations were trained on downstream tasks to assess their performance as well as the impact of the SMILES augmentation on learning. This test-time augmentation meant that the model could take two SMILES variants of the same molecule and either classify them according to the downstream task, or regress the similarity between them. Future work could investigate potential reaction spaces by feeding either the single or Siamese models reactants and products instead. This would be possible due to the [SEP] token which allows multi-molecule input in the base model.

Once the SMILES are passed through during pre-training, fingerprints for the molecule are extracted, and used. In the Siamese models, these fingerprints and their distance in the latent space are used to produce the final output. In the regressor model, cosine similarity is used to calculate the distance between the two fingerprints. In the classifier model, the distance used is the absolute value of the difference between the two fingerprints.

Distributed Pre-training

MLM was used for pre-training across all datasets with 15% token masking probability. This task is a standard task in pre-training LLMs and is the first step in the training process. Cross-

entropy loss was used to validate the predictions of the masked tokens with the ground truth. The MLM task was performed on the entire dataset, however the order of the dataset used was chosen deterministically.

In general, MFBERT’s pre-training hyperparameters closely followed those of RoBERTa’s,⁸¹ with the exception of the batch size which was modified to better suit the distributed training setup. In line with general MLM LLM pre-training in literature, a polynomial decay learning rate schedule was used. The learning rate (LR) was set to 0.0001 and decayed by 0.0001 every 100,000 steps. Warmup steps were also used within the LR up to a peak LR of (0.0006) This peak learning rate is linked to a) the batch size and b) the total dataset size and number of steps. Whilst the peak LR used is different from RoBERTa’s, it is proportionally scaled when considering its multi-variate dependence. To scale the batch size to an even larger value (which is desirable as it has been shown that large batch sizes not only train faster, but also potentially improve downstream performance), gradient accumulation was used over the distributed network of GPUs. In order to optimise the architecture and optimiser hyperparameters, 32 MFBERT models were pre-trained on a subset of the data to find the optimal parameters. The hyperparameters found matched those of RoBERTa’s.

Table 4.2 shows MFBERT’s configuration and the hyperparameters used for pre-training.

The pre-training steps were performed in a distributed manner across 4 separate nodes, each with 4 Nvidia GTX 1080Ti GPUs (11 GB VRAM). The master node was responsible for loading the dataset and distributing it to the workers. The workers were responsible for training the model and reporting the loss back to the master node. The master node was also responsible for saving the model checkpoints. Each node contained a copy of the entire dataset such that only the dataset index needed to be communicated over the network. This allowed for a more efficient distributed training setup. Distributed training was performed using NCCL primitives,²²⁵ Fairseq 0.9.1¹¹ and Pytorch 1.7.1.¹² Gradient optimisation was performed using the AdamW optimiser,^{90,93} which was parallelised over the nodes/GPUs by using the all-reduce primitive.

Hyperparameter	Value
Batch Size/GPU	8
Learning Rate	0.0001
Peak Learning Rate	0.0006
Warmup Steps	10000
Total Epochs	1
Gradient Accumulation Steps	32
Number of GPUs	16
Number of Workers	16
Number of Nodes	4
Hidden Size	768
Number of Layers	12
Number of Heads	12
Intermediate Size	3072

Table 4.2: MFBERT’s distributed pre-training hyperparameters

All-reduce is a collective communication primitive that is used to aggregate the gradients from all workers and then broadcast the result back to all workers. This is a common operation in distributed training and is used to synchronise the gradients across the workers. The all-reduce operation is performed using the NCCL library²²⁵ which is a high performance collective communication library. The all-reduce operation is performed using a tree-based reduction algorithm which is a common approach to all-reduce. The tree-based reduction algorithm is a recursive algorithm that reduces the number of communication steps required to perform the all-reduce operation. The tree-based reduction algorithm is shown in Figure 4.5.

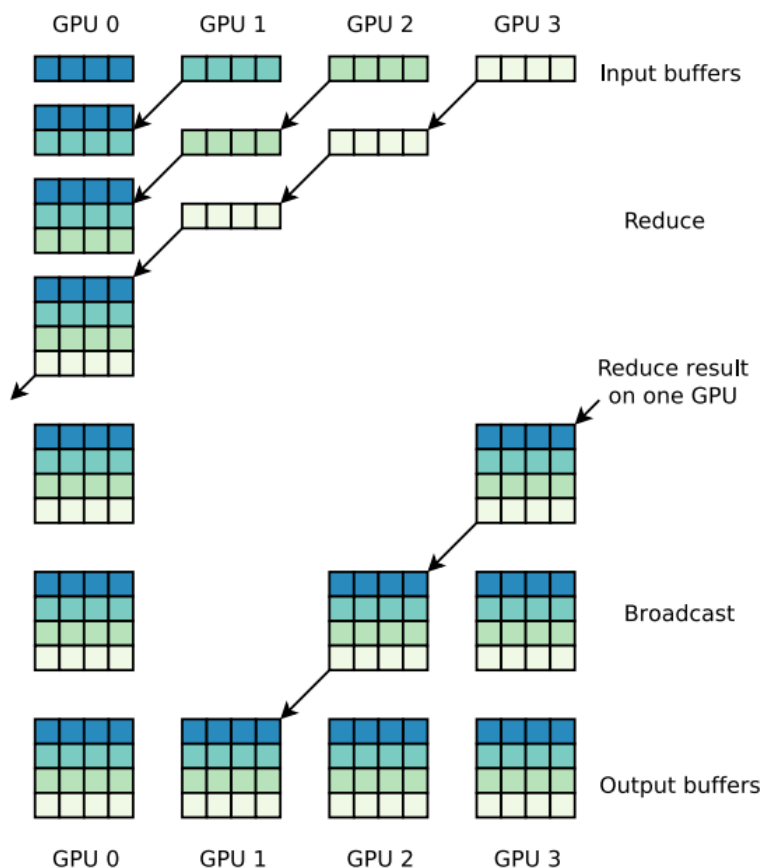


Figure 4.5: Tree-based reduction algorithm such as the one used in NCCL, Figure adapted from Carl Yang’s work²²⁶

Pre-training and Evaluation Pipeline

Xavier Uniform²²⁷ was used to initialise the weights of the model to ensure that they were initialised in a way that was appropriate for the activation functions used. After initialisation, the model was first fed the shuffled GDB-13 dataset.²¹⁸ After this initial pre-training of 1 epoch on GDB-13, the model was then pre-trained on the remaining datasets in aggregation for 1 epoch. It is important to note that in this context, 1 epoch refers to one passthrough over the complete dataset in question. This was the minimal amount the model could be trained on that guaranteed that the model has seen every molecule at least once, independently of any downstream tasks. Further pre-training may potentially improve downstream performance, but the ROI on further CPU/GPU time will be minimal due to the law of diminishing returns. Pre-training for 1 epoch across the whole dataset took approximately 1 month across all nodes. After this pre-training step, the weights were transformed for use with the Huggingface Transformers

library²²⁸ for easier manipulation and fine-tuning. The weights were then saved to disk and used for downstream tasks.

Fine-tuning and Evaluation Pipeline

For Fine-tuning, the pre-trained model was ablated and the MLM head was removed. This allowed access to the latent space of the model, where each token in the output sequence is a contextual 768 vector (contextual embeddings). These contextual embeddings were averaged across all tokens in the output sequence and that resulted in our 768 dimensional base molecular fingerprint. This constructed the base fine-tuning architecture which was further modified for each downstream task.

For RDKit’s benchmarking platform - the virtual screening (RDKit VS) benchmark,^{62,69} there was no training data for further fine-tuning, thus, this was purely an evaluation as to how the model performs in abstracting the chemical space from SMILES MLM pre-training with no knowledge of a specific task. MFBERT was further trained using MLM on the pre-training dataset for 0.5 epochs this was the model used for the RDKit VS benchmark.

For the other downstream tasks, an affine layer was added after the molecular fingerprints with a 20% dropout. This, combined with either a sigmoid or softmax activation and categorical/binary cross-entropy or MSE loss, allowed for the model to be fine-tuned on the regression and classification tasks. For fine-tuning, each training set was passed through for 10 epochs and the Adam optimiser⁹³ was used. The fine-tuning hyperparameters are shown in Table 4.3.

For the Siamese Models, the pre-trained checkpoint was also used as the base model (as opposed to the fine-tuned single model of the particular task). This was in an attempt to reduce model bias where the augmented Siamese model would be stuck in the optimal point found by Adam with the single model, as the weights are shared a simple scaling would land in the same area in the parameter/loss space. The Siamese models were fine-tuned with the same hyperparameters as the single models, but with a train batch size of 12. This was due to the fact that the Siamese models were trained on pairs of molecules and therefore more GPU memory was required for

Hyperparameter	Value
Train Batch Size	16
Valid Batch Size	8
Learning Rate	1×10^{-5}
Warmup Steps	0
Total Epochs	10
Dropout	0.2

Table 4.3: MFBERT’s fine-tuning hyperparameters

each sample (although not double since the weights are shared and the gradients across the model pair are accumulated).

For the classification Siamese architecture (Figure 4.4 b)), The classification head had weights of dimensionality $W_c \in \mathbb{R}^{3n \times l}$, where n is the MFBERT Fingerprint dimensionality (768) and l is the number of classes. With an activation function, σ , the output of the classification head is given by:

$$\sigma(W_c(fp1, fp2, |fp2 - fp1|)) \quad (4.14)$$

For the Regression Siamese architecture (Figure 4.4 c)), there was no head and the fingerprints’ cosine distance is computed directly in the latent space. MSE is then used as the loss function for the model, with the ground truth being a cosine similarity of 1, given that the two molecules are identical, but only permutations. The cosine distance is given by:

$$\frac{fp1 \cdot fp2}{||fp1|| \cdot ||fp2||} \quad (4.15)$$

This was computed using sklearn’s cosine similarity function.¹⁴

This shows that the model is somewhat able to abstract the functional groups of the molecule and their importance in the binding site. An attempt to quantify these observations is made. A Mann-Whitney U test was performed on the attention between the functional group token distribution and the attention weight distribution. The functional group token distribution was computed by summing the attention of all tokens that were mapped to a functional group node. The attention weight distribution was computed by summing the attention of all tokens in the molecule. The resultant p-value for the test was < 0.01 , which is statistically significant, meaning that the distributions do not overlap. This shows that whilst attention of the model is higher on functional groups for some layers/heads, overall, attention weights mainly carry abstract, non-interpretable features, so whilst it may be alluring to read into these weights, it is not advisable to do so. Instead it is best to consider the latent space after fingerprint generation and inference optimisation. This will be discussed in detail in the remainder of this section.

Inference and Fingerprint Pooling

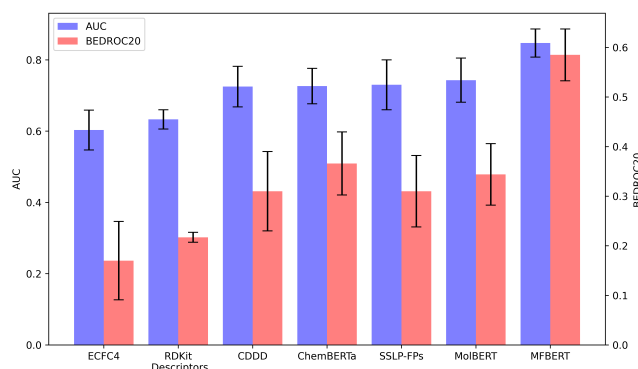
As MFBERT is a Transformer encoder,^{5,55,81} it generates latent representations in the form of contextual embeddings on token level resolution. Therefore, for each molecule, a varying length matrix would be generated, depending on the number of tokens the SMILES has been deconstructed to by SentencePiece,¹⁰ with each token’s representation being a 768-dim float32 vector. In order to use the contextual information for downstream tasks, the embeddings must be pooled into a fixed length vector. In NLP, this is achieved by only considering the first token’s embedding, which is the [CLS] token.⁵⁵ The reason models in NLP are trained in this way is because in language and dialect, a significant proportion of tokens contain no semantic information, and are redundant. Examples of redundant tokens include stop words, (some) punctuation, and filler words. This is not the case for molecules, where every token is equally important since SMILES is a lossless representation of a molecular graph. This hypothesis is assessed by considering the performance of MFBERT on RDKit VS and downstream tasks with both the [CLS] token’s embedding as the molecular fingerprint and the mean embedding across all tokens, to pool the contextual information.

RDKit Benchmarking platform

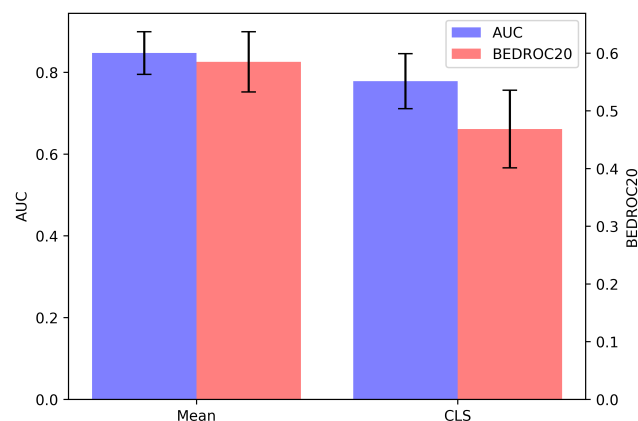
To assess the pre-trained MFBERT fingerprints, the filtered subset of RDKit’s virtual screening benchmark (version 1.2) was used. This subset includes hard-to-separate bioactive molecule (actives) and respective decoys.^{62,69} 69 protein targets were used for this dataset, the full list of targets (and their MFBERT scores) can be found in Appendix A.3. The task was to assess the use of MFBERT fingerprints in separating actives from decoys, given 5 query molecules ($n = 5$). Separation is measured by cosine distance in the fingerprints’ latent space. The metrics used for this task were 1) Area Under the ROC Curve (AUC) and 2) Boltzmann-enhanced discrimination of ROC (BEDROC)²²⁹ with $\alpha = 20$. The BEDROC metric is a more discriminatory variant of the AUC metric, which weighs the top $\alpha\%$ of the ranked retrievals higher than the rest. It is skewed according to the Boltzmann distribution and is important in virtual screening (VS) as it accounts for the early recognition problem. The early recognition problem is almost exclusive to VS and *in silico* drug discovery. Given that very few of the predicted actives within a molecule library (which are often huge) will actually be tested experimentally, the model must do well and rank the active molecules highly in the predicted set of the scored compounds. Therefore, even if a model is able to retrieve all the actives, it is not a good model if it does not rank them highly and its early performance is therefore bad. The BEDROC metric is a more appropriate metric for this task, as it accounts for the early recognition problem. Some initial solutions to the early recognition problem metric include the robust initial enhancement (RIE)^{230,231} and the enrichment factor (EF).²³² The BEDROC metric is a more generalised combination of these metrics, and is therefore more suitable for a VS task. The BEDROC metric is given by:

$$BEDROC = \frac{e^{\alpha R_a} - R_i}{e^{\alpha R_a} - 1} - \frac{R_i}{1 - e^{-\alpha R_i}} \quad (4.16)$$

Where R_a is the ratio of actives $R_a = \frac{n}{N}$, R_i is the ratio of inactives $R_i = \frac{N-n}{N}$, and N is the total number of compounds.



(a) Performance comparisons between MFBERT and other recent top performing models on the RD-Kit VS benchmark



(b) Comparison between [CLS] and mean pooling strategies for MFBERT

Figure 4.7: RDKit’s benchmarking platform results

Figure 4.7 shows a) the results of MFBERT fingerprints on the filtered set of the RDKit VS benchmark compared to other state-of-the-art fingerprints for this benchmark. b) MFBERT’s ROCAUC scores when using both the [CLS] token and pooling at inference as the fingerprint and the BEDROC scores when using both the [CLS] token and pooling at inference as the fingerprint.

Considering just MFBERT (4.7b), both the AUCROC and BEDROC scores are significantly higher with our mean pooling inference strategy for the virtual screening task with pre-trained MFBERT fingerprints. This confirms the hypothesis that mean contextual embedding pooling is more suitable than [CLS] pooling. The [CLS] token is maintained in MFBERT for flexibility in downstream tasks, but it is recommended to use the mean pooling strategy for virtual screening tasks.

Considering figure 4.7a, MFBERT’s performance is comparable to and improves on other state-of-the-art fingerprint methods, such as ECFP/ECFC. This shows that MFBERT is able to learn a latent space that is suitable for virtual screening tasks without an explicit task objective. The other fingerprints compared consist of two classical fingerprinting methods (ECFC4 and RDKit descriptors) and four other ML based methods. It is interesting to see how all ML based approaches significantly outperform the classical fingerprinting methods. This is likely due to the fact that classical fingerprinting methods are not able to capture the contextual information of a molecule, and therefore cannot learn a latent space that is general enough for virtual screening. The algorithms used for comparison, and their approach to fingerprinting are as follows:

- **ECFC4**⁶⁰ - Circular fingerprinting with radius 4, thoroughly discussed in section 2.3 - was performed using the RDKit implementation.
- **RDKit descriptors**²⁰⁶ - A set of 2000 descriptors based on a molecule’s physiochemical properties, implemented in RDKit.
- **Continuous and Data-Driven Descriptors (CDDD)**²⁰³ - An encoder-decoder model pair that learns a latent space representation of a molecule. The encoder is a convolutional network (CNN) and the decoder is a recurrent net (RNN). The latent space representation between the two is then used as the fingerprint.
- **ChemBERTa**³⁷ - A RoBERTa based model for molecular property prediction through transfer learning, it is also pre-trained on a large corpus of chemical data.
- **Self-Supervised Learning Platform for Molecular Fingerprints (SSL-FP)**²³³ - Also a BERT based model that is pre-trained on hundreds of millions of molecules.
- **MolBERT**²⁰⁰ - A BERT based model that considers multiple layers of the encoder’s latent space for fingerprint generation. It is trained on just over 1.5 million molecules and is the second best model for this benchmark after MFBERT.

The improvements in performance of MFBERT over the other ML based methods ($\Delta + 15\%$ in AUCROC and $\Delta + 70\%$ in BEDROC20) is likely due to the fact that MFBERT is pre-trained on a much larger corpus of data than the other ML based models, and is therefore able to learn a more general latent space representation. The mean pooling strategy also significantly improves MFBERT’s performance in this task.

The Siamese MFBERT model was also further trained for this benchmark. A 1 million SMILES sample from the aggregate dataset was used for the further training. The strategy was based on the regressive Siamese model, where the loss was MSE between cosine similarity of a molecule and its augmentation and 1. The model was trained for 1 epoch with a batch size of 12. The model was also ensembled (mean ensembling) 6 times. The results were significantly worse with the best model achieving AUCROC 0.554 ± 0.016 and BEDROC20 0.165 ± 0.015 . This is likely due to the Siamese model getting stuck in the parameter space where it is more optimal in similarly placing permuted SMILES, rather than becoming optimised to learn a general latent space representation of the chemical space. This could also be due to the fact that the Siamese model is not trained on a large enough corpus of data. Future work could train duplicate Siamese models on double the data to see if the results improve.

GDB-13 Exploration

Given that GDB-13 was the largest subset of the aggregate dataset and as it was computationally enumerated, it could therefore be split into further cumulative subsets by molecular features. Blum et al.¹¹³ have created these cumulative subsets whereby each subset gets smaller and contains molecules with cumulative property removals. Table 4.4 shows the cumulative subsets of GDB-13 and their sizes.

Cumulative Subset	Size (N molecules)	Filter Criteria
A	977,468,301	The original GDB-13 dataset
AB	635,647,478	no cyclic or acyclic HetHet bonds
ABC	441,084,370	stable FG*
ABCD	277,628,675	no cyclic $C = C$ or $C \equiv C$ bonds
ABCDE	140,606,518	no acyclic $C = C$ or $C \equiv C$ bonds
ABCDEF	43,729,989	no small rings
ABCDEFG	12,899,741	fragment-like
ABCDEFGH	1,470,284	scaffold-like

*stable functional groups exclude aldehydes, esters, carbonates, sulfates, epoxides, aziridines

Table 4.4: Cumulative subsets of GDB-13

In order to ascertain the impact of the removal/addition of certain molecular features on the performance of MFBERT on the RDKit VS benchmark, MFBERT was trained on each of the cumulative subsets of GDB-13 only. However, this also means that the size of the pre-training dataset for the smallest subset will be significantly smaller than that of the larger subsets. To account for this, each of the subsets were upscaled using our SMILES augmentation strategy such that all subsets were the same size (635,647,478 molecules). We already know that SMILES permutation augmentation has minimal effect on the model’s learned representations,²⁰⁰ therefore only SMILES permutation augmentation was used for the upsampling, and the only independent variable in this study was the features of the molecules in each subset. Figure 4.8 shows the AUCROC scores for each of the 7 models/subsets.

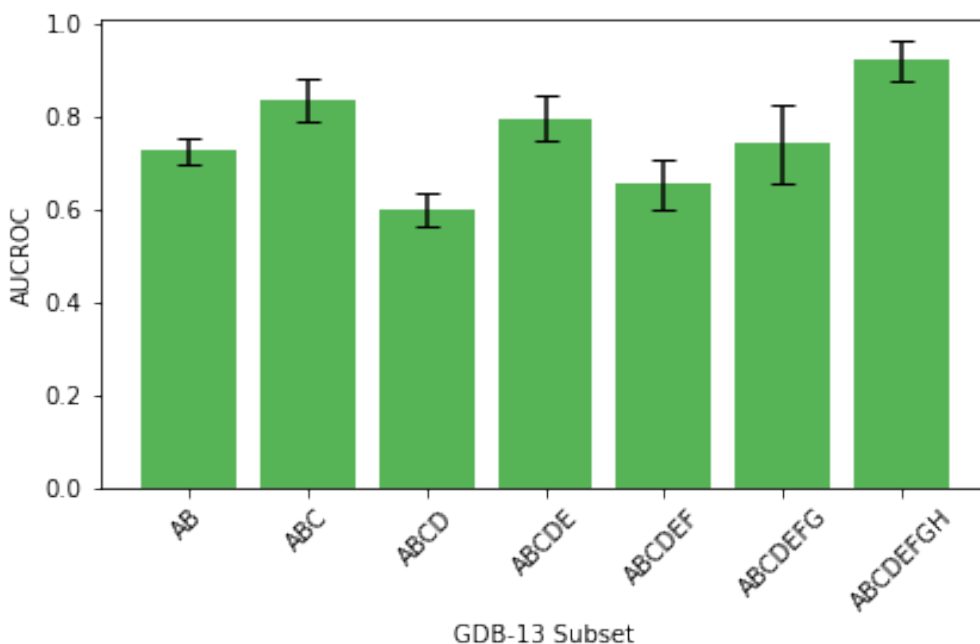


Figure 4.8: AUCROC scores for each of the 7 models/subsets. The AUCROC for pre-training on the whole of GDB-13 (i.e. subset A) was 0.67 - in-line with other subsets and language models assessed in this study.

The AUCROC fluctuates across all subsets with no clear pattern with the exception of ABCDEF, ABCDEFG and ABCDEFGH where there is a clear positive correlation with statistically significant improvements on the performance on the virtual screening task between the latter two sets. This is likely due to the fact that within the smaller subset, the focus was on molecules with similar features to the benchmark molecules. However the performance of the model on the benchmark set is not necessarily indicative of the model’s performance in the real world. Not only that, but these similar molecules were also present in the larger subsets, some of which performed significantly worse. The difference could be due to the fact that the similar molecules were repeated more when the smaller datasets were upscaled, this suggests overfitting however further analysis on the overlap between the two subsets and the benchmark suggest that this is not the case. The overlap between the benchmark molecules and ABCDEF in terms of mean tanimoto similarity is 0.055 ± 0.031 , where as the overlap with the ABCDEFG subset is 0.043 ± 0.028 , and the overlap with the AB and ABCD subsets is 0.038 ± 0.022 and 0.035 ± 0.025 respectively. This suggests that the model is not overfitting to the smaller subsets, but rather that the smaller subsets are more representative of the benchmark molecules. It is

also important to note the other models in literature and their performance on the benchmark. ChemBERTa achieved an AUC score of 0.73, and MolBERT achieved a score of 0.76, even though MolBERT was trained on a significantly smaller dataset. This suggests that whilst dataset size may be important, there comes a point where additional training data does not enable the model to abstract generalised molecular features. This is where data quality becomes important, and the cumulative subsets of GDB-13 are a good representation of how even smaller (in variety), but better quality subsets more suited to the task at hand can outperform larger, but less quality datasets. This explains why the smallest subset, ABCDEFGH performed the best, as it contains only scaffold-like molecules therefore gives a general representation of a variety drug-like molecules, which makes it suitable for a virtual screening task. The fact that larger datasets does not necessarily mean better performance is also supported by Chan et al.’s work on SSLP-FPs.²³³

Pre-trained vs Fine-tuned Fingerprints

Throughout the RDKit VS benchmark, MFBERT was only pre-trained with the MLM task. Therefore the model had no understanding of the objective of the task, and was only trained to learn a general representation of the chemical space. The contextual representations learned by the model could therefore be used agnostically across many tasks.

To assess this, the performance of the pre-trained MFBERT fingerprints with the fine-tuned MFBERT fingerprints were compared on three classification datasets/tasks from MoleculeNet.²⁹ These datasets were:

- **BBBP**²³⁴ - A dataset of 2036 organic compounds with a label as to whether they are active as a blood-brain barrier permeable molecule.
- **HIV**²³⁵ - A dataset of 4110 compounds with a label as to whether they have experimentally been able to inhibit HIV replication.
- **ClinTox**^{236,237} - A dataset where FDA approved/disapproved drugs were tested for their toxicity in a cell line. The task is to predict a binary label of the toxicity of a molecule.

For each classification dataset (BBBP, HIV, ClinTox), a random 80/20 training/validation split was done for fine-tuning and the model trained for 10 epochs. The mean pooling strategy was used for fine-tuning. The fingerprints in the pre-trained model’s latent space are then compared with the fingerprints in the fine-tuned model’s latent space across the entire dataset. The latent spaces are visualised using t-SNE²³⁸ and are shown in figure 4.9.

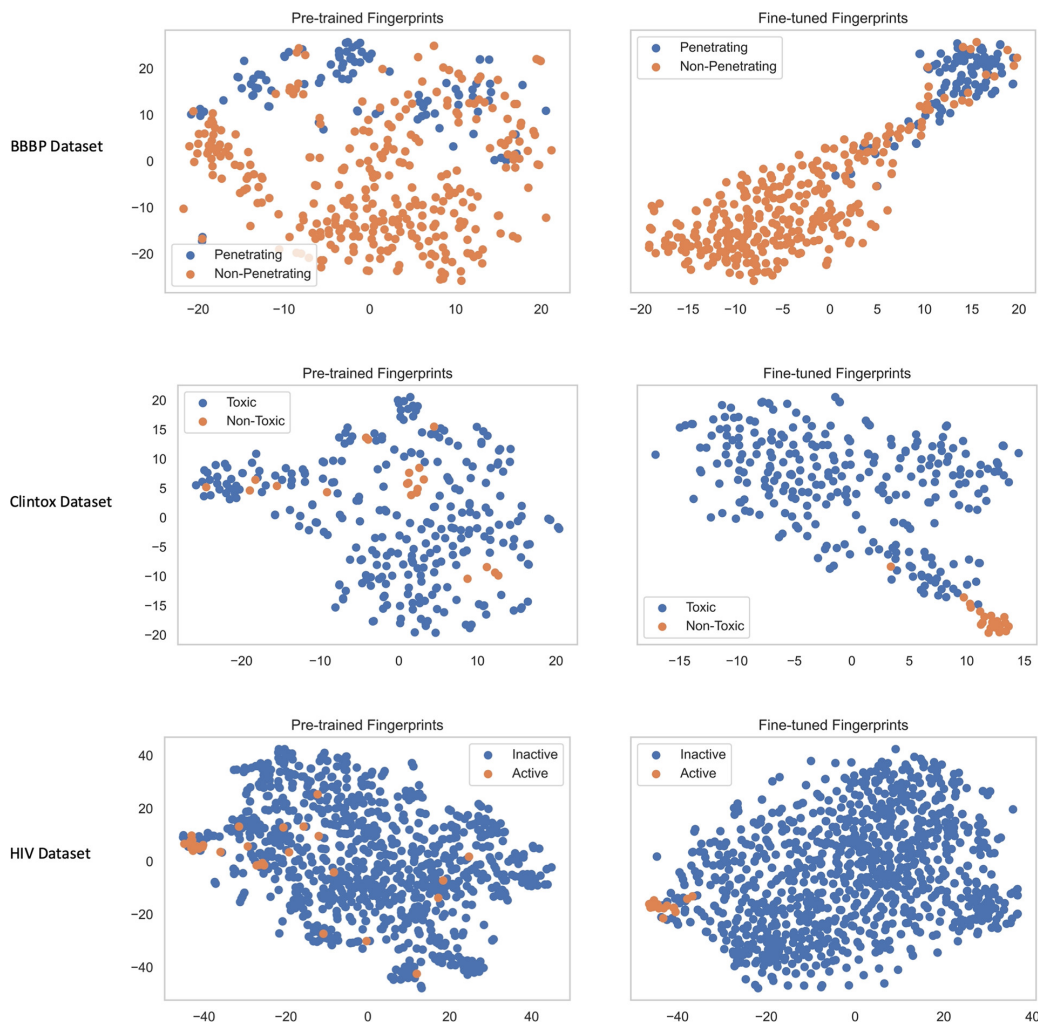


Figure 4.9: t-SNE visualisation of the latent space of the pre-trained and fine-tuned models.²¹⁵

The t-SNE visualisations show that the fine-tuned fingerprints seem to have significantly more distinct class separation than the pre-trained fingerprints for each of the three datasets; which is expected given that MFBERT was fine-tuned for each of the particular tasks. However what is interesting is that the pre-trained fingerprints also seem to have some (albeit less prominent) class separation, which suggests that the pre-trained model has learnt some generalised features that are applicable for these classification tasks. Considering the pre-trained fingerprints on the BBBP²³⁴ dataset, we can see the separation with the non-penetrating molecules mostly

Dataset	Pre-train Fingerprints		Fine-tune Fingerprints	
	AUCROC	Silhouette	AUCROC	Silhouette
BBBP	0.589	0.416	0.741	0.671
ClinTox	0.533	0.471	0.814	0.722
HIV	0.529	0.364	0.744	0.679

Table 4.5: K-means performance on the pre-trained and fine-tuned models on the three classification datasets after PCA for dimensionality reduction

occupying the bottom of the plot, and the penetrating molecules occupying the top. This separation is less consistent with the ClinTox^{236,237} dataset where there is no clear class separation visible with the pre-trained fingerprints. With the HIV dataset,²³⁵ some density based clusters can be seen, however the separation is not as clear as in BBBP. It is also interesting to see that for the fine-tuned HIV fingerprints the clusters are also separate on density, however the separation is much more clear. For the BBBP dataset, one of the features that the model could have abstracted during pretraining that would be applicable to this task is the size of a molecule and the steric hindrances caused by it. There is a net negative correlation between the size of a molecule and its ability to penetrate the blood-brain barrier. In the BBBP dataset this correlation is $r = -0.305$, which suggests that this feature, which could’ve been abstracted during pre-training may have been useful for this task.

To quantify the abstractions, PCA was performed on the fingerprints to reduce to 2 dimensions and K-means was used to cluster the data and assess the models’ performances. Table 4.5 shows the K-means performance on the pre-trained and fine-tuned models on the three classification datasets after PCA for dimensionality reduction.

Given the data in table 4.5, we can see that the fine-tuned models have significantly better performance than the pre-trained models. This is expected given that the fine-tuned models were trained on the specific tasks. However, the pre-trained models have also consistently scored (slightly) better than random across all tasks, which suggests that the pre-trained models have learnt some general representation that is task agnostically valid. This is also supported by the

t-SNE visualisation in figure 4.9.

4.3.4 MFBERT/Siamese MFBERT as a Molecular Property Predictor

Given its task agnostic nature, and fine-tune-ability for chemical tasks, the transfer learning approach of MFBERT means that it is capable of predicting molecular properties across a variety of tasks. The following set-ups are benchmarked: a) MFBERT pre-trained fingerprints b) MFBERT fine-tuned fingerprints c) Siamese MFBERT pre-trained fingerprints and d) Siamese MFBERT fine-tuned fingerprints on the following molecular property prediction datasets from MoleculeNet:²⁹

- **ESOL**²³⁹ - A dataset of 1128 organic compounds with measured water solubility. The task is to predict the water solubility of a molecule.
- **FreeSolv**²⁴⁰ - A dataset of 642 small molecules with experimentally measured and calculated hydration free energy in water. The task is to predict the experimentally determined free energy.
- **Lipophilicity**²⁹ - A dataset of 4200 organic compounds with measured logP values. The task is to predict the logP of a molecule.
- **BBBP**²³⁴ - A dataset of 2036 organic compounds with a label as to whether they are active as a blood-brain barrier permeable molecule.
- **HIV**²³⁵ - A dataset of 4110 compounds with a label as to whether they have experimentally been able to inhibit HIV replication.
- **Tox21**²⁹ - A multi-task dataset of 7831 compounds with 12 different labels as to whether they have experimentally been able to inhibit a specific protein.

- **ClinTox**^{236,237} - A dataset where FDA approved/disapproved drugs have been tested for their toxicity in a cell line. The task is to predict a binary label of the toxicity of a molecule.

These datasets are both regression tasks (ESOL, FreeSolv, Lipophilicity) and classification tasks (BBBP, HIV, Tox21, ClinTox), with Tox21 being a multi-task classification task. The recommended splitting strategies as described in the MoleculeNet paper²⁹ are used. These are as follows: ESOL, FreeSolv, Lipophilicity, Tox21, and ClinTox are split into 80% training, 10% validation, and 10% test randomly, whereas for HIV and BBBP Scaffold splitting is used with the same train/valid/test proportions.²⁹ Different neural heads were used for classification and regression with both fingerprints. These include classical ML heads such as SVM, SVR and RF as well as MFBERT'S fine-tuned FFNN head. Table 4.6 shows the results of the different models/fingerprints on the different datasets.

Dataset	MFBERT Pre-trained		MFBERT Fine-tuned			Siamese MFBERT Fine-tuned		
	SVM/SVR	RF	FFNN	SVM/SVR	RF	FFNN	SVM/SVR	RF
ESOL (logS) [†]	1.169 ± 0.161	1.287 ± 0.146	1.153 ± 0.141	0.423 ± 0.501	0.600 ± 0.040	–	–	–
FreeSolv (kcal/mol) [†]	3.021 ± 0.525	2.515 ± 0.360	1.180 ± 0.440	0.999 ± 0.450	0.624 ± 0.130	–	–	–
Lipophilicity (logD) [†]	0.874 ± 0.047	0.999 ± 0.061	0.868 ± 0.021	0.366 ± 0.032	0.461 ± 0.032	–	–	–
BBBP*	0.661 ± 0.037	0.759 ± 0.042	0.762 ± 0.000	0.689 ± 0.048	0.726 ± 0.048	0.750 ± 0.000	0.686 ± 0.062	0.724 ± 0.043
HIV*	0.500 ± 0.000	0.532 ± 0.016	0.765 ± 0.000	0.511 ± 0.009	0.612 ± 0.019	0.765 ± 0.000	0.511 ± 0.013	0.612 ± 0.064
ClinTox*	0.500 ± 0.000	0.555 ± 0.009	0.912 ± 0.000	0.864 ± 0.099	0.837 ± 0.186	0.913 ± 0.000	0.867 ± 0.092	0.845 ± 0.033

*AUCROC, †RMSE

Table 4.6: Molecular property prediction results on the MoleculeNet datasets

The performance of MFBERT is also compared with the state-of-the-art models on the MoleculeNet datasets. Table 4.7 shows the results of the different models/fingerprints from literature on the different datasets.

Table 4.7: Comparison of MFBERT with state-of-the-art models on the MoleculeNet datasets

Dataset	MFBERT (+FFNN)	MFBERT (best)	ChemBERTa	SLIP-FPs (best)	MoleculeNet (best)	MolBERT (fine-tune)
BBBP*	0.762	0.762	0.663	0.741	0.729	0.762
ClinTox*	0.912	0.912	0.842	0.963 $\pm$ 0.044	0.907	0.912
HIV*	0.765	0.765	0.643	0.601	0.792	0.783
ESOL (logS) [†]	1.153 $\pm$ 0.141	0.423 $\pm$ 0.501		0.915 $\pm$ 0.01	0.58	0.531 $\pm$ 0.04
FreeSolv (kcal/mol) [†]	1.180 $\pm$ 0.440	0.624 $\pm$ 0.130		0.891 $\pm$ 0.075	1.15	0.948 $\pm$ 0.33
Lipophilicity (logD) [†]	0.868 $\pm$ 0.021	0.366 $\pm$ 0.032		0.717 $\pm$ 0.029	0.655	0.561 $\pm$ 0.03

*AUCROC, †RMSE

With each dataset/model combination, either the literature value or an obtained value was given based on the availability of the model. For ChemBERTa,³⁷ the mean of the transformer’s contextual embeddings was used like with MFBERT to give it a similar performance boost. For the SSLP-FPs,²³³ the best performing model from the literature was used. The MoleculeNet (best) scores are an aggregated selection of all the best scores achieved by any of the models tested on the MoleculeNet datasets. These models were: RDKit’s ECFP4 + KernelSVM for BBBP and HIV, Weave molecular model for ClinTox, and MPNNs for ESOL and FreeSolv. Finally, Lipophilicity was achieved with a Graph-CNN.²⁹ For fine-tuned MFBERTs, the fundamental model for all tasks is the pre-trained model, relevant heads make the final task-specific predictions. Whilst MFBERT, ChemBERTa, SSLP-FPs and MolBERT²⁰⁰ share the same fundamental model architecture, the differences and optimisations, as well as pre-training and fine-tuning data selection and sampling are what result in the variance of scores between them. One factor thoroughly considered previously in this work, is dataset size and variety. It seems however that on the classification tasks, there seems to be no qualitative correlation between training data size and downstream performance. However, for the regression tasks, models with larger training sets consistently outperform the ones with smaller sets.

The pre-trained and fine-tuned scores seem to be represented well by the t-SNE²³⁸ plots in figure 4.9. Considering BBBP, the pre-trained model scores 0.58 on the AUC-ROC metric with no task-specific priors. This suggests that during MLM, some feature about molecules that the model has abstracted is relevant for BBBP prediction. Further analysis with the Silhouette score shows that indeed the model is able to significantly offer separation between the two classes.

4.3.5 MultiTask Learning

The performance of MFBERT is further assessed by using a multitask learning approach (MTL) where a single fine-tuned MFBERT is able to predict all 12 tasks in the tox21 dataset simultaneously. The same hyperparameters as in the single-task fine-tuning experiments are used with binary cross-entropy loss. The cross-entropy loss is modified to account for the multi-task

nature of the dataset. The loss function is defined as:

$$\mathcal{L} = \sum_{i=1}^{12} \sum_{j=1}^N [-y_{ij} \log \hat{y}_{ij} - (1 - y_{ij}) \log(1 - \hat{y}_{ij})] \quad (4.17)$$

where y_{ij} is the ground truth label for the i th task and the j th sample, and $\hat{y}_{ij}$ is the predicted label for the i th task and the j th sample. As not every molecule was tested for all 12 targets, many had missing labels. Missing labels were accounted for by masking the loss for the target with the missing label, therefore not including it in the sum. The loss was also weighted for each task by employing a weight parameter that is optimised during training. The weighted loss function is defined as:

$$\mathcal{L} = \sum_{i=1}^{12} \sum_{j=1}^N [-y_{ij} \log \hat{y}_{ij} - (1 - y_{ij}) \log(1 - \hat{y}_{ij})] \times w_i \quad (4.18)$$

where w_i is the weight for the i th task. The weights were optimised during training as done in the approach given by Kendall et al.²⁴¹ This meant that the model was not only learning how to perform the tasks simultaneously, but was also learning how to reward or penalise itself. Table 4.8 shows the AUC-ROC score for each task in tox21.

Table 4.8: AUC-ROC scores for each task in the tox21 dataset using MFBERT with MTL

Task	AUC-ROC
NR-AR	0.75
NR-AR-LBD	0.83
NR-AhR	0.85
NR-Aromatase	0.78
NR-ER	0.75
NR-ER-LBD	0.84
NR-PPAR-gamma	0.80
SR-ARE	0.82
SR-ATAD5	0.85
SR-HSE	0.83
SR-MMP	0.78
SR-p53	0.76

It seems that the variance between the AUC-ROC scores for each task is quite low (0.00125) with the lowest AUC-ROC score being NR-ER at 0.75. This suggests that the model is able to learn the tasks simultaneously, and is not overfitting to any one task. However, the average AUC-ROC score is 0.80 and the average precision score across all tasks is 0.8. These scores are lower than other multitask models within MoleculeNet, where the best model achieves an AUC-ROC score of 0.85 when using single-task training. This suggests that the model is not able to learn the tasks as effectively as other models. Given the training strategy used, the model is unlikely to be overfitting to the last task that it has learnt as it is not learning each task sequentially, which is a common problem with sequential MTL, and is known as the *catastrophic forgetting* problem.^{242,243} This problem is usually mitigated by using a continual learning approach, where the model is able to learn the tasks sequentially by reducing learning

on weights that match with each task. This is known as *incremental learning*.²⁴⁴ Future works should investigate this approach with MFBERT. In this case, however, it seems that whilst MFBERT has somewhat effectively learnt the tasks synchronously, other models are able to learn the tasks in this benchmark more effectively.

4.4 Conclusion

In this chapter, an NLP approach to embedding a large portion of the chemical space was proposed and developed. A model that outputs contextual molecular fingerprints based on some downstream task was created. This model, MFBERT was first pre-trained on a large aggregate SMILES dataset of 1.2B+ unique samples. A multi-node distributed training system was developed to allow for practical training on such a large dataset. The pre-trained model was evaluated on the RDKit benchmarking platform - a virtual screening benchmark, and achieved a score of 0.85 on the AUC-ROC metric. This score is an improvement over other state-of-the-art models on this benchmark. An inference optimisation technique was also discovered, where taking the mean of the transformer encoder’s contextual embeddings to pool the molecular fingerprint was found to be more effective than taking the last layer’s contextual embedding [CLS] token. This technique was also found to be effective when fine-tuning the model on downstream tasks. The model was then fine-tuned on a variety of downstream tasks, including the prediction of toxicity, blood-brain barrier permeation, lipophilicity and others from MoleculeNet - a collection of benchmarks. The results showed that MFBERT scores competitively amongst other ML models on a variety of these tasks. A Siamese version of MFBERT was also tested to see if SMILES augmentations through permutations can be further exploited beyond data augmentation. However these showed no added improvement over the original MFBERT model and often performed worse. Finally, a multi-task learning approach was tested on the Tox21 dataset, where the model was able to learn 12 different tasks simultaneously. Future works should investigate the continual learning approach for sequential multi-task learning, and also investigate the ablation of reducing the model and/or data size to see if the model can be made more efficient.

Chapter 5

Combining Learned Representation for Contextual Scoring Functions of Protein-Ligand Binding Affinity Predictions

One of the key areas of interest throughout CADD is the prediction of protein-ligand binding affinities.^{245–249} This is a challenging task due to the complexity of protein-ligand binding and the interactions between the two entities. The binding process is usually a multi-step process that may or may not depend on the dynamics of a protein²⁵⁰ (i.e some proteins only have their active site open during a small conformational area within the conformation space). Not only that but binding may not occur within the active site, and may be allosteric, where the ligand binds to a different site on the protein and induces a conformational change that leads to the protein function or inhibition.²⁵¹ This is a very complex process that is not fully understood, and is the reason why predicting binding affinity is an important but challenging task.

In this chapter, we will discuss the current state of the art in protein-ligand binding affinity prediction, and how we can use learned representations from MFBERT (Chapter 4) to improve

the accuracy of scoring functions of protein-ligand binding affinity. Interdimensional structure representations of the protein and ligand are used with a combination of two deep learning models to develop a system that is fast and robust for protein-ligand binding affinity scoring. This composite model and structure representation system is trained on the PDBbind 2020 dataset^{252–254} and benchmarked on the CASF-2016²⁵⁵ and CASF-2013²⁵⁶ datasets - the current standard benchmarks for scoring functions.

5.1 Protein-Ligand Binding Affinity Prediction

Predicting protein-ligand binding affinity *in silico* is an important step in the digital drug discovery pipeline and has been an essential part of CADD for many years.^{257–261} The binding affinity of a ligand to a protein is a measure of the strength of the interaction between the two entities. The binding affinity is usually measured in terms of the dissociation/inhibition constant (K_d/K_i) or the half maximal inhibitory concentration (IC50), which is the concentration of the ligand at which half of the binding sites are occupied. In this chapter K_d/K_i will be grouped together as K_a . These three measures can be thought of as ways to experimentally abstract the interaction strength or the ease of reaching a binding pose between the protein-ligand complex (PLC) based on the fundamental atomic protein-ligand interactions (PLIs). As such, classical computational methods that determine the binding affinity of a ligand to a protein are usually based on calculating the PLIs across varying conformations and ligand poses - with the most stable combination being the most optimal pose. Once the PLIs are computed, a K_a can be calculated using the proportionality between K_a and the Gibbs free energy of the PLC binding (ΔG).²⁶² This proportionality is given by the following equation:

$$pK_a = -\frac{1}{\ln(10)} \frac{\Delta G_0^{bind}}{RT} \quad (5.1)$$

Where R is the gas constant and T is the temperature. The ΔG_0^{bind} is the Gibbs free energy of binding. This proportionality is important as it allows for the decomposition of the binding free energy into the sum of the free energies of the individual atoms - and therefore the same

decomposition can be applied when calculating the K_a of a PLC. The methods used to predict the PLIs are usually referred to as Scoring functions, as they score the binding affinity of a ligand to a protein. The most common classic scoring functions depend on computing the electrostatic and van der Waals (VdW) interactions between the atoms of the ligand and the protein. However, renewed interest in this area given recent increases in computational power and the rise of ML and DL methods resulted in scoring functions becoming more complex as well as less defined.^{263–269} A few examples of scoring functions are discussed in the following section.

5.1.1 Scoring Functions

Currently, there are two broad categories of scoring functions - classical functions which rely on theoretical models of PLIs^{248,257} and ML based scoring functions^{263–269} which rely on training data to allow a model to abstract a scoring function. There also exists knowledge-based scoring functions, which are based on statistics of known structure interactions in structural databases such as PDB or CSD.²⁷⁰ Classical scoring functions can exist as either physics (force-field) or quantum mechanics (QM) based models which compute the PLIs, or as regression based models, which rely on experimental results.

Force-Field and QM Based Scoring Functions

Like the AMBER GAFF,¹²³ many variations of force-fields exist for scoring functions depending on the software used. For example Autodock vina uses the Autodock4 Force-field,²⁷¹ which includes terms for VdW interactions, Hydrogen Bonding, Electrostatics, Desolvation and Torsional Free Energy.²⁷² It also accounts for zinc-coordinating ligands by using pseudo-atoms for the tetrahedral coordination geometry of a Zn-protein complex. Some force-field based scoring functions are also combined with empirical factors such as solvent accessible surface area (SASA) and counts of hydrophobic contacts. Other docking force-fields include: GoldScore^{273,274} and GalaxyDock.^{275,276} In general, a force-field based scoring function is of the following form:

$$\Delta G_{binding} = \Delta E_{VdW} + \Delta E_{HB} + \Delta E_{Elec} + \Delta E_{solv} + \Delta E_{Torsion} + \Delta E_{Empirical} \quad (5.2)$$

Where $\Delta E_{Empirical}$ may include torsional degrees of freedom as well as system-specific corrections. Knowledge-based scoring functions and QM corrections can also be combined with force-field scoring functions through this term. It is important to note that the complexity for docking scoring functions is not as limited by computational speed as for MD force-fields as the number of force-field computations required for docking is much lower than for MD simulations. This is because the number of conformations that need to be sampled is lower than for MD simulations, and in most cases, the protein conformation is not changed from the static/relaxed crystal structure.

Regression Based Scoring Functions

Regression based scoring functions are usually based on the experimental data of the binding affinity of a ligand to a protein. The experimental data is usually obtained from the literature or from the Protein Data Bank (PDB).⁵⁷ Coefficients of a functional form with pre-defined terms are determined by fitting the model to the experimental data to obtain a scoring function. This is similar to ML based scoring functions, however regression based functions are usually linear combinations of contributions. A general empirical scoring function is of the following form:

$$\Delta G_{binding} = w_0 + w_1 \Delta E_{VdW} + w_2 \Delta E_{HB} + w_3 \Delta E_{Entropy} \quad (5.3)$$

Where w_0 is the intercept and w_1 , w_2 and w_3 are the coefficients of the terms.²⁷⁷ In some empirical scoring functions, like ChemScore,^{278,279} value scores are added either as a reward or penalty to the binding free energy. This is usually done to account for the fact that some PLIs are not well captured by the scoring function. For example, the value score for a hydrogen bond is usually 1.0, and the value score for a van der Waals interaction is usually 0.5. This is because the scoring function may not be able to capture the exact strength of the interaction,

and therefore a value score is added to account for this. The value score is usually added to the binding free energy as a reward, and therefore the scoring function of ChemScore is of the following form:

$$ChemScore = w_0 + w_1 S_{HB} + w_2 S_{metal} + w_3 S_{lipo} + w_4 H_{rot} + E_{int} + E_{clash} + E_{cov} \quad (5.4)$$

Where S_{HB} is the hydrogen bond score, S_{metal} is the acceptor metal coordination score, S_{lipo} is the lipophilic interactions score, H_{rot} is the loss of conformational entropy after PLC formation, E_{int} is the internal energy of the ligand, E_{clash} is the clash penalty and E_{cov} is the covalent energy. More recent empirical scoring functions include: Vinardo,²⁸⁰ DockTScore,²⁸¹ DockThor²⁸² and AA-Score.²⁸³

Knowledge Based Scoring Functions

Knowledge-based scoring functions derive a potential of mean force based on the statistics of close intermolecular interactions between specific atoms or functional groups. It is assumed that those interactions which are found more frequently in a large structure database than a random distribution are more energetically favourable, and therefore contribute to binding affinity. As such, these scoring functions are based on statistical potentials of the form:

$$S = \sum_{i \in lig} \sum_{j \in prot} \omega_{ij}(r) \quad (5.5)$$

Where $\omega_{ij}(r)$ is the pairwise potential based on distance r between atoms i and j . It is defined as:

$$\omega_{ij}(r) = -k_b T \ln \left(\frac{\rho_{ij}(r)}{\rho_{ij}^0} s \right) \quad (5.6)$$

Where $\rho_{ij}(r)$ is the density of the interaction between atom types i and j at distance r in the database, ρ_{ij}^0 is the reference density state where there is no interaction between atoms i and j

in the distribution²⁸⁴ and s is a scaling factor. The scaling factor is usually set to 1.0, however it can be set to a different value to account for the fact that the database may not be a random distribution. Therefore, a positive contribution to the binding affinity for interactions with atom types $i - j$ would exist in the scoring function if $\rho_{ij}(r) > \rho_{ij}^0$.

It is important to note that since knowledge-based scoring functions are statistically comparative, one of their main benefits is the implicit accounting for fundamental theoretical interactions such as entropic and solvation effects, although, these terms, and others, can be explicitly added to the scoring function. Examples of knowledge-based scoring functions include: SMOG,^{285,286} DrugScore,^{287–290} ITScore,^{291,292} KECSA,²⁹³ M-score,²⁹⁴ DLIGAND2²⁹⁵ and Korp-PL.²⁹⁶

Further information on the different types of scoring functions and their uses can be found in Meli et al.’s review.²⁹⁷

5.2 ML Based Scoring Functions

One of the main issues with classical computational methods is that they are not able to capture the complex nature of the binding process. This is due to approximations made within the models to allow for computational time limitations. As such, there is often a compromise between accuracy when scaling this approach across large databases in scenarios such as virtual screening. Not only that, but the ground truth of scoring can only be determined experimentally for each system; experimental data contains a lot of variance for the same experiment, therefore it is incredibly difficult to optimise scoring functions. However, despite these difficulties, a great deal of progress has been made with classical scoring, ML scoring and experimental data, with these improvements in each area complementing the other. This section will discuss the current state of ML based scoring functions, the experimental data available to train and test them as well as some benchmarking methods.

5.2.1 Datasets

Over the years, datasets of binding affinity data have been compiled from the literature and from the PDB.⁵⁷ These datasets are now widely used across many applications, from benchmarking scoring functions to training ML models. Table 5.1 shows some of the publicly accessible datasets that provide the 3D structure of a PLC along with the corresponding binding affinity.

Table 5.1: Publicly accessible datasets of binding affinity data

Dataset	Size	Superset
PDBbind 2020	19,443	NA
CASF-2016	285	PDBbind 2016
CASF-2013	195	PDBbind 2013
CASF-2007	195	PDBbind 2007
Binding MOAD 2020	15,223	NA
CSAR-NCS HiQ	343	Binding MOAD + PDBbind
CSAR-NCS HiQ Update	123	Binding MOAD + PDBbind
Astex Diverse Set	74	NA
BindingDB	11,442	NA
D3R GC 4	20	NA
D3R GC 3	24	NA
D3R GC 2	36	NA
D3R GC 2015	24	NA

In this chapter, we will consider in detail, the PDBbind²⁵²⁻²⁵⁴ 2020 set and its benchmark subsets CASF-2016²⁵⁵ and CASF-2013.²⁵⁶

PDBbind 2020

PDBbind^{253,254} is one of the most common datasets used to train regressive and ML scoring functions (SFs) due to its large size and the fact that it is publicly available.^{298–300} Initially, PDBbind began in 2004 when PLCs from the PDB⁵⁷ were aggregated and their references screened for the binding affinity data. This dataset grew over the years, with the latest version at the time of writing being PDBbind 2020, which contains 19,443 PLCs. The dataset also includes data for protein-protein and ligand-nucleic acid complexes, however, these were excluded in this work. Over the years, additional features and filters were added to PDBbind. This includes the addition of the “refined” set, which is a subset of the “general” set. The refined set only contains PLCs with a resolution of greater than or equal to 2.5 Å, where the PLC contains only one ligand that isn’t covalently bonded and has no significant steric clashes. Also, only systems with K_i and K_d values were included as IC_{50} values are heavily dependent on the experimental design of the assay and therefore are more difficult to understand and interpret. From PDBbind 2013, further criteria were added to filter structures into the refined set. These included: 1) No extreme K_a values. (specifically, $1pM < K_a < 10mM$). 2) No multiple binding sites where the K_a difference is greater than 10x. 3) No non-standard amino acids in the binding site (within 5 Å from the ligand). 4) No shallow binders (where less than 15% of the ligand surface is buried). PDBbind 2016 onwards also include copies of the structures where only the binding-site is present (that is, atoms within 10 Å of the ligand). This is to allow for the use of the binding-site only in the scoring function.³⁰¹ The refined set is also used to create the Comparative Assessment of Scoring Functions (CASF) benchmark sets. Further filtrations to PDBbind were performed in this work, they are discussed in section 5.3.

Comparative Assessment of Scoring Functions

The Comparative Assessment of Scoring Functions (CASF)^{256,301} is a subset of the PDBbind refined set that is used as a standardised evaluation, or benchmark for scoring functions on scoring as well as ranking, docking and screening tasks.

One of the main features that is needed for a good benchmark is the diversity of the samples within it. This is because the benchmark should be able to test the scoring function on a wide range of different systems. This is achieved in CASF by only considering the high-quality structures (as it is a subset of the refined subset), and clustering them according to amino-acid sequence similarity using the Basic Local Alignment Search Tool (BLAST). Structures which have $> 90\%$ similarity were included in the same cluster as they are considered to be likely to be of the same protein family. Only clusters consisting of four or more PLCs were included, resulting in a total of 65 clusters that could be used in PDBbind/CASF-2007. From each cluster, three samples are systematically selected, one with a high binding affinity, one with a low binding affinity and one with a close-to-the-mean binding affinity. This results in a total of 195 samples in CASF-2007.

For CASF-2013, the same process was used to generate the dataset, however, only clusters with five or more PLCs were considered. Moreover, the variance in the selected samples must be significant. As such, only PLCs with a binding affinity difference of at least 10x were considered. Due to the increase in available data in the PDB (and therefore in PDBbind) between 2007 and 2013, this stricter filtering resulted in the same number of valid clusters and the same number of selected PLCs for the benchmark.

Further data and refinements were introduced for CASF-2016. PDBbind 2016's refined set consists of 4057 PLCs. They were clustered with BLAST, then the clusters were sampled as above, however, rather than sampling only three PLCs, five were sampled from each cluster. The selection method in this case targeted the PLC with the highest binding affinity, the lowest binding affinity and the three that are most evenly distributed between them. This resulted in a total of 57 clusters and 285 PLCs in CASF-2016.

Critical Assessment of Structure Prediction (CASP)

CASP (Critical Assessment of Techniques for Protein Structure Prediction) is a biannual competition for research groups to compare and assess their protein folding modelling algorithms,⁴⁰ from *ab initio* to template-based, there are various metrics and sub assessments to test the

prediction quality. The protein structures are predicted from their amino acid sequence. Experimental groups hold out some of their newly solved protein structures (through X-ray crystallography/NMR) for this competition before publication. All the solved target structures from previous CASP experiments are archived and aggregating them gives the CASP dataset. This dataset consists of high-quality protein structures that have been thoroughly experimentally verified and are suitable for computational prediction - that is, they are not too large or too small, as they are selected for prediction purposes.

From the aggregated CASP archive, a subset of 800 protein structures were selected for this work. These were high diversity structures, including secondary, tertiary and quaternary proteins. The smallest system selected consisted of 299 atoms and the largest 18192 atoms. The selected subset only contained atoms of type C, H, O, N and S. The structures are as much as possible from different protein families, with some structures being completely novel in a sense that there are no similar sequences which are used to test de novo modelling techniques.

In this work, the CASP dataset was used to pre-train the Neural Network Potential (NNP) sub-model used in our deep learned scoring function (DLSF) (see section 5.3.1). This protein-only dataset, set the domain of the NNP to be familiar with proteins. This was achieved by running MD simulations on the solved structures to scan the conformational space and compute the potential for each conformation. This was then used to train the NNP sub-model. The NNP sub-model was then used with MFBERT to train the DLSF on the PDBbind 2020 set.²⁵³

5.2.2 Neural Network Potentials

One of the most common applications of DL in computational chemistry is the task of using a neural function to approximate, or abstract, the potential energy surface (PES) of a system.^{54,302–309} Potential energy of a system is computed in essentially all computational tasks, from classical/molecular mechanics (MM) based docking and MD, to Density Functional Theory (DFT) for Quantum Mechanics (QM). Neural Networks and DL have shown promising performance in potential energy approximations, with quantum level accuracy at classical level speeds. This can be attributed to the fact that DL NNs are universal function approximators

that are (theoretically) able to learn the non-linearities in the potential in a parallel matrix efficient manner. Although there have been many implementations of various Neural Network Potentials (NNPs), in this work we will specifically consider the ANAKIN-ME (Accurate NeurAl network engINe for Molecular Energies) NNP (ANI),³⁰⁸ in particular, its PyTorch implementation, TorchANI,³¹⁰ which featurizes the system using modified Behler and Parrinello symmetry functions (BPSFs)³⁰⁶ and discretises the system atomically. With each atom type having its own FFNN that takes in the atom’s Atomic Environment Vector (AEV - a vector of the BPSFs of the atom and its neighbours) and outputs the atom’s energy. The total energy of the system is then the sum of the energies of all the atoms in the system.

ANAKIN-ME

The pipeline of ANI consists of a number of steps to compute a single atomistic energy, which is then summed for the total system energy. The first step is to compute the AEV of the atom using the BPSFs which describe the local environment of an atom. The BPSFs are a set of radial and angular functions that are used to describe the local environment of an atom.^{306,308} The radial functions are used to describe the distance between the atom and its neighbours, while the angular functions are used to describe the angle between the atom and its neighbours. The combination of the radial and angular functions (the AEV) describe the local environment of the atom. An AEV is a set of m elements as parameterised by the BPSFs and is defined as: $\vec{G}_i^X = \{G_1, G_2, G_3, \dots, G_m\}$, where G_i is the i^{th} element of the AEV and X is the atom type. The AEV is then passed to the FFNN of the atom type to compute the atom’s energy which is then summed across all atoms in the system to compute the total system energy.

$$V = \sum_{i=1}^N V_i \quad (5.7)$$

Where V is the total system energy, V_i is the energy of the i^{th} atom and N is the number of atoms in the system.

Atomic Environment Vectors

The main innovation of ANI³⁰⁸ is the AEV, which is a vector of the BPSFs of the atom and its neighbours. It allows the functional form of the total energy to be discretised and used within a FFNN. As described above, the AEV describes the atomic environment of each atom of type X in a system. It is important to note that the AEV only considers the local environment for each atom, and does not consider the global environment of the system. This locality is approximated through a piece-wise cut-off function:

$$f_C(R_{ij}) = \begin{cases} 0.5 \times \cos\left(\frac{\pi R_{ij}}{R_C}\right) & \text{for } R_{ij} \leq R_C \\ 0 & \text{for } R_{ij} > R_C \end{cases} \quad (5.8)$$

Where R_{ij} is the distance between the i^{th} and j^{th} atoms, R_C is the cut-off distance - a parameter and f_C is the continuous cut-off function. The AEV is then a combination of radial and angular elements. The radial part is defined by the BPSFs and is parameterised by the cut-off function:

$$G_m^R = \sum_{j \neq i}^N \exp(-\eta(R_{ij} - R_S)^2) f_C(R_{ij}) \quad (5.9)$$

Where G_m^R is the m^{th} radial element of the AEV, that is, the m^{th} parameter set of R_S and η , given a set of m parameter pairs. R_{ij} is the distance between the i^{th} and j^{th} atoms, R_S and η are the shift and width parameters of the Gaussian distribution respectively. For the ANI-1 model, only one η value is used with many R_S values to generate ring peaks that probe a wide range of areas within the local radial environment.

The angular BPSF used in ANI-1 is modified. A θ_S parameter is added to the angular BPSF to allow angular function shift.³⁰⁸ The R_S parameter is also added to the angular symmetry functions. This allows the local angular environment to be probed across multiple radii that are correlated with the average distance between an atom and its neighbours. The angular part of the AEV is thus defined as:

$$G_m^A = 2^{1-\zeta} \sum_{j,k \neq i}^N (1 + \cos(\theta_{ijk} - \theta_s))^\zeta \exp \left[-\eta \left(\frac{R_{ij} + R_{ik}}{2} - R_s \right)^2 \right] f_C(R_{ij}) f_C(R_{ik}) \quad (5.10)$$

Where G_m^A is the m^{th} angular element of the AEV. θ_{ijk} is the angle between the i^{th} , j^{th} and k^{th} atoms, θ_s is the shift parameter of the Gaussian factor, R_{ij} and R_{ik} are the distances between the i^{th} , j^{th} and i^{th} , k^{th} atoms respectively, R_s and η are the shift and width parameters of the Gaussian distribution respectively, ζ is the angular symmetry function exponent, which also changes the width of the peaks in the angular environment. For the ANI-1 model, only one η value is used with many R_s and θ_s values to generate thin, specific peaks that probe a wide range of areas within the local angular environment. Figure 5.1 shows an example AEV of an atom in a H, C, N, O system.

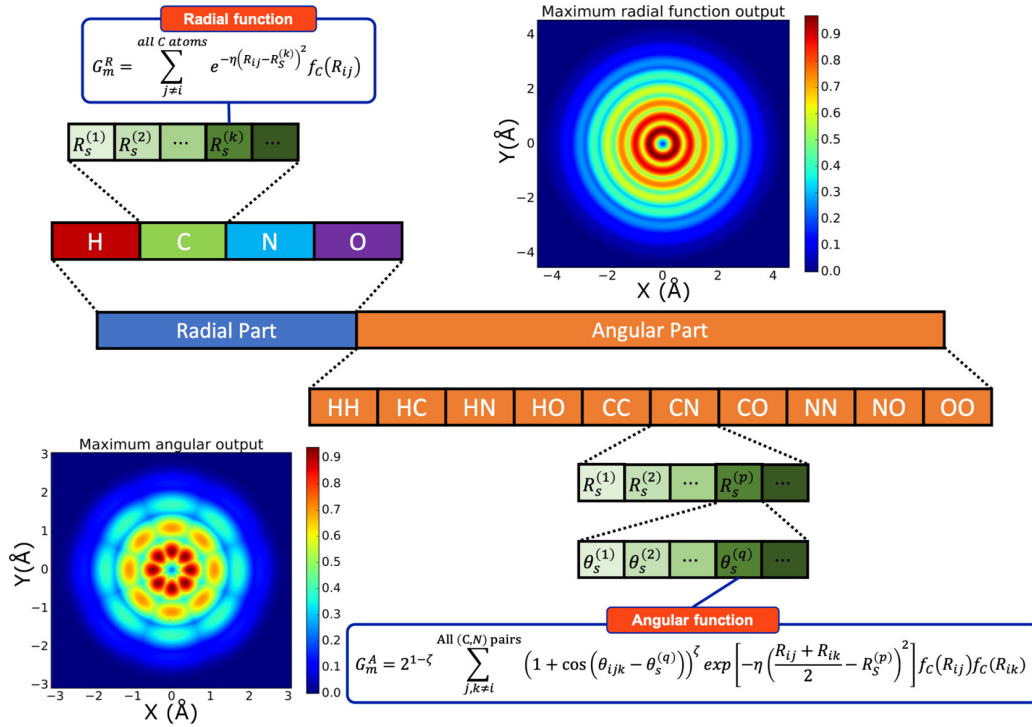


Figure 5.1: Example AEV of an atom in a H, C, N, O system. Reproduced from Gao et al.³¹⁰

This AEV is computed for every atom in the system and passed to the FFNN of the atom type to compute the atom's energy. That is, a separate neural network exists for every atom type. ANI-1 was trained on DFT ω B97X/6-31G(d)^{311,312} level quantum computations on a synthetically generated dataset (a subset of GDB-11^{313,314}) created by random sampling the

conformational space of 57k organic molecules of elements C, N, O, H. Normal mode sampling was used to ensure near-equilibrium structures were computed. It is also worth noting that the latest version of ANI at the time of writing, ANI-2x³¹⁵ was trained on a larger dataset built on top of the ANI-1 dataset through further conformational sampling as well as the addition of more data points. Further additions to BPSF parameterisations and training data also meant that ANI-2x was now able to predict systems with elements S, F and Cl at the same level of theory. The ANI-2x parameters and model architecture are described in detail in Devereux et al.’s work.³¹⁵ Further references to ANI in this thesis will refer to ANI-2x.

5.3 Learning Binding Affinities With MFBERT and Neural Network Potentials

In this work, an MFBERT-ANI combined architecture to learn binding affinities of small molecules to proteins is developed, with the MFBERT part focusing on the ligand, and the ANI part focusing on the protein. Some of the main features of this approach is the non-reliance on a 3D ligand structure, as it is abstracted within the latent space of MFBERT from the SMILES input. This massively speeds up the prediction process compared to other SFs and allows the use of larger datasets where 3D ligand structures are not available. It also allows testing large databases of ligands for a particular target, quickly in a virtual screening like manner. This section discusses the details of the model, training, data and pipeline.

5.3.1 Data

For this model to work effectively, large quantities of high quality and varied data are required. Since datasets with those two qualities are rare, a pre-training and fine-tuning strategy was employed such that more data could be used for training whilst fine-tuning on the higher quality data ensures the model’s accuracy remains at the higher level of the higher quality data.

Pre-training

The MFBERT part of the model is pre-trained on a large dataset of SMILES strings with MLM as described in chapter 4. For the ANI part, an energy dataset was created with MD from the CASP structure archives.

CASP Energy Dataset Creation

For this dataset, 801 human-like proteins from the CASP structure archives were used. The structures were cleaned with `pdb4amber` and `reduce`.^{119,120,122} Once hydrogens were added, topology files were created using `LEaP`^{119,120,122} and the `ff19SB` force-field.¹¹⁸ No proteins contained metal ions (or any of the atom types not supported by ANI-2x³¹⁵). Each protein was minimised for 500 steps with steepest descent, then 500 steps with conjugate gradient and heated to 298.15K in 100k, 0.5 fs timesteps before the production run. For the MD production run, Generalised Born implicit solvation was used (specifically `IGB=5`¹²¹) and `SHAKE`³¹⁶ was used to constrain the bonds and angles. Each simulation was run for 10ns with 2 fs timesteps with a frame saved every 1 ps for analysis. Once all the simulations were complete, the energies were extracted from the trajectory files and then normalised. The dataset was split into training, validation and test sets with 80%, 10% and 10% of the data respectively. The dataset was then shuffled and used to pre-train ANI-2x.³¹⁵ This pre-train step was done to familiarise our model with human-like protein structures as the original ANI-2x was only trained on small organic molecules.

Fine-tuning

For the MFBERT part, no fine-tuning was done. For the ANI part, the model is fine-tuned on the original DFT ANI dataset described in Smith et al.'s work.³¹⁷

Combination Training

Once each part of the model has been pre-trained and fine-tuned independently on their respective datasets/tasks (MLM for MFBERT and regression for ANI). The combined model is then trained on the PDBbind 2020 dataset²⁵³ (see section 5.2). Since the CASF-2013 and CASF-2016^{255,256,301} benchmarks (what is used to test the model in this work) are subsets of PDBbind, the PLCs from these benchmarks were removed from the PDBbind 2020 dataset such that there was no overlap. Whilst there is no exact overlap between PLCs in the training and benchmarking sets (“hard overlaps”), there exists some “soft overlaps” where proteins, ligands and/or binding sites may be too similar between some PLCs in the training set and some PLCs in CASF. To address this, following Su et al.’s algorithm,³¹⁸ a similarity threshold of 90% across the three metrics was used to remove PLCs from the training set that were too similar to PLCs in the CASF benchmarks and were therefore redundant. To compute the three metrics used, ggsearch in FASTA³¹⁹ was used to compute sequence alignment, which then was used to compute the maximal similarity between any pair of chains, one from each protein. For ligand similarity, the 3D ligands were aligned using the Kabsch algorithm in pytraj³²⁰ and the RMSD was computed. For binding site similarity, the binding site was defined as the region of the protein within 4.5Å of the ligand. The binding site was then aligned using the Kabsch algorithm and the RMSD was computed. An RMSD of 0.5Å was used as the threshold for similarity. This was also used in combination with the tanimoto similarity of the ligand to decide to remove redundant ligands. PocketMatch³²¹ was used to compute the binding-site similarity. It uses an algorithm that quantifies similarity based on residue types and interactions within the pocket. As defined in PDBbind 2020, a pocket is any residue that is within 10Å from any ligand atom. Once the similarity metrics were computed, Heifetz’s algorithm^{318,322} was used to compute the similarities across the entire datasets and to combine the similarity metrics into a single similarity score which the 90% threshold was applied to. Further analyses of the distribution of PLCs, their similarities, metrics and other statistics within CASF-2013 and CASF-2016 are given in Li et al.²⁵⁶ and Su et al.’s work³¹⁸ respectively.

Initially the entire remaining PDBbind 2020 PLC dataset was used for pre-training, excluding

PLCs with metal ions and other atom types not supported by ANI-2x.³¹⁵ PLCs with S, P, Br and I were kept and were treated the same as their closest neighbour supported by ANI-2x (P-S and Br/I-Cl). Once pre-trained, the model was then trained on the PDBbind 2020 refined set with the exclusions above. Metal complexes however, were added back at this stage with metal ions being treated as a dummy element, X. This meant that all metal ions were treated identically, which is a limitation of the model. PLCs with other elements (such as Lithium and Selenium) were removed.

For the CASF-2013 benchmark, the binding pocket was not already provided. The binding pocket was therefore extracted using pytraj³²⁰ with the same 10Å threshold used in CASF-2016 and PDBbind 2020.²⁵⁵

5.3.2 Model Architecture

Most current ML models that learn binding affinities or abstract PLIs are either FFNN, CNN, Graph or VAE based or some combination of the above.^{38,263,265,266,269} In all four cases, a 3D model of the full system is required, that is, if you'd like to try and predict the binding affinity of different ligands to the same protein, you'd need to generate a 3D model of the PLC for each ligand, either through docking or MD or other classical/quantum energy minimisation/fitting methods. A general NN based SF is shown in figure 5.2.

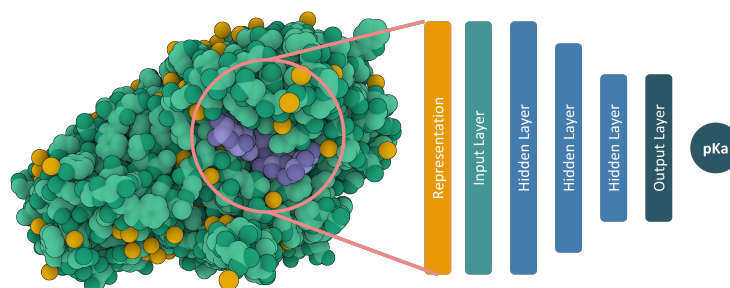


Figure 5.2: Representation of a general deep learning system applied to a protein-ligand complex (PDB: 1XD0). The binding pocket is represented computationally (usually as a graph or a matrix) and is passed through a set of hidden layers, depending on the architecture of the model to predict the pKa. Protein illustration was generated with Mol*³²³

In general, system (PLC) representation plays a huge role in the robustness and performance of

the model. Some models represent the whole graph whilst others combine force-field parameterisations and flatten them into a vector with ligand-protein distances.^{38,302,307,315,324} However, current robust models all require the 3D structure for both the protein and the ligand. It is important to note that recent studies show that most ML SFs tend to be biased towards the ligand 3D structure.^{325–328} Therefore, an architecture which is fast and limits the need for 3D models of the ligand is desirable. This is where the combination MFBERT-ANI model series comes in. Figure 5.3 shows the architecture of the MFBERT-ANI model series.

The architectures shown in figure 5.3 consist of two approaches that were assessed in this work. a) shows the latent model which takes a concatenation of the mean of the final hidden layer of each of the atom-type specific neural networks in ANI.^{308,315} The mean was taken to ensure a fixed vector output size, given that not every PLC would use all the atom-type specific neural networks. Accessing the latent space of each ANI FFNN was achieved using Pytorch’s¹² forward and backwards hooks. This mean latent vector was then concatenated with the mean of the contextual embeddings from MFBERT - the molecular fingerprint of the ligand. This was then concatenated with the Solvent Accessible Surface Area (SASA) of the ligand and the inside of the binding pocket. To accelerate this process during training and inference, a SASA computer was implemented in Pytorch for ease of use (see Linear Combinations of Pairwise Overlaps below). Given the parameter set used for AEV computation, each AEV was a 1008-dim vector. The MFBERT fingerprint is 768-dim and the SASA is 1-dim. For ANI’s FFNNs, the latent space dimensions varied depending on the atom type. This was modified such that the latent space of each FFNN was 160-dim - the maximum dimension across the original set of FFNNs in the ANI-2x architecture.³¹⁵ Therefore, the latent model passes a total of 929-dims to the final FFNN. The final FFNN is a four layer FFNN (not shown) with ELU activations between each layer. ELU activation was used as pK_a is analogous to potential (see equation 5.1). ELUs are smoother than ReLUs and therefore are more suitable for this representation. b) shows the sum model which takes a concatenation of the sum of the final hidden layer of each of the atom-type specific neural networks in ANI. This is also suitable to be used for pK_a prediction, again, due to equation 5.1.

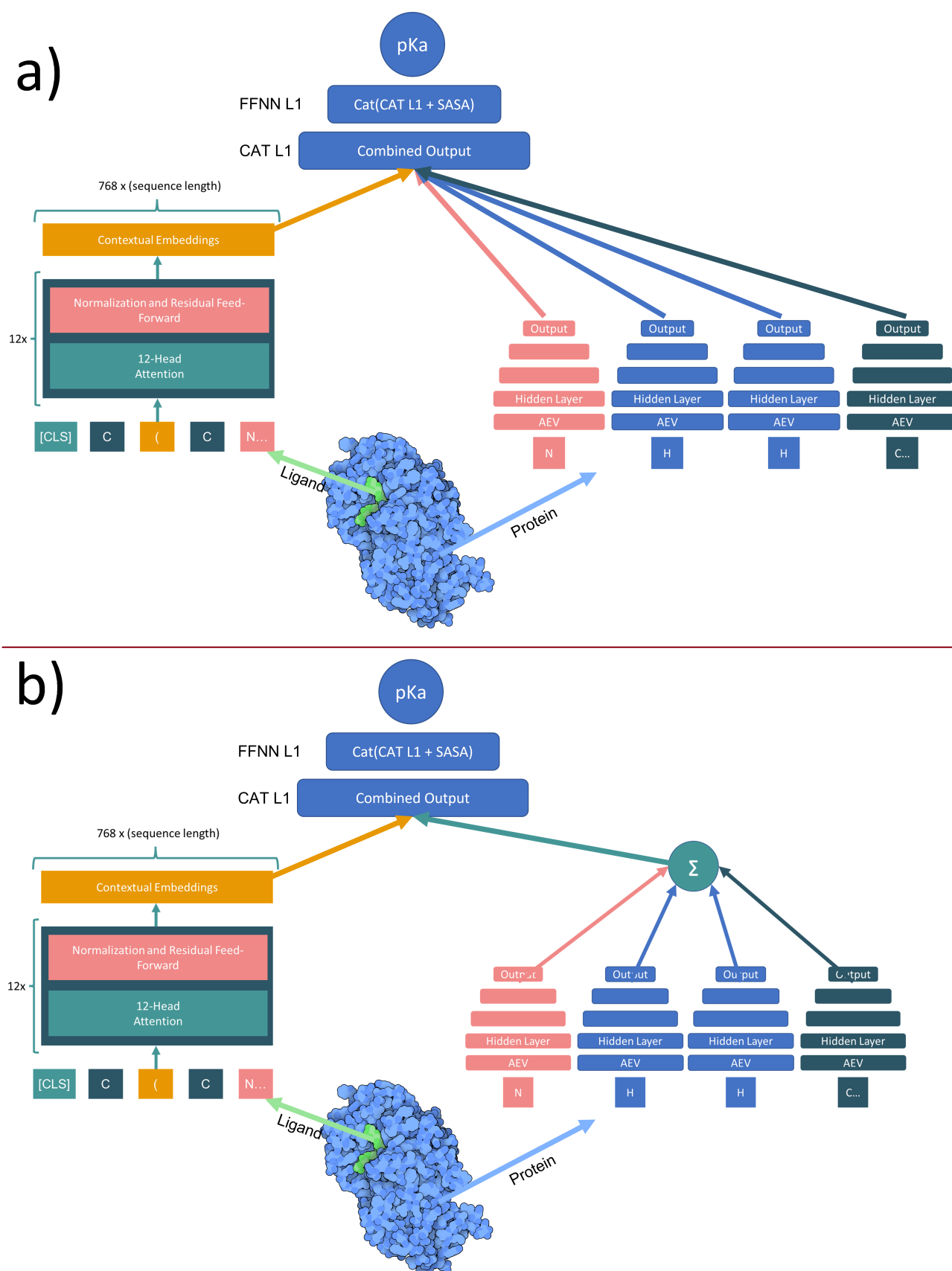


Figure 5.3: Schematic representation of the MFBERT-ANI model series. The protein-ligand complex (PDB ID: 1XD0, rendered with Illustrate³²⁹). Schematic a) shows the latent model of the MFBERT-ANI model series. Schematic b) shows the sum MFBERT-ANI architecture

Linear Combinations of Pairwise Overlaps

In order to rapidly compute the SASA across large PLC pockets/ligands, an approximation algorithm that is commonly used in CADD/MD software, Linear Combinations of Pairwise Overlaps (LCPO)³³⁰ was implemented in Pytorch.¹² LCPO is a fast approximation of the SASA of a molecule. It is a linear combination of the overlaps between atom pairs. The algorithm is as follows: considering two spheres only, i and j of radii r_i and r_j respectively, the accessible surface area (ASA) of i is given by:

$$A_i = S_i - A_{ij} \quad (5.11)$$

$$S_i = 4\pi r_i^2 \quad (5.12)$$

$$A_{ij} = 2\pi r_i \left(r_i - \frac{d_{ij}}{2} - \frac{r_i^2 - r_j^2}{2d_{ij}} \right) \quad (5.13)$$

where d_{ij} is the distance between the centres of the spheres. A_{ij} is the surface overlap between the two spheres. S_i is the total surface area of the sphere and A_i is the accessible surface area of the sphere. The SASA of a molecule is the sum of the ASA of each atom. This simple algorithm can then be extended to more than two spheres - an N-body approximation where the ASA of each atom is calculated using the overlap of all other atoms:

$$A_i = P_1 S_i + P_2 \sum_{j \in N(i)} A_{ij} + P_3 \sum_{\substack{j, k \in N(i) \\ k \in N(j) \\ k \neq j}} A_{jk} + P_4 \sum_{j \in N(i)} A_{ij} \left(\sum_{\substack{j, k \in N(i) \\ k \in N(j) \\ k \neq j}} A_{jk} \right) \quad (5.14)$$

where $N(i)$ is the set of all atoms that are within a cut-off distance of atom i (in other words, the atom neighbours - 2.5Å was the cut-off used in our implementation). P_1 to P_4 are parameters that are used to tune the accuracy of the algorithm. The LCPO authors³³⁰ use linear regression (least squares) to find the optimal values of P_1 to P_4 across a variety of molecules. This concludes the LCPO PyTorch implementation and all parts needed for our MFBERT-ANI models.

5.3.3 Training

For training, the AEVs and SASAs were precomputed and lazy loaded. This was done since both AEVs and SASAs, given the new PyTorch implementations for both, were computed on the GPU. Therefore computing them at training-time with the entire model, AEVComputer, data and SASAComputer loaded into GPU memory would mean lower batch-size and slower training. Not only that, but PLC systems are of different sizes, GPU memory overhead was necessary to avoid out-of-memory (OOM) errors and make training more stable. The Hyperparameters used for pre-training the MFBERT are the same as in section 4.3.2. The ANI pre-training weights were initialised with kaiming normal (a weight initialisation method that specifically accounts for non-linear activations such as ReLU)³³¹ with the a parameter being 1. Both AdamW⁹⁰ and SGD⁹¹ were used for both pre-training and fine-tuning with L2 weight decay. It is important to note that L2 weight decay is the same as L2/ridge regularisation, which in this context is used to prevent overfitting as it makes the weights smaller. Meta Bayesian optimisation with a Gaussian process³³² was used to find the optimal hyperparameters for the MFBERT-ANI model series (see appendix A.4). Particularly, the learning rate, weight decay and number of epochs were optimised. The optimal hyperparameters were found by meta optimisation to be: learning rate of 0.000005, weight decay of 0.048 and 140 epochs. In practise, the model was trained for 500 epochs (with checkpoints at every epoch, and the best checkpoint taken), learning rate of 0.0001 and weight decay of 0.05. The Adam⁹³ optimiser was used for the final training step with a batch size of 16. $\beta_1 = 0.9$, $\beta_2 = 0.999$. The model was trained on a single NVIDIA RTX 3090 GPU. The training time was approximately three days for the refined and filtered PDBbind 2020 set for 500 epochs. The loss function used was the mean squared error (MSE) loss.

5.3.4 Performance and Analysis

Now that our two model variants are trained, inference is performed on the two most common ML SF benchmarks: CASF-2013 and CASF-2016.^{255,256,301} The scoring power is assessed with

model variants as well as system variations, the ranking power and target specific power is also assessed.

Scoring and Ranking Performance

The scoring power of the model is assessed by comparing the model’s predictions to the negative log experimental binding affinities (pKa). To measure the linear correlation between the two quantities in a comparable manner, the Pearson correlation coefficient is used. It is a measure of the linear correlation between two variables and is given by:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (5.15)$$

where x and y are the two predictions and truth values respectively, $\bar{x}$ and $\bar{y}$ are the means of x and y respectively and n is the number of data points. The Pearson correlation coefficient ranges from -1 to 1. A value of 1 indicates a perfect positive linear correlation, a value of -1 indicates a perfect negative linear correlation and a value of 0 indicates no linear correlation. The Pearson correlation coefficient is a measure of the linear correlation between two variables. It is not a measure of the accuracy of the model. The RMSE between predicted and truth across the dataset gives an indication of the performance of the model. Figures 5.4a and 5.4b show a scatter plot of predicted vs. true pKa values, the Pearson correlation coefficient, the spearman rank coefficient and the RMSE between the model predictions and the truth values for the CASF-2013 dataset for both the latent and sum model variants of MFBERT-ANI.

The latent model variant of MFBERT-ANI has a Pearson correlation coefficient of 0.74, a spearman rank coefficient of 0.76 and an RMSE of 1.54. The sum model variant of MFBERT-ANI has a Pearson correlation coefficient of 0.76, a spearman rank coefficient of 0.78 and an RMSE of 1.50. The sum model variant seems to significantly outperform the latent model. In both cases, the model checkpoint with the best validation score during training was used for inference. The improvement in performance of the sum model over the latent model is likely due to the fact that the sum model is able to capture more closely the total environment based

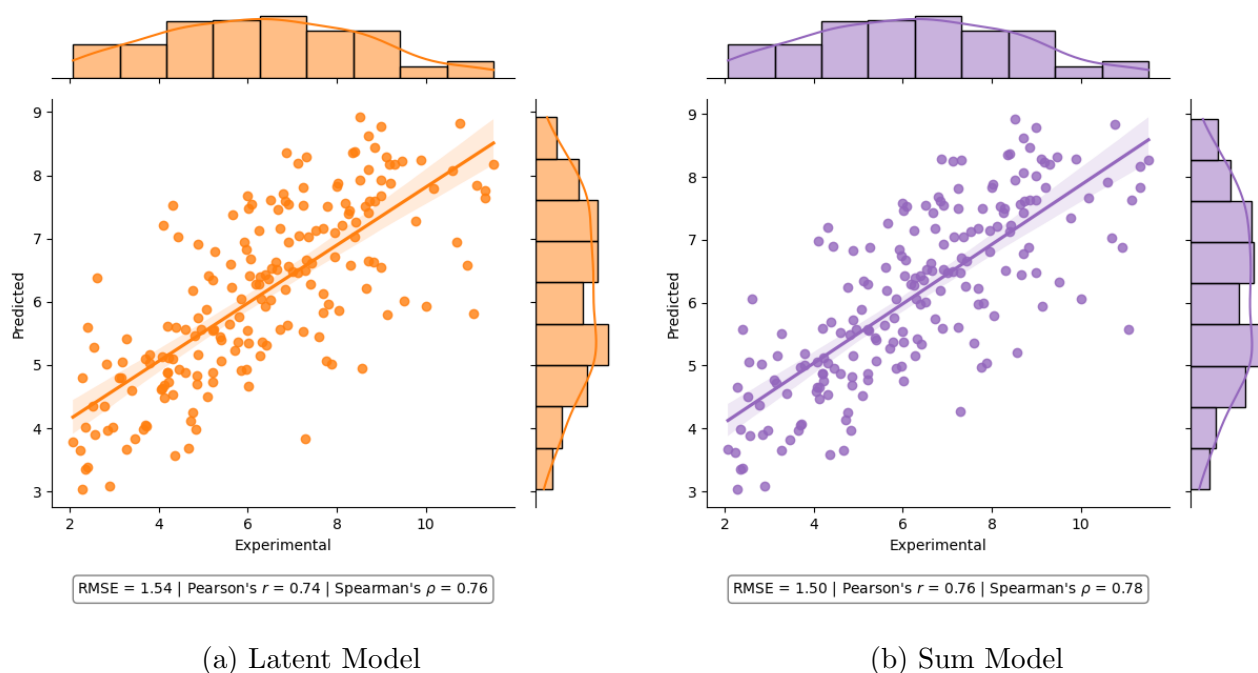


Figure 5.4: CASF-2013 Scoring Performance of both latent and sum model variants of MFBERT-ANI. Values are given in pKa.

on the output of the ANI part for each atom. This can be due to the analogous nature of pKa to ΔG given in equation 5.1. Since energy (and therefore ΔG) is atomically additive, this simplifies the prediction for the model and leads to the better score. Contrasting with this is the latent model which rather than sums the outputs, takes the average vector of all atom environment outputs, which is not as closely related to the additive nature of energy/pKa as the sum model.

Further discretised analysis was performed on the CASF-2013 dataset. Since in the generation of the CASF-2013 dataset, the PLCs were clustered by similarity, each cluster can be classed as a target cluster. The model's performance on each target cluster is considered separately and both Pearson's correlation/RMSE and Spearman rank/RMSE values for each cluster are compared. Figures 5.5a, 5.5b, 5.5c and 5.5d show the Pearson correlation coefficient, the Spearman rank coefficient and the RMSE between the model predictions and the truth values for the CASF-2013 dataset for both the latent and sum model variants of MFBERT-ANI for each target cluster.

Considering Pearson's correlation coefficients of each target cluster and their RMSEs for the

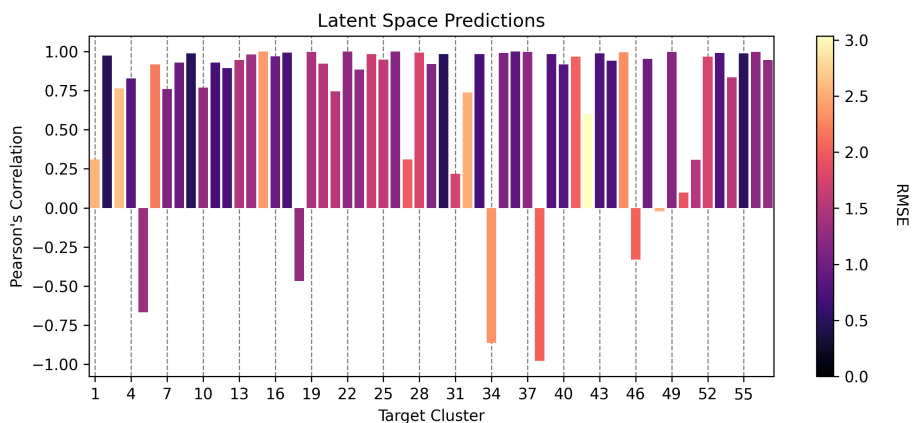
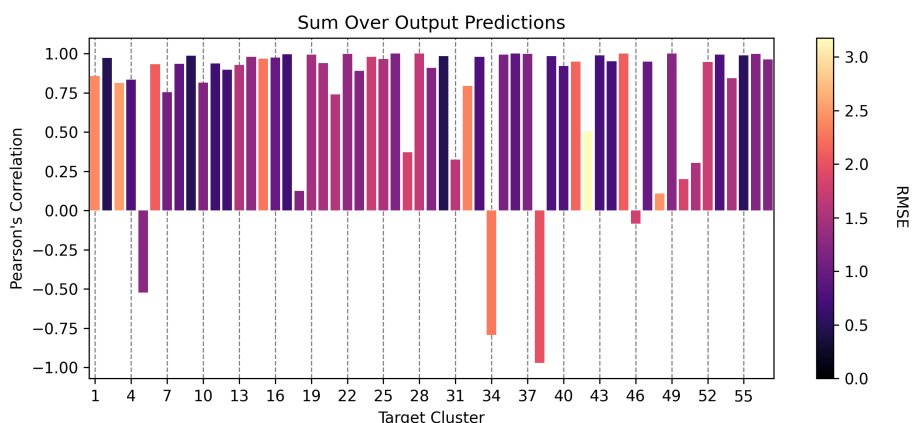
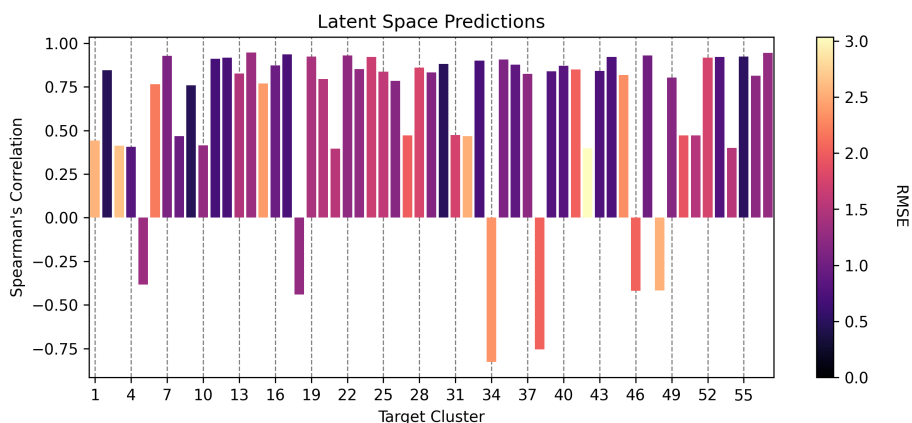
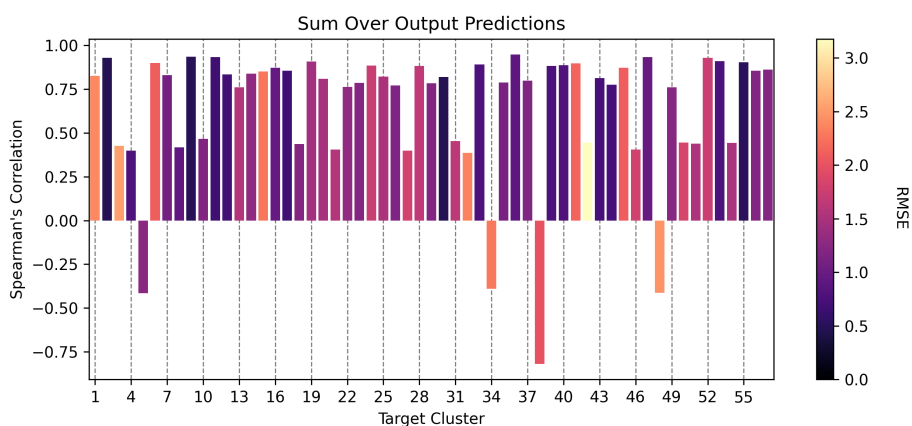
(a) Latent Model Pearson's r (b) Sum Model Pearson's r (c) Latent Model Spearman's ρ (d) Sum Model Spearman's ρ

Figure 5.5: CASF-2013 Scoring Performance on each target cluster of both latent and sum model variants of MFBERT-ANI. RMSE Values are given in pKa.

latent model (Figure 5.5a), we see that whilst the model's predictions may correlate with the ground truth, there is no correlation between that correlation and the RMSE between the model's predictions and the ground truth. For example, consider target clusters 30 and 45, whilst they have similar r coefficients, their RMSE is wildly different, with the RMSE of 45 being much higher than that of 30. This gives some indication on the robustness of the model. Theoretically a higher r correlation coefficient should be complemented by a lower RMSE, however this is not the case for the latent model. What is concerning is the huge range of variety in the r values for each target cluster. Whilst the model is able to achieve positive correlation with most target clusters, some target clusters (5, 18, 34, 38, 46 and 48) have a negative correlation. This is likely due to the fact that the model is not able to capture the correct environment for these target clusters. These clusters are of the following proteins as described in the supplementary information of.²⁵⁶ The proteins are: 5: Angiotensin Converting Enzyme, 18: Mitogen-Activated Protein Kinase 14, 34: Serine/Threonine-Protein Kinase 6, 38: Glutamate Receptor 2, 46: Antibody FAB fragment, 48: Antibody FAB Fragment. These proteins seem to have a high degree of structural homology which could be the reason why the model is consistently not able to capture the correct environment for these target clusters. One other reason to consider when thinking about failure cases is that proteins in real life are dynamic and their conformational changes may impact binding. The CASF-2013 dataset is a static dataset and does not consider this. Considering the sum model in Figure 5.5b, we see that the model's predictions are more robust and the r values are more consistent across target clusters. For example consider target cluster 1 in the latent model and the sum model, the correlation has tripled. Looking at the clusters which had a negative correlation with the latent model, we see much of the same but at a reduced correlation magnitude. For example, target cluster 46's correlation is still negative but not by much. Two negatively correlated clusters have flipped by switching from the latent to the sum model - 18 and 48. However, 5, 34, 38 and 46 remained negative.

Now let's consider the Spearman rank correlation coefficients of each target cluster and their RMSEs for the latent and sum models (Figures 5.5c and 5.5d). The Spearman rank correlation coefficient is defined as:

$$\rho = 1 - \frac{6 \sum_{i=1}^n (x_i - y_i)^2}{n(n^2 - 1)} \quad (5.16)$$

where x_i and y_i are the i th ranked values of the two variables being compared (predicted and true pKa values respectively). The Spearman rank correlation coefficient is a non-parametric measure of correlation. It is a measure of the monotonicity of the relationship between two variables. It is a rank correlation coefficient and does not assume that the relationship between the two variables is linear. It is a good measure of correlation for ordinal data. In this context, it is a measure of the ranking power of the model - that is, a measure of the model's ability to rank the binding affinities of different ligands given a specific target. Similarly to the scoring Pearson's r metric, for both models we see a majority of strong positive ranking correlations with only a few negative correlations. However, the sum model has a much more consistent ranking correlation across target clusters. The negative clusters in the latent model are clusters: 5, 18, 34, 38, 46, 48. The negative clusters in the sum model are clusters: 5, 34, 38 and 48 only. Again we see improvement between the latent model and the sum model on target clusters 18 and 46 - the same cluster 18 improved from negative to positive correlation with Pearson's r , however 48 did not with spearman's ρ as it did with Pearson's r , this time it was cluster 46. And again we see little correlation between Spearman's correlation and RMSE in both models. Across all target clusters, the latent model's mean Pearson's r coefficient was $r = 0.68$ and the sum model's mean Pearson's r coefficient was $r = 0.73$. The latent model's mean Spearman's ρ coefficient was $\rho = 0.58$ and the sum model's mean Spearman's ρ coefficient was $\rho = 0.63$. The latent model's mean RMSE was RMSE=1.41 and the sum model's mean RMSE was RMSE=1.37. This shows significant performance boosts across all three metrics with the sum MFBERT-ANI model architecture over the latent MFBERT-ANI model architecture.

Ensemble Scoring

Rather than only use the model checkpoint which performed best on the validation set during training, 100 copies of each MFBERT-ANI model are independently trained, in-parallel, across multiple NVIDIA RTX 3090 GPUs. They were all used simultaneously for inference. This

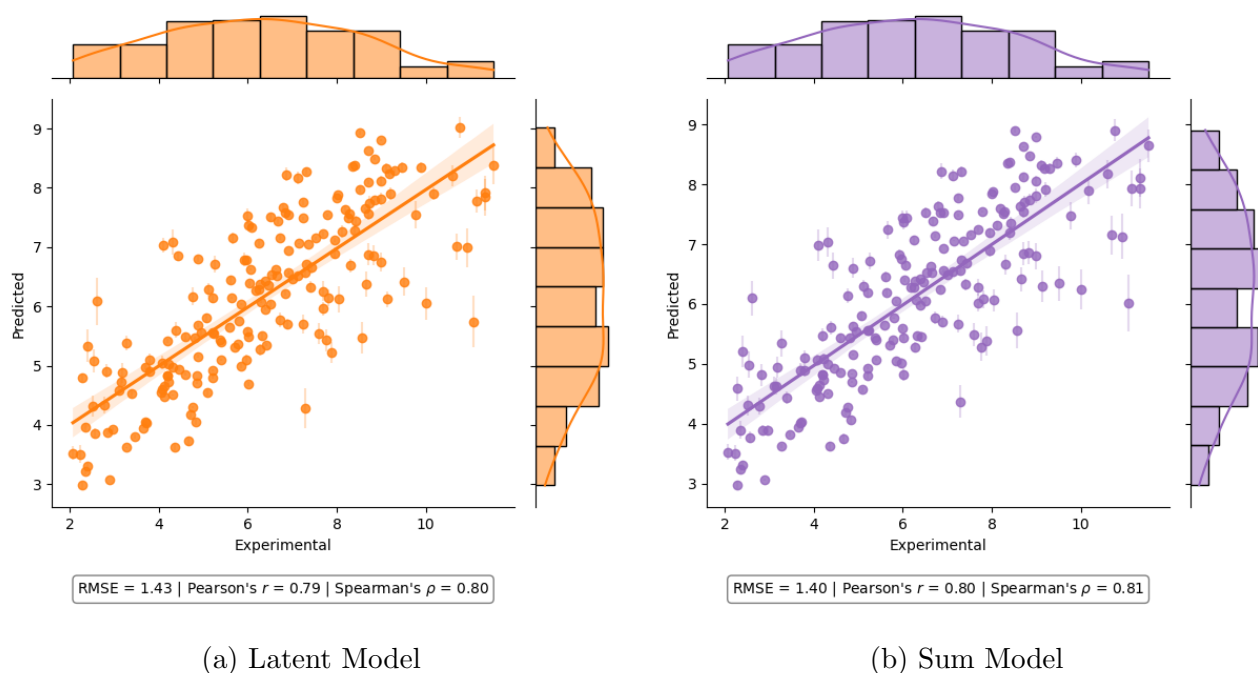


Figure 5.6: CASF-2013 Scoring Performance of both latent and sum model variants of ensembles of MFBERT-ANI. Values are given in pKa.

kind of ensembling has been reported to improve performance in literature^{38,326,333,334} and has been referred to as consensus scoring. In this case, the sentiment holds as performance does increase. Figures 5.6a and 5.6b show the scoring performance of the latent and sum MFBERT-ANI models respectively on the CASF-2013 dataset in the form of a predicted vs. true pKa values scatter plot with regression.

Comparing the ensemble model results with the single model results of both models, we see significant improvements across the three metrics with Spearman's ρ , Pearson's r and RMSE improving by 0.04 (5%), 0.05 (6%) and 0.11 (7%) respectively between the single latent model and the ensemble latent model. And by 0.03 (3%), 0.04 (5%) and 0.10 (6%) respectively between the single sum model and the ensemble sum model. Whilst the improvements are small, they are considerable considering the implications of the model being used in a real-world or CADD setting. The ensemble model is more robust and less likely to be affected by a single model's performance blunder (of which there are many, see target cluster error bars below). Looking at the error bars and 0.95 confidence intervals in the regression plots however, we can see that in general, the ensemble confidence is high when the predictions are more accurate. This is a desirable property of ensembling models in this manner as when performing inference on an

unknown PLC, the standard error across the model scores will indicate the likelihood of the model being correct.

Now let's consider the performance of the ensemble models on the CASF-2013 target clusters. Figures 5.7a, 5.7b, 5.7c and 5.7d show the Pearson correlation coefficient, the spearman rank coefficient and the RMSE for each target cluster, the error bars on each target are the mean of the standard error taken across the 100 models and over the three targets in each cluster.

Whilst there seems to be some slight improvement in the overall performance across both correlation metrics and across both model types, the target specific performance seems to be relatively similar. Looking at the sum ensemble Pearson's r , the number of negatively correlated target clusters is three, compared to the single sum model which has four. The discrepancy is within target cluster 46 which is now slightly positively correlated. It is interesting to note that the latent ensemble model's Pearson's r for target cluster 46 is still negative, and only in the sum ensemble has it switched around. This suggests that the sum model is more sensitive to this target cluster and training fluctuations than the latent model. Future work should explore the effects of various optimisers when training these models, as the large parameter count makes the error landscape significantly complex.

Looking at the error bars for each target cluster, we can see that the standard error across the 100 models is generally low, with the exception of a few (for example 1 and 45). Interestingly, some error bars are larger for particular target clusters with the sum ensemble than the latent ensemble (for example target cluster 38). It is also important to note that the error bars on some target clusters (for example clusters 1 and 38) exceed the limits of the Pearson's r correlation $[-1, 1]$. This confirms that the ensemble distribution across the multidimensions of the 100 models and each target in the cluster is not normally distributed but is rather shifted or is unstable. This also signifies the amount of variance in weights during training and indicates that whilst the model is optimised to some low error state, it is in an unstable state and is likely to be affected by training fluctuations such as time, early stopping, learning rate, etc. Ensembling models in this manner is a way to mitigate this instability and increases the repeatability of the model's performance.

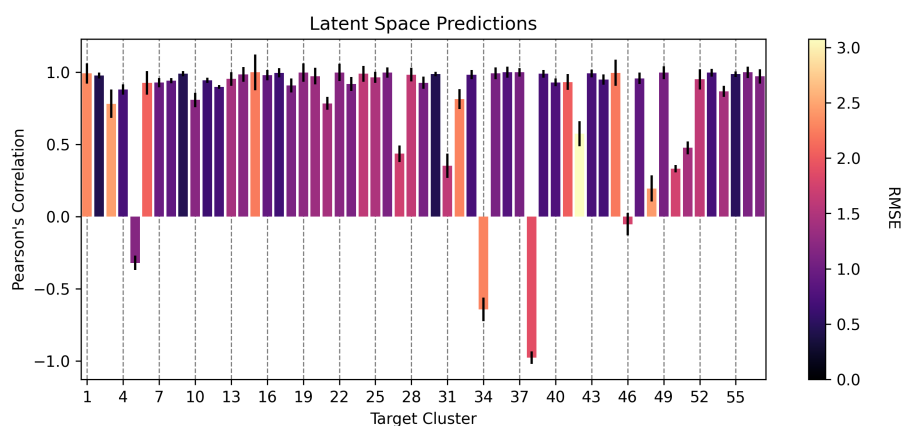
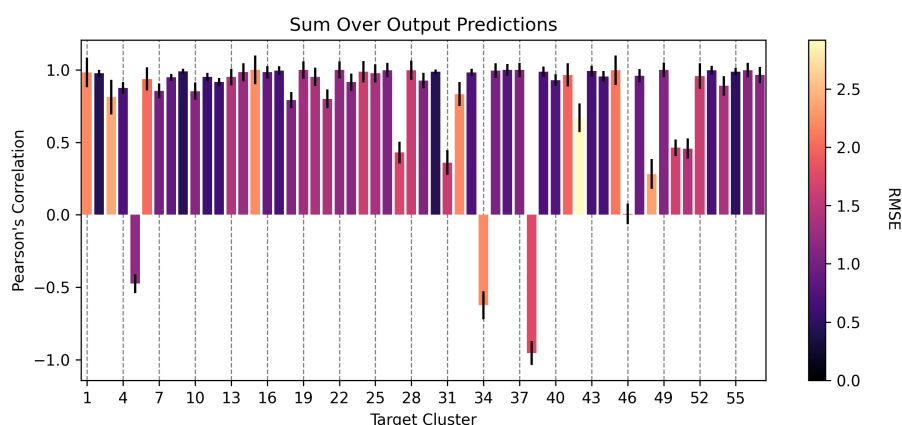
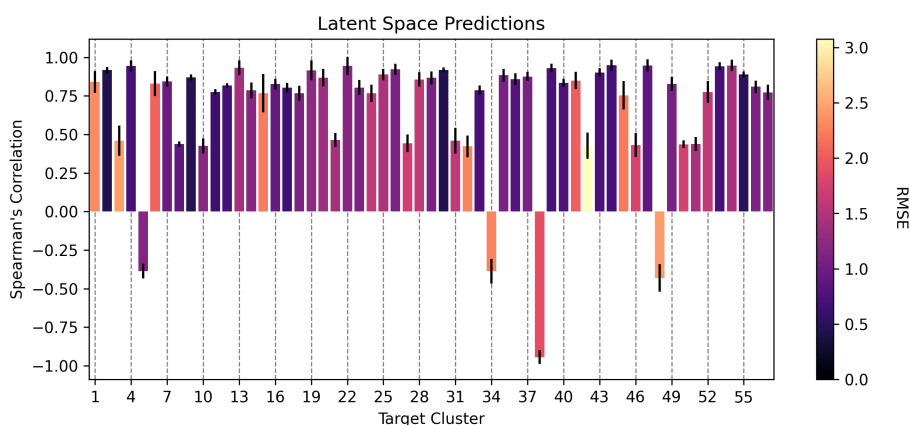
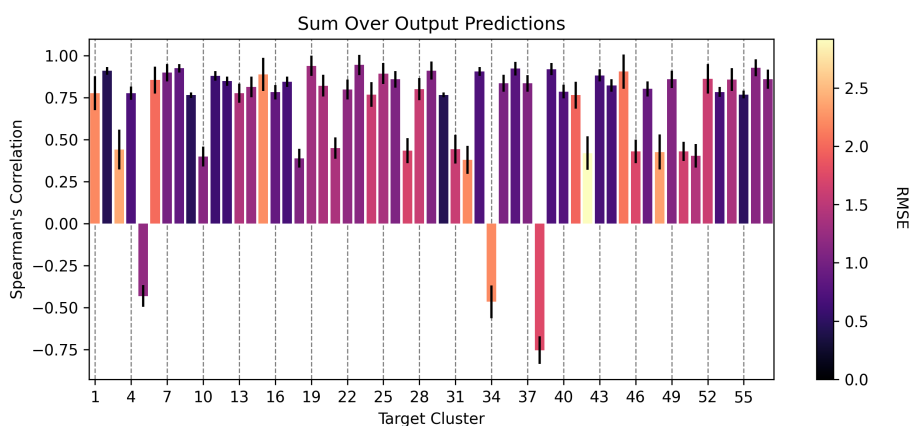
(a) Ensemble Latent Model Pearson's r (b) Ensemble Sum Model Pearson's r (c) Ensemble Latent Model Spearman's ρ (d) Ensemble Sum Model Spearman's ρ

Figure 5.7: CASF-2013 Scoring Performance on each target cluster of both latent and sum model variants of the Ensembled MFBERT-ANI. RMSE Values are given in pKa.

As for the Spearman’s rank correlation, the number of negatively correlated target clusters is four for the latent ensemble model and three for the sum ensemble model. The number of positively correlated target clusters is four for the latent ensemble model and three for the sum ensemble model. The discrepancy is this time within target cluster 48. Target cluster 48 was consistently negative in both types of the single models. The sum ensemble model improved massively with this cluster and now shows a moderate positive Spearman’s rank correlation (> 0.25). The latent ensemble model however, still shows a negative Spearman’s rank correlation, with a smaller error bar. This suggests that the latent ensemble model is less sensitive to this target cluster and training fluctuations than the sum ensemble model.

Finally, the RMSE values for the latent ensemble model are generally similar to the sum ensemble model. However, they are still similar to those achieved by the single models. Considering the scale range of the RMSE pKa values, the maximum RMSE for the ensembled latent model is larger than that of the sum model but is similar to that of the sum single model. This overall reinforces the fact that the sum model architecture performs better than the latent model architecture in CASF-2013.

CASF-2016

The CASF-2016 benchmark is based on the PDBbind 2016 refined Core set. It includes 285 PLCs out of the 290 in the PDBbind set. The discrepancy being that PLCs 3CYX, 4MSC, 4MSN, 4MRW, 4MRZ and 5C1W are in PDBbind but not CASF-2016 and that CASF-2016 also includes 1G2K which is not in the PDBbind set. The same methodology used with CASF-2013 was used with CASF-2016. Figures 5.8a and 5.8b show the regression performance of the latent and sum model variants of the MFBERT-ANI on the CASF-2016 benchmark.

The latent model variant shows a Pearson’s r correlation of 0.87 and a Spearman’s rank correlation of 0.88. The sum model variant shows a Pearson’s r correlation of 0.87 and a Spearman’s rank correlation of 0.89. The RMSE values for the latent and sum model variants are 1.31 and 1.29 respectively. Like with CASF-2013, the sum model variant performs better than the latent model variant. In both cases however, the correlations are much higher and therefore the 0.95

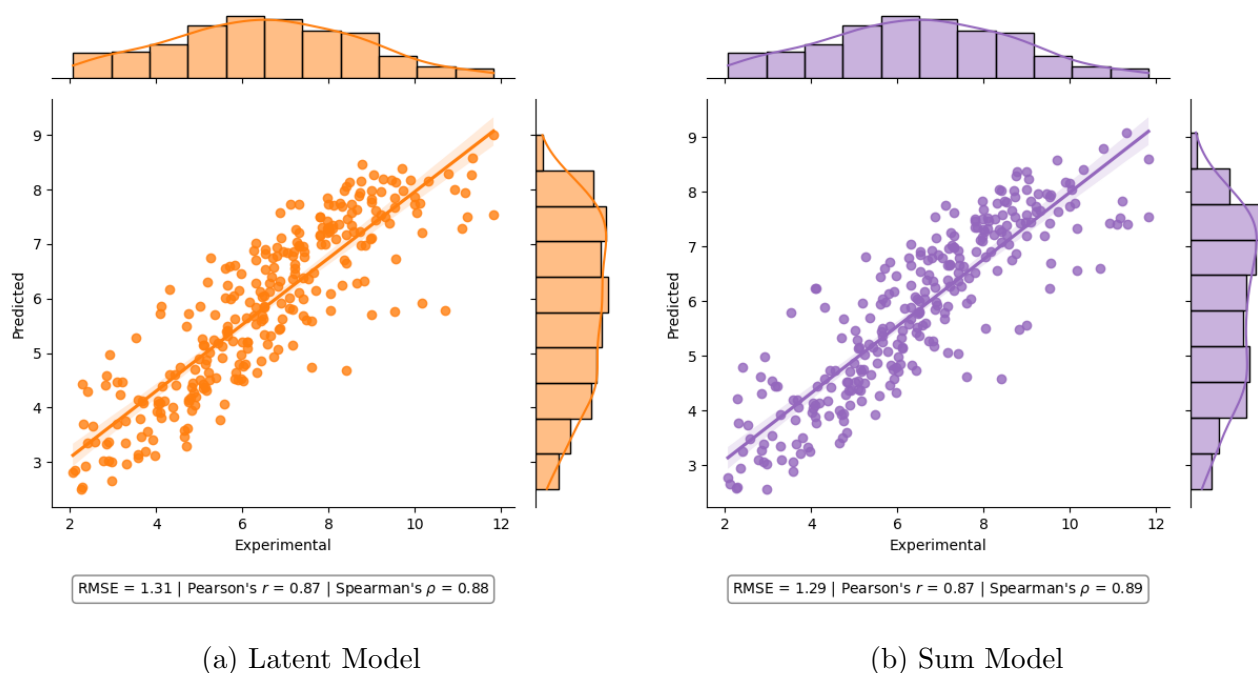


Figure 5.8: CASF-2016 Scoring Performance of both latent and sum model variants of MFBERT-ANI. Values are given in pKa.

confidence interval is much narrower as most points lie along or close-to the diagonal regression line. This is a good indication that the model is performing well and suggests that CASF-2016 is a much easier benchmark to predict than CASF-2013. This could be due to its larger size so the model has more chance to infer correctly.

Like above, the scoring and ranking performance of both model variants on the target specific clusters is assessed. Figures 5.9a, 5.9b, 5.9c and 5.9d show the Pearson correlation coefficient, the spearman rank coefficient and the RMSE for each target cluster. The latent and sum model variants of the ensembled MFBERT-ANI are shown for comparison.

Firstly, from the RMSE range it can be seen that the max RMSE for the latent model variant is 2.5 and the max RMSE for the sum model variant is 2.0. Whilst both of these values are (relatively) low compared to other models in literature, the sum model still consistently outperforms the latent model across this dataset in terms of RMSE. When considering the other two metrics however, some discrepancies begin to arise between the performances of the two variants. Consider the Pearson's r correlation plots. Both the latent and sum models have two target clusters with a negative correlation. These clusters are the same across both models

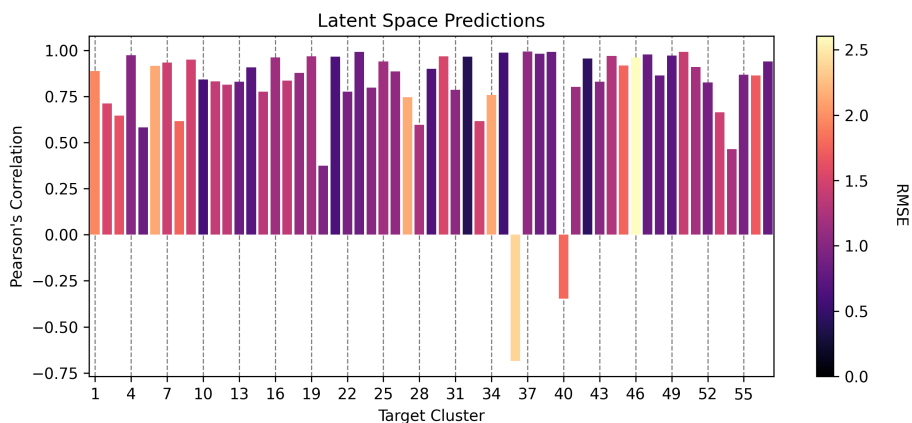
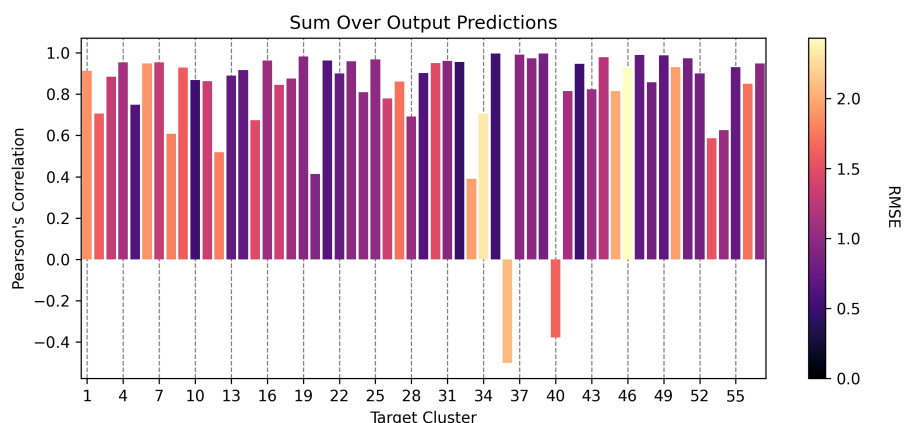
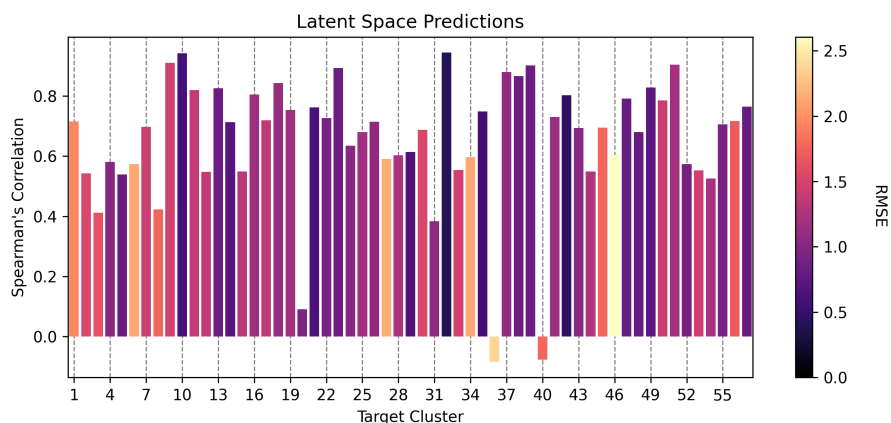
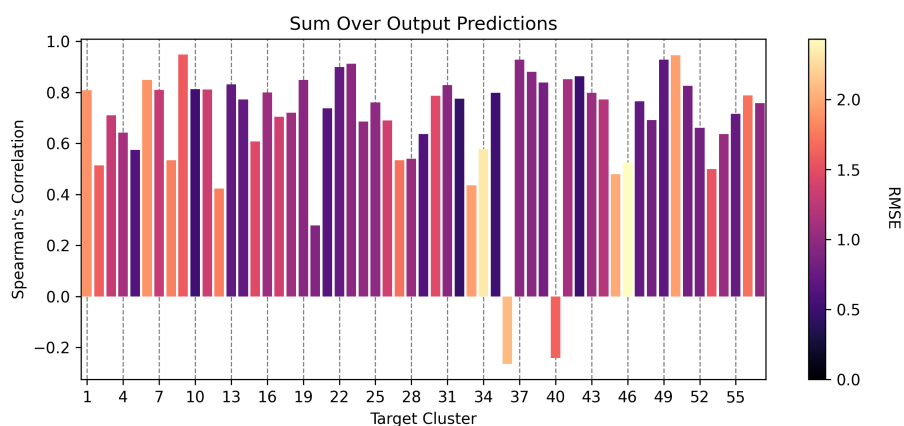
(a) Latent Model Pearson's r (b) Sum Model Pearson's r (c) Latent Model Spearman's ρ (d) Sum Model Spearman's ρ

Figure 5.9: CASF-2016 Scoring and Ranking Performance on each target cluster of both the latent and sum model variants of MFBERT-ANI. RMSE Values are given in pKa.

(target clusters 36 and 40). Target cluster 36 is O-glcnacase BT_4395 and target cluster 40 is Queuine tRNA-ribosyltransferase. The full list of targets are in the supplementary information of.²⁵⁵ While it is currently unclear why the model does not perform well on these clusters in either case, we later explore, through visualisations, the structural nature of these clusters and the reasons for the poor performance of the model on these clusters (see visualisation section below).

Finally let's consider the Spearman's rank correlation plots. The model variants also have a negative spearman's rank correlation for two target clusters 36 and 40 as above, however they are significantly less pronounced with the latent model than with the sum model. This suggests that the latent model is less likely to suffer from perturbations during training and therefore is more robust to outliers. This is a good indication that the latent model is more likely to perform well on unseen data, however it also prevents the latent model from learning and exploring the true relationship between the input and pKa. This is a trade-off that must be considered when choosing between the latent and sum model variants.

The mean values of the scoring and ranking metrics for the latent and sum model variants are computed and compared. The latent model variant has a mean Spearman's rank correlation of 0.66 and a mean Pearson's r correlation of 0.80. The sum model variant has a mean Spearman's rank correlation of 0.67 and a mean Pearson's r correlation of 0.81. The latent model variant has a mean RMSE of 1.22 and the sum model variant has a mean RMSE of 1.19. The sum model variant performs better than the latent model variant in terms of these metrics.

Ensemble Scoring CASF-2016

100 models of each variant of MFBERT-ANI for CASF-2016 are re-trained and ensembled to see if the performance of the model can be improved. The ensemble model is trained in the same way as the single model, however the training data is sampled with replacement. The ensemble model is then evaluated on the CASF-2016 test set. The ensemble model is then compared to the single model and the ensemble of the single models. The regression results are shown in figures 5.10a and 5.10b.

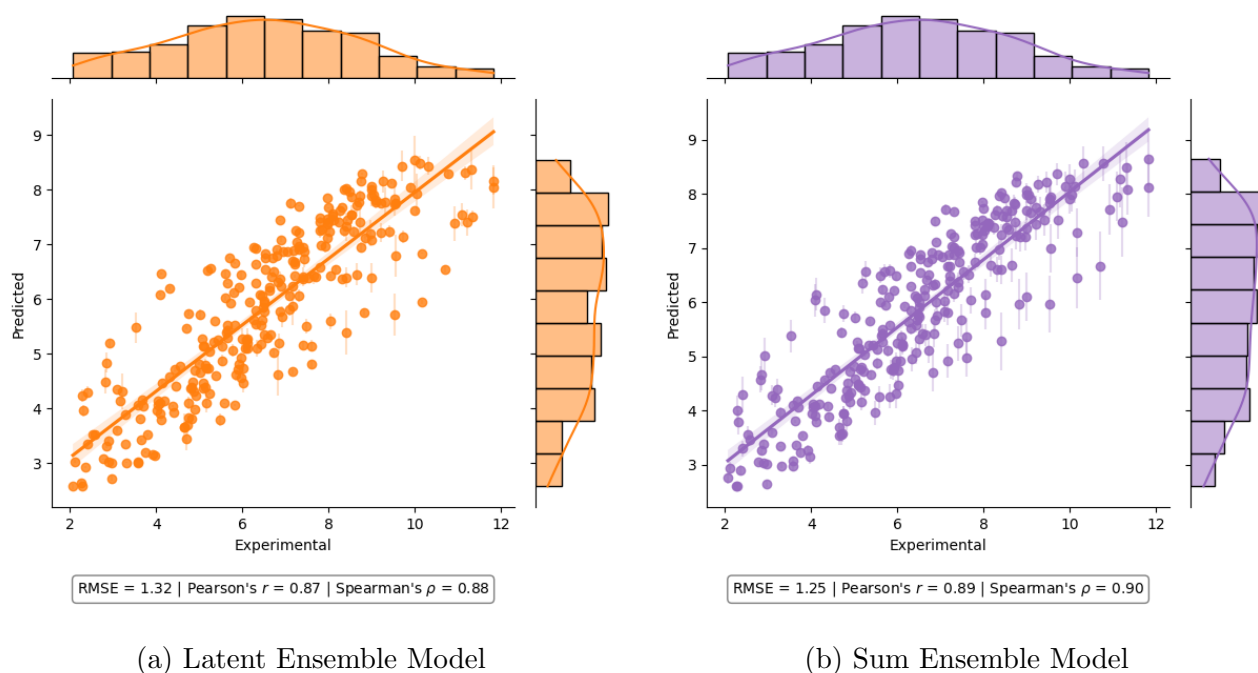


Figure 5.10: CASF-2016 Scoring Performance of both latent and sum model variants of ensembles of MFBERT-ANI. Values are given in pKa.

Interestingly, when comparing the RMSE of the latent ensemble with the latent single model and the sum ensemble with the sum single model, the latent ensemble performs worse than the latent single model but the sum ensemble performs better than the sum single model. The latent ensemble achieves an RMSE of 1.32, a Pearson's r correlation of 0.87 and a Spearman's rank correlation of 0.88. The sum ensemble achieves an RMSE of 1.25, a Pearson's r correlation of 0.89 and a Spearman's rank correlation of 0.90. The latent single model achieves an RMSE of 1.31, a Pearson's r correlation of 0.87 and a Spearman's rank correlation of 0.88. The sum single model achieves an RMSE of 1.29, a Pearson's r correlation of 0.87 and a Spearman's rank correlation of 0.89. Therefore, even though the latent ensemble performs worse on the RMSE metric than its single model counterpart, the correlation between the predicted and ground truth pKa binding affinities is consistent. In other words, the model is better at ordering the samples based on binding affinity, rather than predict the accurate pKa value. Another observation is with the error bars of the ensemble models. For the sum variant, the bars are rather large compared to the latent variant. This suggests that even though the ensemble is more accurate, its confidence in the predictions is limited and can therefore be easily perturbed by outliers or minor fluctuations within a PLC. This lack of stability/robustness is less pronounced

in the latent variant, making it a better choice for more reliable but slightly less accurate predictions.

The target cluster performance of the latent and sum ensemble models is considered. The results are shown in figures 5.11a, 5.11b, 5.11c and 5.11d. Again, like the single models, two target clusters have both negative Pearson’s r and Spearman’s rank correlations. These are the same clusters as above, 36 and 40. One thing to note, particularly with the sum ensemble’s Spearman’s ρ is that the error bars for these two clusters are huge and are larger than the mean correlation magnitude itself. This again highlights the instability and inconsistencies of the sum through this mean ensembling method. Future works should consider other ensembling methods such as meta optimised weighted ensembling. Overall, there doesn’t seem to be any significant advantage in assessing the target clusters between single models and ensembling.

The mean RMSE, Pearson’s r correlation coefficient and Spearman’s rank correlation coefficient of the latent ensemble model across all target clusters is 1.23, 0.80 and 0.67 respectively. For the sum ensemble model they are 1.17, 0.83 and 0.69 respectively. Again the sum ensemble is more accurate in this case than the latent ensemble, however the error bars of the sum model for both correlation metrics are larger than those of the latent model. Therefore, there is a trade-off between accuracy and stability.

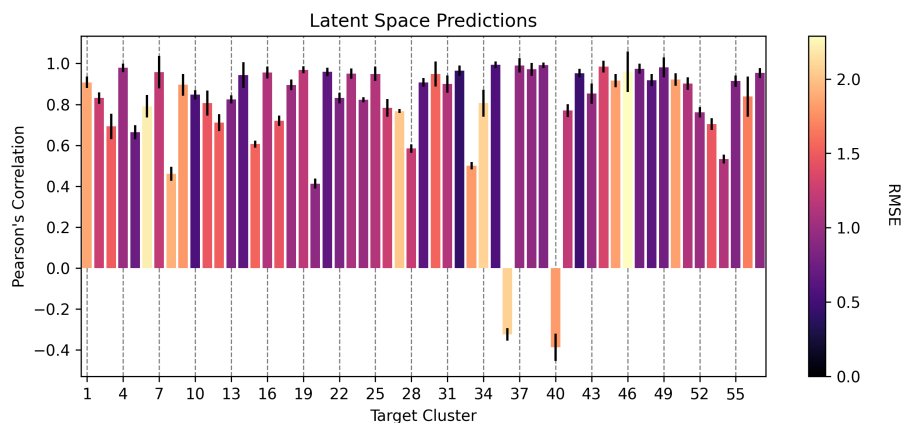
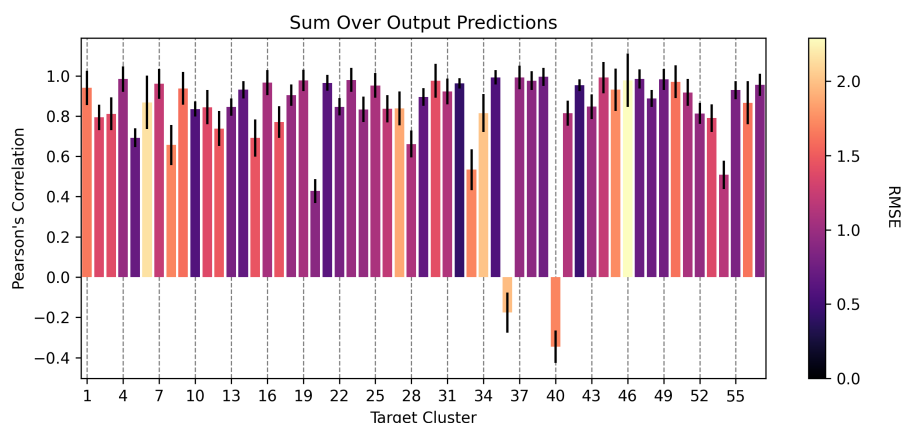
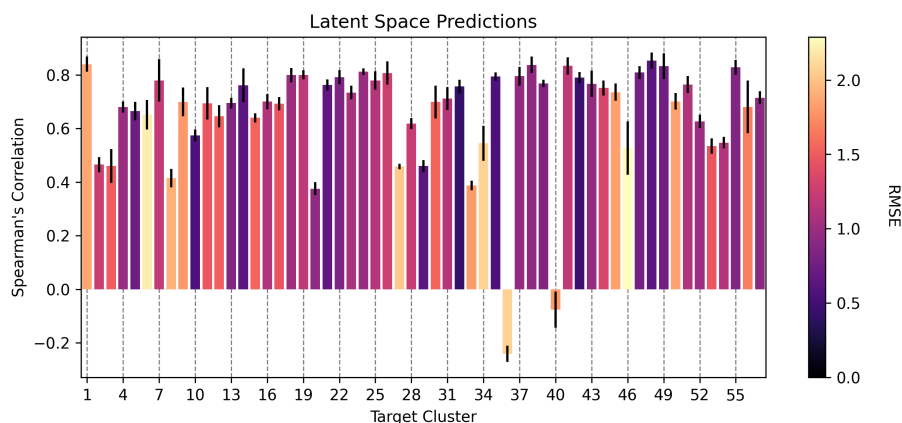
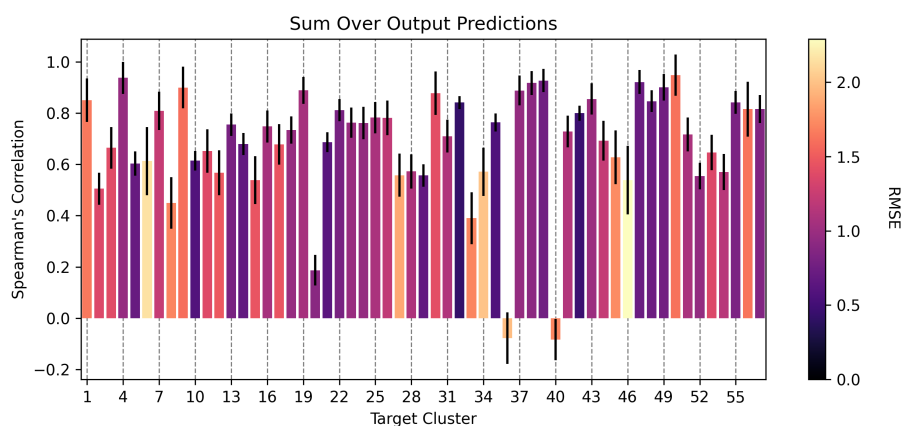
(a) Latent Model Pearson's r (b) Sum Model Pearson's r (c) Latent Model Spearman's ρ (d) Sum Model Spearman's ρ

Figure 5.11: CASF-2016 Scoring and Ranking Performance on each target cluster of both the latent and sum ensemble model variants of MFBERT-ANI. RMSE Values are given in pKa.

5.3.5 Visualisation

To further get an intuition of the model's performance, visualisations to see the regression scatter plot with the PLC interactively were viewed. A web app was built with bokeh¹⁹³ and NGL³³⁵ to allow for this. The web app allows the user to select a point from the regression plot. Once the user clicks a point it becomes highlighted and the relevant 3D PDB that the point relates to, as well as the corresponding PDB ID, experimental pKa and the model's predicted value are shown. The user can then navigate the 3D structure by click-dragging the mouse to rotate the structure and scrolling to zoom in and out. The user can also select a different point from the regression plot to view the corresponding 3D structure. The web app is shown below.

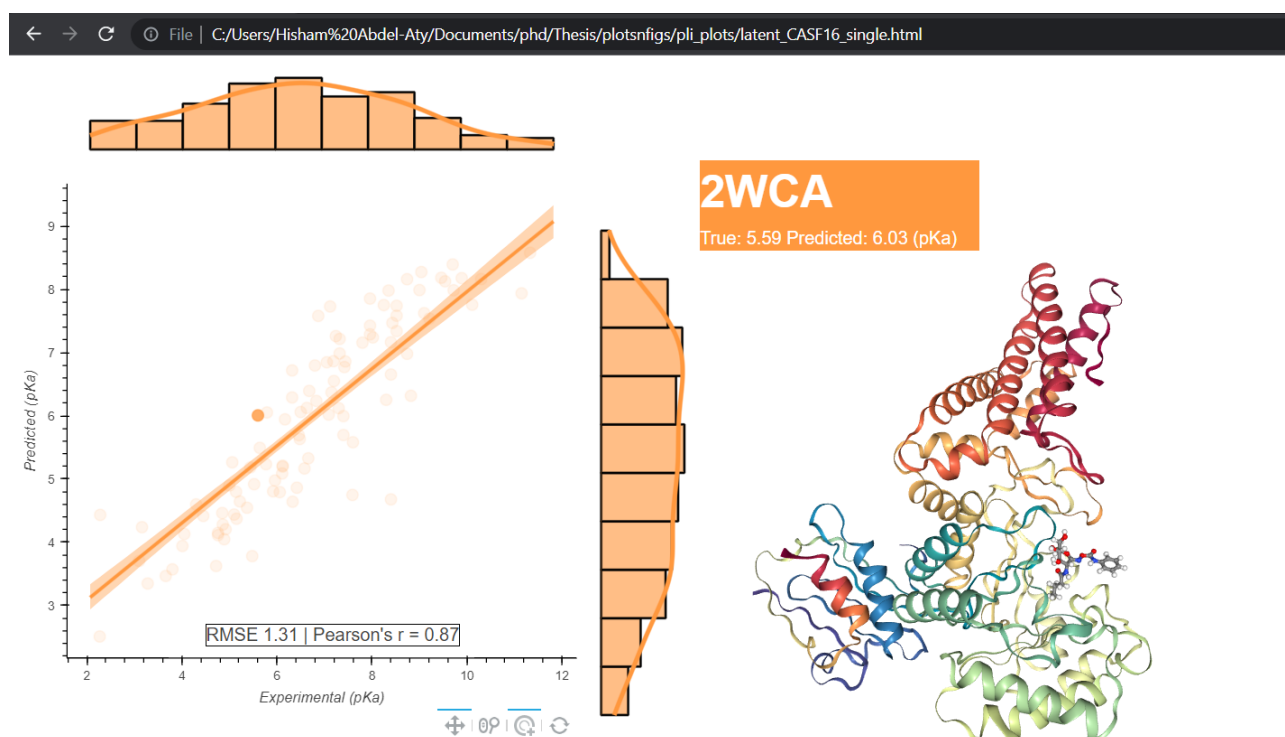


Figure 5.12: Web App for visualising the regression plot and the corresponding 3D structure. This screenshot shows the selected point in the regression plot for the single latent model and the corresponding 3D structure. The point represents PDB ID: 2WCA which is a member of cluster 36, where there was a negative Pearson's r and Spearman's rank correlation.

Considering this structure, we see that it is a quaternary complex of many subunits. The subunits are somewhat similar in structure and are all very close to each other. Within the binding pocket, the ligand is surrounded by mainly beta-sheets, however the rest of the structure

is mainly alpha-helices. Given that the model only considers the binding pocket, and the rest of the structure is not considered, this is not a problem. The ligand is also not in a very favourable position for binding, as it is not in the centre of the binding pocket, but rather on the edge. This is likely the reason for the negative correlation, as further visualisations of other targets in this cluster show similar properties - a beta-sheet based pocket and an alpha-helix based rest of protein with side-like ligand binding of a large ligand. Figure 5.13 shows another example of a structure in target cluster 40.

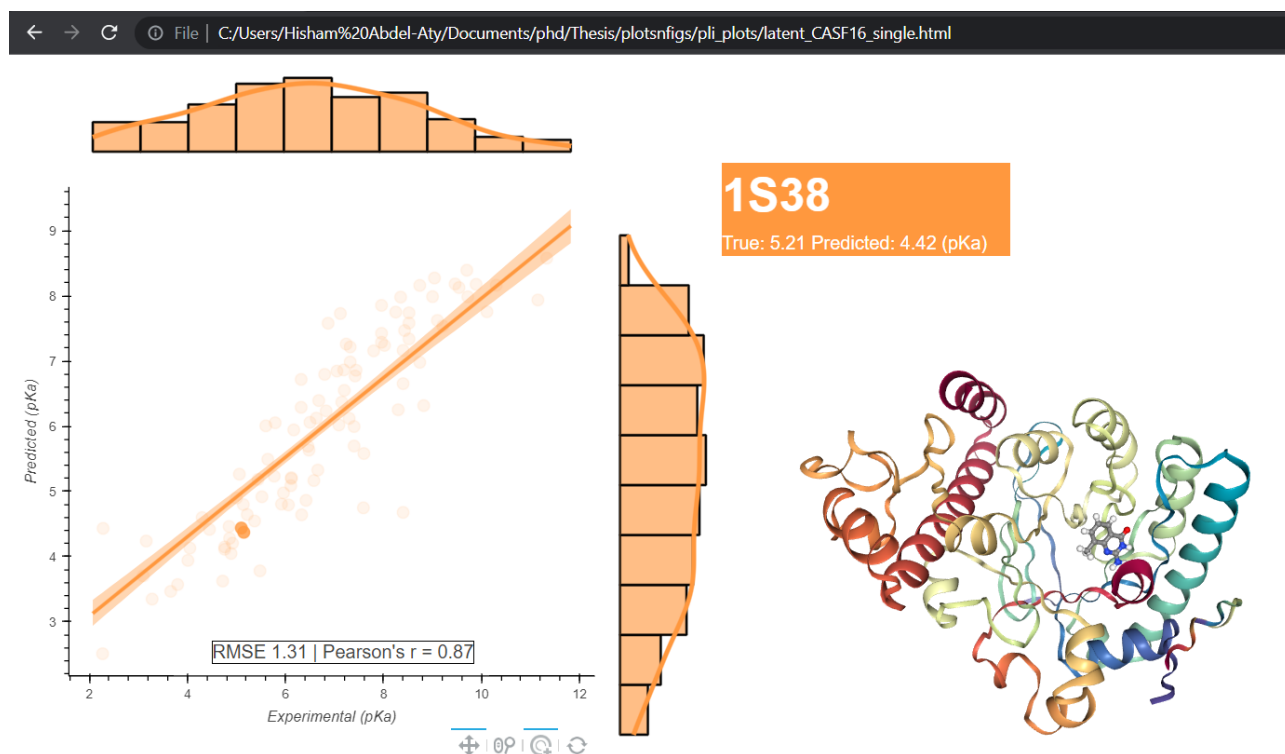


Figure 5.13: Web App for visualising the regression plot and the corresponding 3D structure. This screenshot shows the selected point in the regression plot for the single latent model and the corresponding 3D structure. The point represents PDB ID: 1S38 which is a member of cluster 40, where there was a negative Pearson's r and Spearman's rank correlation.

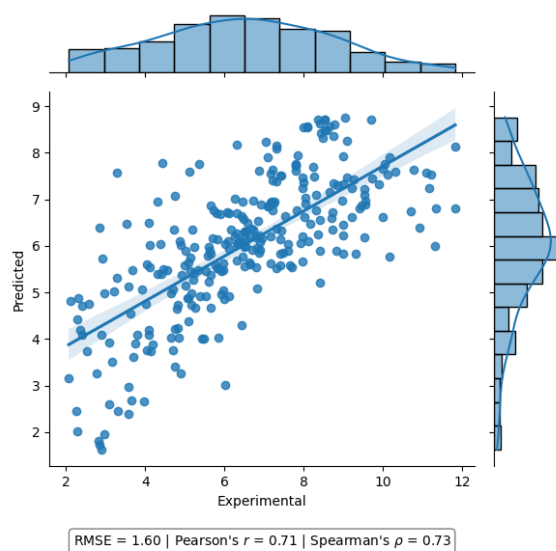
In this case, the protein is a tertiary structure with an alpha-helix based binding pocket and a ligand that snugly fits inside, unlike the side-like binding of 36. The ligand is also very small. The MAE of this prediction is 0.79 compared with 0.44 for 2WCA. More specifically, 2WCA is an exo-N-acetyl-beta-glucosaminidase, a hydrolytic enzyme found within Gram-negative bacteria and 1S38 is tRNA-Guanine Transglycosylase - an enzyme which modifies transfer RNA (tRNA). This suggests that the model is able to somewhat predict the pKa values of these

functionally different targets, albeit inconsistently across the series. The web app is a powerful way to quickly view the model results. It is portable, static and can be hosted on any webserver with no back-end required and is simply generated with a python script.

5.3.6 Ligand-only Prediction

As mentioned previously, most AI models for binding affinity predictions are able to perform well with just the ligand information as ligand features are sufficiently effective, in most cases, for this task.³³⁶ However in some cases, this bias a model may develop towards ligand features may hinder its performance.^{318,327} Therefore, it is interesting to also test the model's performance when only the ligand information is used as input. That is, taking MFBERT and fine-tuning it on the PDBbind ligands to predict pKa. After training for 100 epochs with a batch size of 32 and a learning rate of 1e-5 for the Adam optimiser, the model is evaluated on the CASF-2013 and CASF-2016 sets similarly to above with MFBERT-ANI. The regression, Pearson's r and Spearman's rank correlation coefficient plots for CASF-2016 are shown in Figures 5.14a, 5.14b and 5.14c respectively.

The ligand-only model performs (relatively) well on the benchmark. However it is distinctively worse than both variants of the full MFBERT-ANI model. Overall, the model achieves an RMSE of 1.60, a Pearson's r of 0.71 and a Spearman's rank correlation coefficient of 0.73. The 0.95 confidence interval of the regression is considerably larger than that of the full model, which suggests that the model is not as confident in its predictions. This is likely due to the fact that the model is not able to use the protein information to help it make more accurate predictions. Looking at the target cluster performance, we see a wide variety of correlation coefficients across the targets, three of which have a negative Pearson's r (8, 12, 43) and four have a negative Spearman's rank correlation coefficient (8, 12, 41, 43). Interestingly, targets 36 and 40 which had a negative Pearson's r correlation with the full model have a positive Pearson's r correlation with the ligand-only model. The RMSE values across the target clusters are generally higher when compared with the full model. The observation that a model is able to learn the pKa values purely from ligand features has been observed previously.³³⁶ This is not surprising as the



(a) Ligand-only Regression Plot

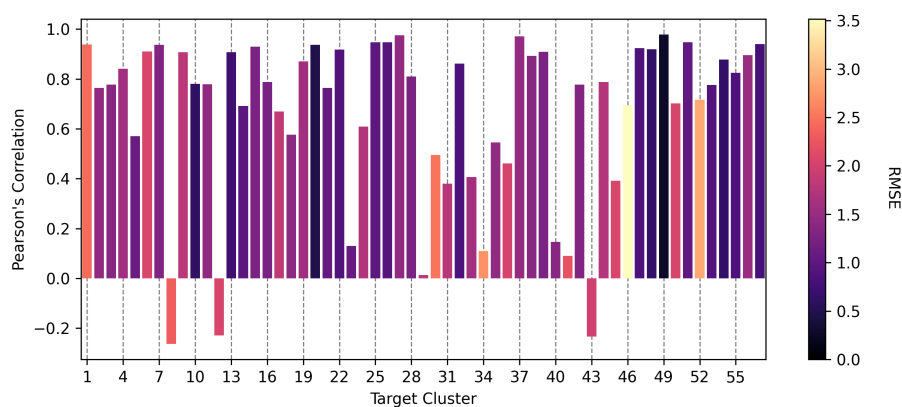
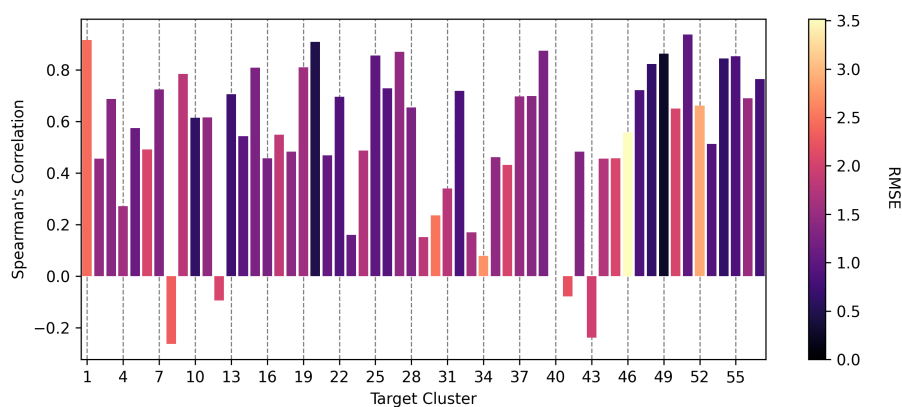
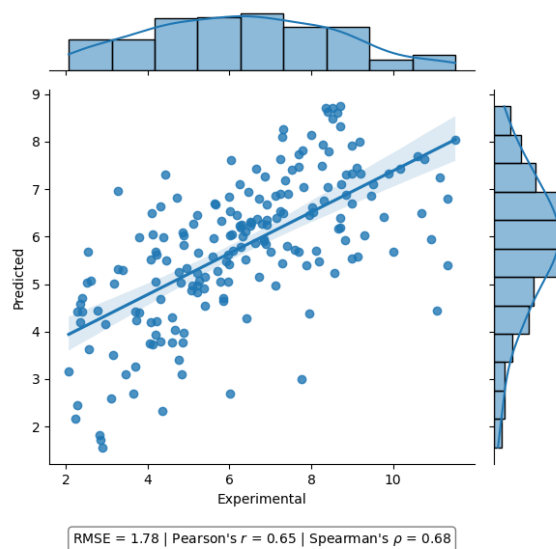
(b) Ligand-only Pearson's r (c) Ligand-only Spearman's ρ

Figure 5.14: CASF-2016 Scoring and Ranking Performance overall and per target cluster of the MFBERT ligand-only model. RMSE Values are given in pKa.

ligand features are very effective for this task. But this introduces bias within ML models during training as the model becomes optimised to attend to ligand features,^{318,322,327} which then leads to failure cases where protein features are important for the prediction. The fact that MFBERT alone is significantly worse than MFBERT-ANI indicates that this ligand-dependence bias has been mitigated in the model. This is confirmed by the fact that the protein information is used to help the model make more accurate predictions. Further evidence can be gathered by looking at the CASF-2013 results of the ligand-only model. The results are shown in Figures 5.15a, 5.15b and 5.15c.

The ligand-only performance on CASF-2013 is not as good with an overall RMSE of 1.78, a Pearson's r correlation of 0.65 and a Spearman's ρ of 0.67. The 0.95 confidence interval of the regression is also large here, however as with all the previous models, the prediction distribution is almost as normally distributed as the experimental. The target cluster performance is also not as good as the full model. The Pearson's r correlation is negative for 3 targets (8, 12, 43) and the Spearman's ρ is negative for 4 targets (8, 12, 41, 43). The RMSE values are also high. This confirms that protein information is being used by the full model to make predictions and that the model is not biased towards the ligand. Further work could investigate the gradient flow within the model to further understand the feature abstraction and attention mechanisms. This could help to understand why the ligand-only model is not as good as the full model (explainable AI²⁰).



(a) Ligand-only Regression Plot

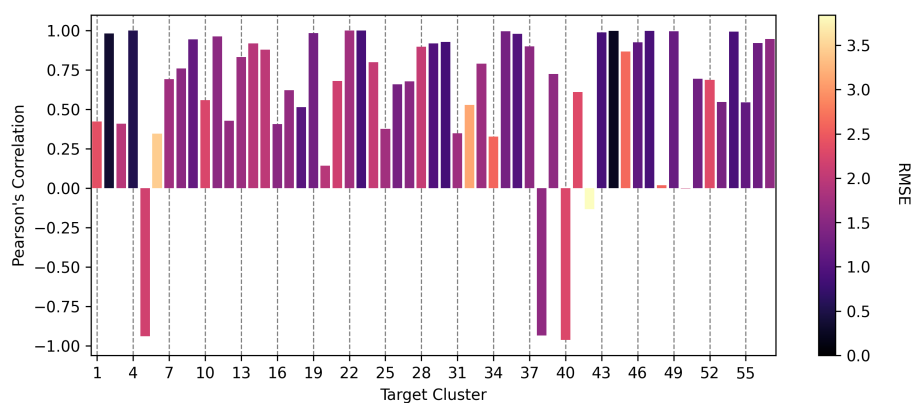
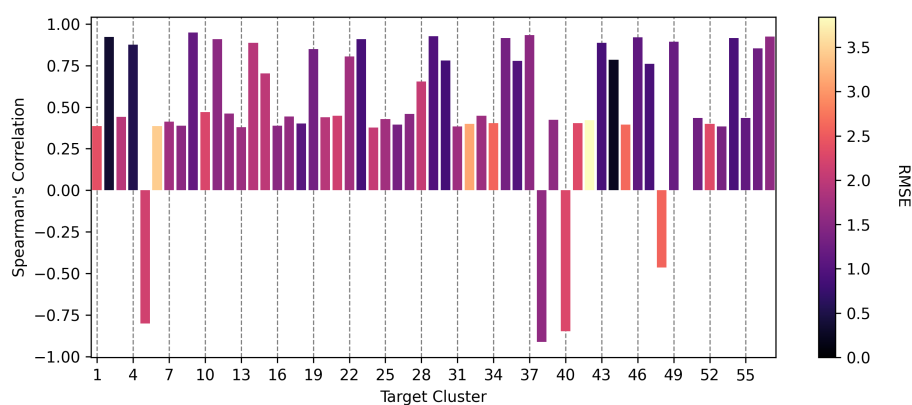
(b) Ligand-only Pearson's r (c) Ligand-only Spearman's ρ

Figure 5.15: CASF-2013 Scoring and Ranking Performance overall and per target cluster of the MFBERT ligand-only model. RMSE Values are given in pKa.

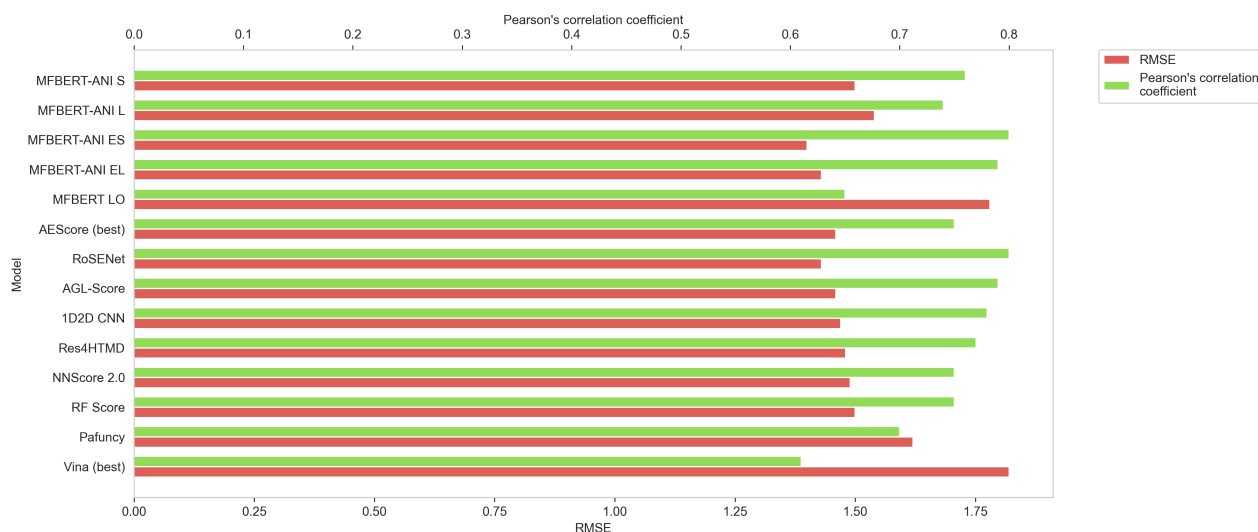
5.3.7 Comparison with other models

The performance of the MFBERT-ANI and MFBERT-ligand-only series of models is compared with other state-of-the-art models from literature across both CASF-2013 and CASF-2016 test sets. All these models were trained on either the PDBbind 2016 refined, or general sets, whereas MFBERT-ANI was trained on the PDBbind 2020 filtered refined set which removes hard and soft overlaps. Figures 5.16a and 5.16b show the performance of the MFBERT-ANI series and MFBERT-ligand-only models compared to other models from literature. The models are compared on the RMSE and Pearson’s correlation coefficient metrics.

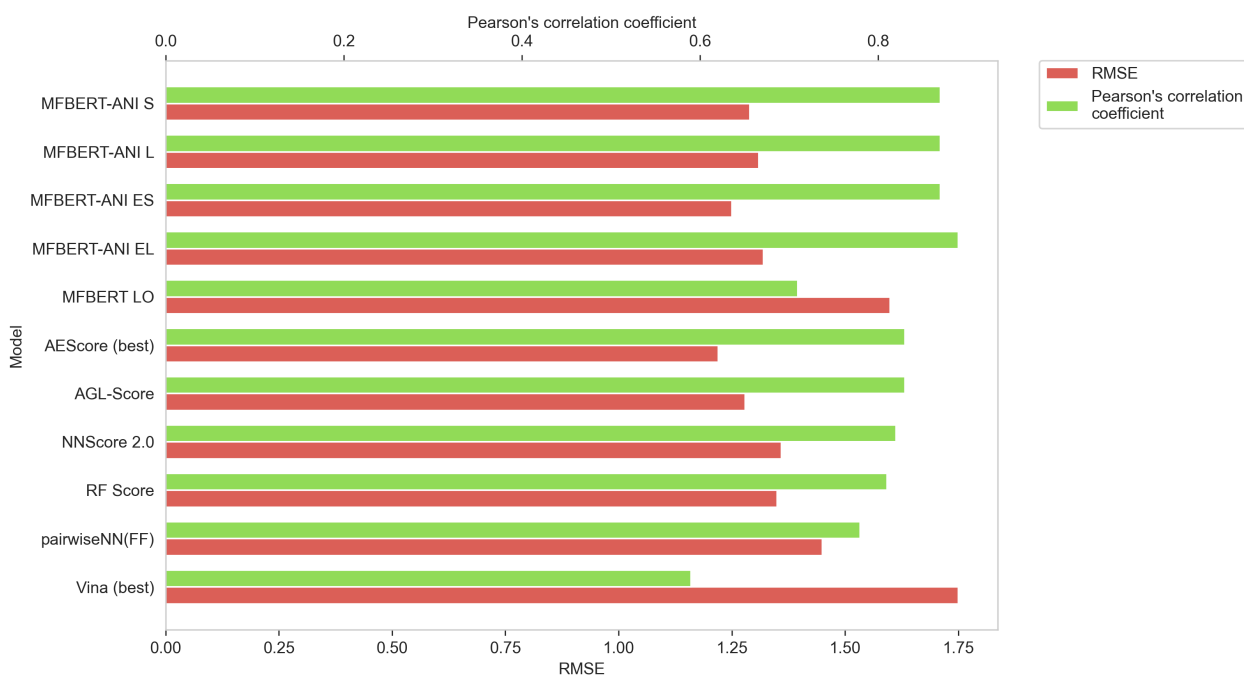
As can be seen from the figures, the MFBERT-ANI series of models score competitively with other state-of-the-art models on both benchmarks. Considering CASF-2013, whilst none of the models from the MFBERT-ANI series outperform the best model in literature currently - RosENet,²⁶⁵ MFBERT-ANI ES is on par with RosENet’s performance both in terms of RMSE and Pearson’s correlation coefficient. RosENet is a CNN based DL model that uses an ensemble of 3D convolutional layers to extract features from the protein-ligand complex through combination of voxelised MM energies and molecular descriptors.

Considering the CASF-2016 scores (figure 5.16b), AEScore³⁸ performs best in terms of RMSE, however the MFBERT-ANI series outperforms all others on correlation coefficient. AEScore, is a purely AEV based FFNN where both protein and ligand atoms are passed through an ensemble of FFNN by atom type - like ANI. The model’s performance indicates that whilst these metrics can be used as a guide for model selection for a CADD researcher, they can be inconsistent in describing the true performance of the model, and given that the fluctuations in scores is minimal across all the models, it is likely that the models are all performing similarly well.

The main differences between these models however, is the approach they take towards the problem and their input requirements. This not only determines their use case, but their flexibility when fitting into already existing CADD pipelines. One of the main advantages of MFBERT-ANI’s approach is that it’s fast and only requires the ligand’s SMILES. This



(a) CASF-2013 Dataset



(b) CASF-2016 Dataset

Figure 5.16: Comparison of the MFBERT-ANI and MFBERT-ligand-only models with other state-of-the-art models from literature on the CASF-2013 and CASF-2016 benchmarks. The models are denoted as follows: the Sum only model as MFBERT-ANI S, the latent model as MFBERT-ANI L, the ligand-only model as MFBERT-ANI LO, and the ensemble models as MFBERT-ANI ES and MFBERT-ANI EL for the sum and latent variants respectively. The models used for comparison can be found here^{263,265,266,268,269,324,326,336–342}

eliminates the need for any docking during a virtual screening process where you try to find a binder for a specific target from a large database of (usually SMILES of) ligands. Not only that but using the AEV of the binding pocket means only the atom type and coordinates of each atom are necessary. This is in contrast to other models in literature which also rely on MM FFs and their parameterisations - this usually limits these models based on the limits of Newtonian mechanics. Another advantage of using AEV as the representation is, compared to other models (CNN based or otherwise) where a Euclidian co-ordinate space is used, AEV is translationally and rotationally invariant, whereas with other models, rotating or moving the whole PLC may give different results.

As a sanity check, standard docking scoring is performed to compare its results with our models. The AutoDock Vina scoring function²⁶¹ implemented in smina³⁴³ was used. The results are shown in figures 5.16a and 5.16b. As expected, all ML/DL models outperform this baseline, which indicates the limitations of classical scoring functions.

5.3.8 Conclusion

In this chapter, a translationally and rotationally invariant combination model of MFBERT and ANI, MFBERT-ANI was developed and tested. It is able to predict absolute protein-ligand binding affinities without ligand co-ordinates (only SMILES) and with protein co-ordinates. No other external parameterisations (such as force-fields) are needed. Utilising atomic environment vectors from the Behler-Parrinello representations, and molecular fingerprints from MFBERT, the model is able to score competitively with other state-of-the-art models in less computational and human time. The lack of need for ligand co-ordinates eliminates the need for docking as a precursor step for binding affinity prediction in a CADD pipeline. It also somewhat reduces the bias that is usually common with these types of models where only the ligand atom features are used for prediction.

Chapter 6

Conclusion

The current landscape and intersections across the fields of machine learning, computational chemistry, and biology were explored in this thesis. It was shown that a variety of ML algorithms can be used to learn representations of the chemical space, and that these representations can be used to perform a variety of tasks. It was also shown that these algorithms can be used across a variety of areas of chemical research, from accelerating experimental workflows to predicting the properties of new molecules and their interactions with proteins.

In order to explore and assess the applications and utility of these methods, new machine learning pipelines and systems were developed. The main developments and contributions of this thesis are as follows:

- A fast-inference machine learning pipeline and friendly user interface to allow experimentalists to quickly analyse their lipid SAXS spectra such that they can make decisions on their experiments on site.
- A novel method to learn representations of the chemical space from large databases using a combination of distributed pre-training, transfer learning and the transformer architecture - and use these representations to generate contextual molecular fingerprints.

- Assessing combinations of molecular representations to further improve the performance of biological system representations such as protein-ligand complexes and using the combined representation to predict binding affinity.

The results within this thesis show that whilst these algorithms perform relatively well across a variety of tasks, there are significant caveats to their performance and success. These caveats are not only limited to the high computational resource requirements needed, but also apply to the data, the inference samples and the task itself. Comparative assessments were performed with other models within the field for each task, however, benchmarking such models, given the limited understanding of the reasoning for a model's predictions, is still an open problem. This is especially true for the biological tasks, where the lack of a gold standard for the task with the same train and test data makes it difficult to assess the performance of the models. As such, benchmark performances are not usually representative of the true performance of the models, and the results of these benchmarks should be considered with the bigger picture of the model's training and testing data in mind.

6.0.1 Future Work

Given the work within this thesis (and generally within the field). We now have powerful models that can represent molecules effectively across many contexts. However, there are still many challenges that need to be addressed before these models can be used in a practical setting. The two main challenges that should be addressed are the following:

- **Interpretability:** The current models are black-box models, it is therefore difficult to understand why a model makes a certain prediction. This is especially true for tasks that are difficult to benchmark (i.e. those where ground truth values are inconsistent, such as biological experimental results).
- **Integration** The current models are not integrated with existing research and development workflows, as such, they are not used in a practical setting. One of the most

important future directions is integration as it will not only allow the models to be used to accelerate research, but feedback from larger scale use will be invaluable for improving the models and further understanding their predictions and limitations. Integration will also mean that better visualisations of the models' predictions and internal workings must be developed, which will allow for better interpretability.

With the bigger picture in mind, given the current available models and the challenges that need to be addressed, it is clear that there is still a lot of work to be done, especially with the larger high-performance deep learning models. However, the results of this thesis show that we now have molecular representations and models that are already powerful enough to be used in a practical setting, and that they can be used to accelerate research and development in a variety of areas.

Bibliography

- (1) Krizhevsky, A.; Sutskever, I.; Hinton, G. E. In *Advances in Neural Information Processing Systems*, ed. by Pereira, F.; Burges, C.; Bottou, L.; Weinberger, K., Curran Associates, Inc.: 2012; Vol. 25, <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.
- (2) Brown, T. B. et al. Language Models Are Few-Shot Learners, 2020, arXiv: 2005.14165 [cs], <http://arxiv.org/abs/2005.14165> (accessed 03/04/2023).
- (3) Jumper, J. et al. Highly Accurate Protein Structure Prediction with AlphaFold. *Nature* **2021**, *596*, 583–589, <https://www.nature.com/articles/s41586-021-03819-2> (accessed 06/17/2022).
- (4) Tan, M.; Le, Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, 2020, arXiv: 1905.11946 [cs, stat], <http://arxiv.org/abs/1905.11946> (accessed 03/03/2023).
- (5) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need, 2017, arXiv: 1706.03762 [cs], <http://arxiv.org/abs/1706.03762> (accessed 03/04/2023).
- (6) Schwaller, P.; Laino, T.; Gaudin, T.; Bolgar, P.; Hunter, C. A.; Bekas, C.; Lee, A. A. Molecular Transformer: A Model for Uncertainty-Calibrated Chemical Reaction Prediction. *ACS Central Science* **2019**, *5*, 1572–1583, <https://doi.org/10.1021/acscentsci.9b00576> (accessed 03/04/2023).

- (7) Choudhary, K.; DeCost, B.; Chen, C.; Jain, A.; Tavazza, F.; Cohn, R.; Park, C. W.; Choudhary, A.; Agrawal, A.; Billinge, S. J. L.; Holm, E.; Ong, S. P.; Wolverton, C. Recent Advances and Applications of Deep Learning Methods in Materials Science. *npj Computational Materials* **2022**, *8*, 1–26, <https://www.nature.com/articles/s41524-022-00734-6> (accessed 03/07/2023).
- (8) Virtanen, P. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **2020**, *17*, 261–272.
- (9) Harris, C. R. et al. Array Programming with NumPy. *Nature* **2020**, *585*, 357–362, <https://www.nature.com/articles/s41586-020-2649-2> (accessed 03/03/2023).
- (10) Kudo, T.; Richardson, J. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing, 2018, arXiv: 1808.06226 [cs], <http://arxiv.org/abs/1808.06226> (accessed 03/04/2023).
- (11) Ott, M.; Edunov, S.; Baevski, A.; Fan, A.; Gross, S.; Ng, N.; Grangier, D.; Auli, M. Fairseq: A Fast, Extensible Toolkit for Sequence Modeling, 2019, arXiv: 1904.01038 [cs], <http://arxiv.org/abs/1904.01038> (accessed 07/23/2023).
- (12) Paszke, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library, 2019, arXiv: 1912.01703 [cs, stat], <http://arxiv.org/abs/1912.01703> (accessed 07/23/2023).
- (13) van der Walt, S.; Schönberger, J. L.; Nunez-Iglesias, J.; Boulogne, F.; Warner, J. D.; Yager, N.; Goullart, E.; Yu, T.; scikit-image contributors, t. Scikit-Image: Image Processing in Python. *PeerJ* **2014**, *2*, e453, <https://doi.org/10.7717/peerj.453>.
- (14) Pedregosa, F. et al. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.
- (15) Pandey, M.; Fernandez, M.; Gentile, F.; Isayev, O.; Tropsha, A.; Stern, A. C.; Cherkasov, A. The Transformational Role of GPU Computing and Deep Learning in Drug Discovery. *Nature Machine Intelligence* **2022**, *4*, 211–221, <https://www.nature.com/articles/s42256-022-00463-x> (accessed 03/07/2023).

- (16) Guest, D.; Cranmer, K.; Whiteson, D. Deep Learning and Its Application to LHC Physics. *Annual Review of Nuclear and Particle Science* **2018**, *68*, 161–181, <https://doi.org/10.1146/annurev-nucl-101917-021019> (accessed 03/07/2023).
- (17) Webb, S. Deep Learning for Biology. *Nature* **2018**, *554*, 555–557, <https://www.nature.com/articles/d41586-018-02174-z> (accessed 03/07/2023).
- (18) Ramsundar, B.; Eastman, P.; Walters, P.; Pande, V.; Leswing, K.; Wu, Z., *Deep Learning for the Life Sciences*; O'Reilly Media: 2019.
- (19) Thompson, N. C.; Greenewald, K.; Lee, K.; Manso, G. F. The Computational Limits of Deep Learning, 2022, arXiv: 2007.05558 [cs, stat], <http://arxiv.org/abs/2007.05558> (accessed 03/07/2023).
- (20) Gohel, P.; Singh, P.; Mohanty, M. Explainable AI: Current Status and Future Directions, 2021, arXiv: 2107.07045 [cs], <http://arxiv.org/abs/2107.07045> (accessed 03/07/2023).
- (21) Bowman, S. R.; Angeli, G.; Potts, C.; Manning, C. D. A Large Annotated Corpus for Learning Natural Language Inference, 2015, arXiv: 1508.05326 [cs], <http://arxiv.org/abs/1508.05326> (accessed 03/07/2023).
- (22) Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; Fei-Fei, L. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp 248–255.
- (23) Martinc, M.; Pollak, S.; Robnik-Šikonja, M. Supervised and Unsupervised Neural Approaches to Text Readability. *Computational Linguistics* **2021**, *47*, 141–179, arXiv: 1907.11779 [cs], <http://arxiv.org/abs/1907.11779> (accessed 03/07/2023).
- (24) Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.-J. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics: Philadelphia, Pennsylvania, USA, 2002, pp 311–318, <https://aclanthology.org/P02-1040>.
- (25) Rajpurkar, P.; Zhang, J.; Lopyrev, K.; Liang, P. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Association for Computational

- Linguistics: Austin, Texas, 2016, pp 2383–2392, <https://aclanthology.org/D16-1264>.
- (26) Rajpurkar, P.; Jia, R.; Liang, P. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Association for Computational Linguistics: Melbourne, Australia, 2018, pp 784–789, <https://aclanthology.org/P18-2124>.
- (27) Reddy, S.; Chen, D.; Manning, C. D. CoQA: A Conversational Question Answering Challenge, 2019, arXiv: 1808.07042 [cs], <http://arxiv.org/abs/1808.07042> (accessed 03/07/2023).
- (28) Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge, 2015, arXiv: 1409.0575 [cs], <http://arxiv.org/abs/1409.0575> (accessed 03/07/2023).
- (29) Wu, Z.; Ramsundar, B.; Feinberg, E. N.; Gomes, J.; Geniesse, C.; Pappu, A. S.; Leswing, K.; Pande, V. MoleculeNet: A Benchmark for Molecular Machine Learning. *Chemical Science* **2018**, *9*, 513–530, <https://pubs.rsc.org/en/content/articlelanding/2018/sc/c7sc02664a> (accessed 03/04/2023).
- (30) Fei, N.; Lu, Z.; Gao, Y.; Yang, G.; Huo, Y.; Wen, J.; Lu, H.; Song, R.; Gao, X.; Xiang, T.; Sun, H.; Wen, J.-R. Towards Artificial General Intelligence via a Multimodal Foundation Model. *Nature Communications* **2022**, *13*, 3094, <https://www.nature.com/articles/s41467-022-30761-2> (accessed 03/07/2023).
- (31) Goodfellow, I.; Bengio, Y.; Courville, A., *Deep Learning*; MIT Press: 2016.
- (32) Gao, K.; Sener, O. Modeling and Optimization Trade-off in Meta-learning, 2021, arXiv: 2010.12916 [cs, math, stat], <http://arxiv.org/abs/2010.12916> (accessed 03/07/2023).
- (33) Amazon Scraps Secret AI Recruiting Tool That Showed Bias against Women. *Reuters* **2018**, <https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G> (accessed 03/07/2023).

- (34) Götz, A. W.; Williamson, M. J.; Xu, D.; Poole, D.; Le Grand, S.; Walker, R. C. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born. *Journal of Chemical Theory and Computation* **2012**, *8*, 1542–1555, <https://doi.org/10.1021/ct200909j> (accessed 03/07/2023).
- (35) Salomon-Ferrer, R.; Götz, A. W.; Poole, D.; Le Grand, S.; Walker, R. C. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald. *Journal of Chemical Theory and Computation* **2013**, *9*, 3878–3888, <https://doi.org/10.1021/ct400314y> (accessed 03/07/2023).
- (36) Andrade, X.; Pemmaraju, C. D.; Kartsev, A.; Xiao, J.; Lindenberg, A.; Rajpurohit, S.; Tan, L. Z.; Ogitsu, T.; Correa, A. A. Inq, a Modern GPU-Accelerated Computational Framework for (Time-Dependent) Density Functional Theory. *Journal of Chemical Theory and Computation* **2021**, *17*, 7447–7467, <https://doi.org/10.1021/acs.jctc.1c00562> (accessed 03/07/2023).
- (37) Chithrananda, S.; Grand, G.; Ramsundar, B. ChemBERTa: Large-Scale Self-Supervised Pretraining for Molecular Property Prediction, 2020, arXiv: 2010.09885 [physics, q-bio], <http://arxiv.org/abs/2010.09885> (accessed 03/04/2023).
- (38) Meli, R.; Anighoro, A.; Bodkin, M. J.; Morris, G. M.; Biggin, P. C. Learning Protein-Ligand Binding Affinity with Atomic Environment Vectors. *Journal of Cheminformatics* **2021**, *13*, 59, <https://doi.org/10.1186/s13321-021-00536-w> (accessed 03/06/2023).
- (39) Schneider, N.; Fechner, N.; Landrum, G. A.; Stiefl, N. Chemical Topic Modeling: Exploring Molecular Data Sets Using a Common Text-Mining Approach. *Journal of Chemical Information and Modeling* **2017**, *57*, 1816–1831, <https://doi.org/10.1021/acs.jcim.7b00249> (accessed 03/07/2023).
- (40) Kryshchuk, A.; Schwede, T.; Topf, M.; Fidelis, K.; Moult, J. Critical Assessment of Methods of Protein Structure Prediction (CASP) – Round XIII. *Proteins* **2019**, *87*, 1011–1020, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6927249/> (accessed 03/06/2023).

- (41) Crevier, D., *AI: The Tumultuous History of the Search for Artificial Intelligence*; Basic Books, Inc.: USA, 1993.
- (42) Turing, A. M. Computing Machinery and Intelligence. *Mind; a quarterly review of psychology and philosophy* **1950**, *59*, 433–460, JSTOR: 2251299, <http://www.jstor.org/stable/2251299>.
- (43) Rosenblatt, F. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* **1958**, *65*, 386–408.
- (44) Quinlan, J. R. Induction of Decision Trees. *Machine Learning* **1986**, *1*, 81–106, <https://doi.org/10.1007/BF00116251> (accessed 03/07/2023).
- (45) Fix, E.; Hodges, J. L. Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review / Revue Internationale de Statistique* **1989**, *57*, 238–247, JSTOR: 1403797, <http://www.jstor.org/stable/1403797> (accessed 03/07/2023).
- (46) Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; Siskind, J. M. Automatic Differentiation in Machine Learning: A Survey. *Journal of Machine Learning Research* **2018**, *18*, 1–43, <http://jmlr.org/papers/v18/17-468.html> (accessed 03/07/2023).
- (47) Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. Learning Representations by Back-Propagating Errors. *Nature* **1986**, *323*, 533–536, <https://www.nature.com/articles/323533a0> (accessed 03/07/2023).
- (48) Jordan, M. I. *Serial Order: A Parallel Distributed Processing Approach*. Technical Report, June 1985–March 1986; tech. rep. (AD-A-173989/5/XAB; ICS-8604); California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986, <https://www.osti.gov/biblio/6910294> (accessed 03/07/2023).
- (49) Schmidhuber, J. Deep Learning in Neural Networks: An Overview. *Neural Networks* **2015**, *61*, 85–117, arXiv: 1404.7828 [cs], <http://arxiv.org/abs/1404.7828> (accessed 03/07/2023).
- (50) Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* **1998**, *86*, 2278–2324.

- (51) Ho, T. K. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1995; Vol. 1, 278–282 vol.1.
- (52) Cortes, C.; Vapnik, V. Support-Vector Networks. *Machine Learning* **1995**, *20*, 273–297, <https://doi.org/10.1007/BF00994018> (accessed 03/07/2023).
- (53) Arulkumaran, K.; Deisenroth, M. P.; Brundage, M.; Bharath, A. A. A Brief Survey of Deep Reinforcement Learning. *IEEE Signal Processing Magazine* **2017**, *34*, 26–38, arXiv: 1708.05866 [cs, stat], <http://arxiv.org/abs/1708.05866> (accessed 03/07/2023).
- (54) Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; Dahl, G. E. Neural Message Passing for Quantum Chemistry, 2017, arXiv: 1704.01212 [cs], <http://arxiv.org/abs/1704.01212> (accessed 03/06/2023).
- (55) Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2019, arXiv: 1810.04805 [cs], <http://arxiv.org/abs/1810.04805> (accessed 03/04/2023).
- (56) Ramsundar, B.; Eastman, P.; Walters, P.; Pande, V.; Leswing, K.; Wu, Z., *Deep Learning for the Life Sciences*; O'Reilly Media: 2019.
- (57) Berman, H.; Henrick, K.; Nakamura, H. Announcing the Worldwide Protein Data Bank. *Nature Structural & Molecular Biology* **2003**, *10*, 980–980, <https://www.nature.com/articles/nsb1203-980> (accessed 03/05/2023).
- (58) Dalby, A.; Nourse, J. G.; Hounshell, W. D.; Gushurst, A. K. I.; Grier, D. L.; Leland, B. A.; Laufer, J. Description of Several Chemical Structure File Formats Used by Computer Programs Developed at Molecular Design Limited. *Journal of Chemical Information and Computer Sciences* **1992**, *32*, 244–255, <https://doi.org/10.1021/ci00007a012> (accessed 03/07/2023).
- (59) Minkin, V. I. Glossary of Terms Used in Theoretical Organic Chemistry. *Pure and Applied Chemistry* **1999**, *71*, 1919–1981, <https://www.degruyter.com/document/doi/10.1351/pac199971101919/html> (accessed 03/07/2023).

- (60) Rogers, D.; Hahn, M. Extended-Connectivity Fingerprints. *Journal of Chemical Information and Modeling* **2010**, *50*, 742–754, <https://doi.org/10.1021/ci100050t> (accessed 03/05/2023).
- (61) Filik, J.; Ashton, A. W.; Chang, P. C. Y.; Chater, P. A.; Day, S. J.; Drakopoulos, M.; Gerring, M. W.; Hart, M. L.; Magdysyuk, O. V.; Michalik, S.; Smith, A.; Tang, C. C.; Terrill, N. J.; Wharmby, M. T.; Wilhelm, H. Processing Two-Dimensional X-ray Diffraction and Small-Angle Scattering Data in DAWN 2. *Journal of Applied Crystallography* **2017**, *50*, 959–966, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5458597/> (accessed 09/10/2022).
- (62) Riniker, S.; Fechner, N.; Landrum, G. A. Heterogeneous Classifier Fusion for Ligand-Based Virtual Screening: Or, How Decision Making by Committee Can Be a Good Thing. *Journal of Chemical Information and Modeling* **2013**, *53*, 2829–2836, <https://doi.org/10.1021/ci400466r> (accessed 03/04/2023).
- (63) David, L.; Thakkar, A.; Mercado, R.; Engkvist, O. Molecular Representations in AI-driven Drug Discovery: A Review and Practical Guide. *Journal of Cheminformatics* **2020**, *12*, 56, <https://doi.org/10.1186/s13321-020-00460-5> (accessed 03/07/2023).
- (64) Kipf, T. N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks, 2017, arXiv: 1609.02907 [cs, stat], <http://arxiv.org/abs/1609.02907> (accessed 03/07/2023).
- (65) Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *Journal of Chemical Information and Computer Sciences* **1988**, *28*, 31–36, <https://pubs.acs.org/doi/abs/10.1021/ci00057a005> (accessed 03/04/2023).
- (66) Weininger, D.; Weininger, A.; Weininger, J. L. SMILES. 2. Algorithm for Generation of Unique SMILES Notation. *Journal of Chemical Information and Computer Sciences* **1989**, *29*, 97–101, <https://doi.org/10.1021/ci00062a008> (accessed 03/07/2023).

- (67) O'Boyle, N. M. Towards a Universal SMILES Representation - A Standard Method to Generate Canonical SMILES Based on the InChI. *Journal of Cheminformatics* **2012**, *4*, 22, <https://doi.org/10.1186/1758-2946-4-22> (accessed 03/07/2023).
- (68) F., D.; Macks, D. Simplified Molecular-Input Line-Entry System. *Wikipedia* **2023**, https://en.wikipedia.org/w/index.php?title=Simplified_molecular-input_line-entry_system&oldid=1142248372 (accessed 03/07/2023).
- (69) Riniker, S.; Landrum, G. A. Open-Source Platform to Benchmark Fingerprints for Ligand-Based Virtual Screening. *Journal of Cheminformatics* **2013**, *5*, 26, <https://doi.org/10.1186/1758-2946-5-26> (accessed 03/04/2023).
- (70) 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* **2020**, *1*, 045024, <https://dx.doi.org/10.1088/2632-2153/aba947> (accessed 03/07/2023).
- (71) Bilodeau, C.; Jin, W.; Jaakkola, T.; Barzilay, R.; Jensen, K. F. Generative Models for Molecular Discovery: Recent Advances and Challenges. *WIREs Computational Molecular Science* **2022**, *12*, e1608, <https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1608> (accessed 03/07/2023).
- (72) Morgan, H. L. The Generation of a Unique Machine Description for Chemical Structures - A Technique Developed at Chemical Abstracts Service. *Journal of Chemical Documentation* **1965**, *5*, 107–113, <https://doi.org/10.1021/c160017a018> (accessed 03/04/2023).
- (73) Tyler, A. I. I.; Law, R. V.; Seddon, J. M. In *Methods in Membrane Lipids*, Owen, D. M., Ed.; Springer New York: New York, NY, 2015, pp 199–225, https://doi.org/10.1007/978-1-4939-1752-5_16.
- (74) Simeone, O. A Very Brief Introduction to Machine Learning With Applications to Communication Systems, 2018, arXiv: 1808.02342 [cs, math], <http://arxiv.org/abs/1808.02342> (accessed 03/07/2023).

- (75) Machine Learning for Email Spam Filtering: Review, Approaches and Open Research Problems - ScienceDirect, <https://www.sciencedirect.com/science/article/pii/S2405844018353404> (accessed 03/07/2023).
- (76) Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, AAAI Press: Portland, Oregon, 1996, pp 226–231, (accessed 03/07/2023).
- (77) Jolliffe, I. T.; Cadima, J. Principal Component Analysis: A Review and Recent Developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **2016**, *374*, 20150202, <https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202> (accessed 03/07/2023).
- (78) Amini, M.-R.; Feofanov, V.; Pauletto, L.; Devijver, E.; Maximov, Y. Self-Training: A Survey, 2023, arXiv: 2202.12040 [cs], <http://arxiv.org/abs/2202.12040> (accessed 03/07/2023).
- (79) Blum, A.; Mitchell, T. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, 1998, pp 92–100.
- (80) Joachims, T. et al. In *Icml*, 1999; Vol. 99, pp 200–209.
- (81) Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019, arXiv: 1907.11692 [cs], <http://arxiv.org/abs/1907.11692> (accessed 03/04/2023).
- (82) Wang, B.; Komatsuzaki, A. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model, 2021, <https://github.com/kingoflolz/mesh-transformer-jax>.
- (83) Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; Dean, J. In *Advances in Neural Information Processing Systems*, ed. by Burges, C.; Bottou, L.; Welling, M.; Ghahramani, Z.; Weinberger, K., Curran Associates, Inc.: 2013; Vol. 26, <https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf>.

- (84) Pennington, J.; Socher, R.; Manning, C. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics: Doha, Qatar, 2014, pp 1532–1543, <https://aclanthology.org/D14-1162>.
- (85) Clark, K.; Luong, M.-T.; Le, Q. V.; Manning, C. D. In *ICLR*, 2020, <https://openreview.net/pdf?id=r1xMH1BtvB>.
- (86) van Otterlo, M.; Wiering, M. In *Reinforcement Learning: State-of-the-Art*, Wiering, M., van Otterlo, M., Eds.; Adaptation, Learning, and Optimization; Springer: Berlin, Heidelberg, 2012, pp 3–42, https://doi.org/10.1007/978-3-642-27645-3_1 (accessed 03/07/2023).
- (87) Wang, J.; Chen, Q.; Chen, Y. In *Advances in Neural Networks – ISNN 2004*, ed. by Yin, F.-L.; Wang, J.; Guo, C., Springer: Berlin, Heidelberg, 2004, pp 512–517.
- (88) Zou, H.; Hastie, T. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* **2005**, *67*, 301–320, JSTOR: 3647580, <https://www.jstor.org/stable/3647580> (accessed 03/07/2023).
- (89) Mohan, P. Ridge, Lasso & Elastic Net Regression, 2021, <https://blog.devgenius.io/ridge-lasso-elastic-net-regression-2ea752186e51> (accessed 03/07/2023).
- (90) Loshchilov, I.; Hutter, F. Decoupled Weight Decay Regularization, 2019, arXiv: 1711.05101 [cs, math], <http://arxiv.org/abs/1711.05101> (accessed 03/03/2023).
- (91) Ruder, S. An Overview of Gradient Descent Optimization Algorithms, 2017, arXiv: 1609.04747 [cs], <http://arxiv.org/abs/1609.04747> (accessed 07/23/2023).
- (92) Graves, A. Generating Sequences With Recurrent Neural Networks, 2014, arXiv: 1308.0850 [cs], <http://arxiv.org/abs/1308.0850> (accessed 03/07/2023).
- (93) Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization, 2017, arXiv: 1412.6980 [cs], <http://arxiv.org/abs/1412.6980> (accessed 03/03/2023).
- (94) Duchi, J.; Hazan, E.; Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* **2011**, *12*, 2121–2159, <http://jmlr.org/papers/v12/duchi11a.html> (accessed 03/07/2023).

- (95) Wilson, A. C.; Roelofs, R.; Stern, M.; Srebro, N.; Recht, B. The Marginal Value of Adaptive Gradient Methods in Machine Learning. *Advances in neural information processing systems* **2017**, *30*.
- (96) Wang, Z.; Dahl, G. E.; Swersky, K.; Lee, C.; Nado, Z.; Gilmer, J.; Snoek, J.; Ghahramani, Z. Pre-Trained Gaussian Processes for Bayesian Optimization, 2023, arXiv: 2109.08215 [cs, stat], <http://arxiv.org/abs/2109.08215> (accessed 03/07/2023).
- (97) Zoph, B.; Le, Q. V. Neural Architecture Search with Reinforcement Learning, 2017, arXiv: 1611.01578 [cs], <http://arxiv.org/abs/1611.01578> (accessed 03/07/2023).
- (98) He, X.; Zhao, K.; Chu, X. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems* **2021**, *212*, 106622, arXiv: 1908.00709 [cs, stat], <http://arxiv.org/abs/1908.00709> (accessed 03/07/2023).
- (99) Amini, A.; Gilitschenski, I.; Phillips, J.; Moseyko, J.; Banerjee, R.; Karaman, S.; Rus, D. Learning Robust Control Policies for End-to-End Autonomous Driving from Data-Driven Simulation. *IEEE Robotics and Automation Letters* **2020**.
- (100) Amini, A.; Wang, T.-H.; Gilitschenski, I.; Schwarting, W.; Liu, Z.; Han, S.; Karaman, S.; Rus, D. In *2022 International Conference on Robotics and Automation (ICRA)*, 2022.
- (101) Wang, T.-H.; Amini, A.; Schwarting, W.; Gilitschenski, I.; Karaman, S.; Rus, D. In *2022 International Conference on Robotics and Automation (ICRA)*, 2022.
- (102) Sandfort, V.; Yan, K.; Pickhardt, P. J.; Summers, R. M. Data Augmentation Using Generative Adversarial Networks (CycleGAN) to Improve Generalizability in CT Segmentation Tasks. *Scientific Reports* **2019**, *9*, 16884, <https://www.nature.com/articles/s41598-019-52737-x> (accessed 03/07/2023).
- (103) Cybenko, G. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems* **1989**, *2*, 303–314, <https://doi.org/10.1007/BF02551274> (accessed 03/07/2023).
- (104) Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. *Learning Internal Representations by Error Propagation*; tech. rep.; California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

- (105) Williams, R. J.; Zipser, D. In *Backpropagation: Theory, Architectures, and Applications*; L. Erlbaum Associates Inc.: USA, 1995, pp 433–486, (accessed 03/07/2023).
- (106) Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **1997**, *9*, 1735–1780.
- (107) Cho, K.; van Merriënboer, B.; Bahdanau, D.; Bengio, Y. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, 2014, arXiv: 1409.1259 [cs, stat], <http://arxiv.org/abs/1409.1259> (accessed 03/07/2023).
- (108) Olah, C. Understanding LSTM Networks – Colah’s Blog, <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> (accessed 03/07/2023).
- (109) Bahdanau, D.; Cho, K.; Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate, 2016, arXiv: 1409.0473 [cs, stat], <http://arxiv.org/abs/1409.0473> (accessed 03/07/2023).
- (110) Capecchi, A.; Probst, D.; Reymond, J.-L. One Molecular Fingerprint to Rule Them All: Drugs, Biomolecules, and the Metabolome. *Journal of Cheminformatics* **2020**, *12*, 43, <https://doi.org/10.1186/s13321-020-00445-4> (accessed 03/04/2023).
- (111) Daylight Theory Manual, <https://www.daylight.com/dayhtml/doc/theory/> (accessed 03/07/2023).
- (112) Bajusz, D.; Rácz, A.; Héberger, K. Why Is Tanimoto Index an Appropriate Choice for Fingerprint-Based Similarity Calculations? *Journal of Cheminformatics* **2015**, *7*, 20, <https://doi.org/10.1186/s13321-015-0069-3> (accessed 03/07/2023).
- (113) Blum, L. C.; van Deursen, R.; Reymond, J.-L. Visualisation and Subsets of the Chemical Universe Database GDB-13 for Virtual Screening. *Journal of Computer-Aided Molecular Design* **2011**, *25*, 637–647, <https://doi.org/10.1007/s10822-011-9436-y> (accessed 03/05/2023).
- (114) Heller, S. R.; McNaught, A.; Pletnev, I.; Stein, S.; Tchekhovskoi, D. InChI, the IUPAC International Chemical Identifier. *Journal of Cheminformatics* **2015**, *7*, 23, <https://doi.org/10.1186/s13321-015-0068-4> (accessed 03/08/2023).

- (115) Adams, P. D. et al. Announcing Mandatory Submission of PDBx/mmCIF Format Files for Crystallographic Depositions to the Protein Data Bank (PDB). *Acta Crystallographica Section D: Structural Biology* **2019**, *75*, 451–454, <https://journals.iucr.org/d/issues/2019/04/00/rr5177/> (accessed 03/08/2023).
- (116) Frenkel, D.; Smit, B., *Understanding Molecular Simulation: From Algorithms to Applications*; Elsevier: 2001; Vol. 1.
- (117) Leach, A. R., *Molecular Modelling: Principles and Applications*; Pearson education: 2001.
- (118) Tian, C.; Kasavajhala, K.; Belfon, K. A. A.; Raguet, L.; Huang, H.; Miguels, A. N.; Bickel, J.; Wang, Y.; Pincay, J.; Wu, Q.; Simmerling, C. ff19SB: Amino-Acid-Specific Protein Backbone Parameters Trained against Quantum Mechanics Energy Surfaces in Solution. *Journal of Chemical Theory and Computation* **2020**, *16*, 528–552, <https://doi.org/10.1021/acs.jctc.9b00591> (accessed 03/06/2023).
- (119) Case, D. et al. AMBER 2020, University of California, San Francisco, 2020.
- (120) Case, D. A.; Cheatham III, T. E.; Darden, T.; Gohlke, H.; Luo, R.; Merz Jr., K. M.; Onufriev, A.; Simmerling, C.; Wang, B.; Woods, R. J. The Amber Biomolecular Simulation Programs. *Journal of Computational Chemistry* **2005**, *26*, 1668–1688, <https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20290> (accessed 03/06/2023).
- (121) Onufriev, A.; Bashford, D.; Case, D. A. Exploring Protein Native States and Large-Scale Conformational Changes with a Modified Generalized Born Model. *Proteins* **2004**, *55*, 383–394.
- (122) Salomon-Ferrer, R.; Case, D. A.; Walker, R. C. An Overview of the Amber Biomolecular Simulation Package. *WIREs Computational Molecular Science* **2013**, *3*, 198–210, <https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1121> (accessed 03/06/2023).
- (123) 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* **2004**, *25*, 1157–1174.

- (124) Grubmüller, H.; Heller, H.; Windemuth, A.; Schulten, K. Generalized Verlet Algorithm for Efficient Molecular Dynamics Simulations with Long-Range Interactions. *Molecular Simulation* **1991**, *6*, 121–142.
- (125) Onufriev, A. V.; Case, D. A. Generalized Born Implicit Solvent Models for Biomolecules. *Annual Review of Biophysics* **2019**, *48*, 275–296.
- (126) Bashford, D.; Case, D. A. Generalized Born Models of Macromolecular Solvation Effects. *Annual Review of Physical Chemistry* **2000**, *51*, 129–152.
- (127) Cramer, C. J.; Truhlar, D. G. Implicit Solvation Models: Equilibria, Structure, Spectra, and Dynamics. *Chemical Reviews* **1999**, *99*, 2161–2200, <https://doi.org/10.1021/cr960149m> (accessed 03/08/2023).
- (128) Dominy, B. N.; Brooks, C. L. Development of a Generalized Born Model Parametrization for Proteins and Nucleic Acids. *The Journal of Physical Chemistry B* **1999**, *103*, 3765–3773, <https://doi.org/10.1021/jp984440c> (accessed 03/08/2023).
- (129) Jayaram, B.; Sprous, D.; Beveridge, D. L. Solvation Free Energy of Biomacromolecules: Parameters for a Modified Generalized Born Model Consistent with the AMBER Force Field. *The Journal of Physical Chemistry B* **1998**, *102*, 9571–9576, <https://doi.org/10.1021/jp982007x> (accessed 03/08/2023).
- (130) Lee, M. S.; Salsbury, F. R.; Brooks, C. L. Novel Generalized Born Methods. *The Journal of Chemical Physics* **2002**, *116*, 10606–10614, <https://aip.scitation.org/doi/10.1063/1.1480013> (accessed 03/08/2023).
- (131) Onufriev, A.; Bashford, D.; Case, D. A. Modification of the Generalized Born Model Suitable for Macromolecules. *The Journal of Physical Chemistry B* **2000**, *104*, 3712–3720, <https://doi.org/10.1021/jp994072s> (accessed 03/08/2023).
- (132) Schaefer, M.; Karplus, M. A Comprehensive Analytical Treatment of Continuum Electrostatics. *The Journal of Physical Chemistry* **1996**, *100*, 1578–1599, <https://doi.org/10.1021/jp9521621> (accessed 03/08/2023).

- (133) Still, W. C.; Tempczyk, A.; Hawley, R. C.; Hendrickson, T. Semianalytical Treatment of Solvation for Molecular Mechanics and Dynamics. *Journal of the American Chemical Society* **1990**, *112*, 6127–6129, <https://doi.org/10.1021/ja00172a038> (accessed 03/08/2023).
- (134) Alberts, B.; Johnson, A.; Lewis, J.; Raff, M.; Roberts, K.; Walter, P., *Molecular Biology of the Cell*, 4th; Garland Science: 2002.
- (135) Sullan, R. M. A.; Li, J. K.; Zou, S. Direct Correlation of Structures and Nanomechanical Properties of Multicomponent Lipid Bilayers. *Langmuir* **2009**, *25*, 7471–7477, <https://doi.org/10.1021/la900395w> (accessed 03/02/2023).
- (136) Xi, E.; Patel, A. J. The Hydrophobic Effect, and Fluctuations: The Long and the Short of It. *Proceedings of the National Academy of Sciences* **2016**, *113*, 4549–4551, <https://www.pnas.org/doi/abs/10.1073/pnas.1603014113> (accessed 03/02/2023).
- (137) Fuller, N.; Rand, R. P. The Influence of Lysolipids on the Spontaneous Curvature and Bending Elasticity of Phospholipid Membranes. *Biophysical Journal* **2001**, *81*, 243–254, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1301507/> (accessed 03/02/2023).
- (138) Israelachvili, J. N.; Mitchell, D. J.; Ninham, B. W. Theory of Self-Assembly of Hydrocarbon Amphiphiles into Micelles and Bilayers. *Journal of the Chemical Society, Faraday Transactions 2: Molecular and Chemical Physics* **1976**, *72*, 1525–1568, <https://pubs.rsc.org/en/content/articlelanding/1976/f2/f29767201525> (accessed 03/02/2023).
- (139) Alipour, E.; Halverson, D.; McWhirter, S.; Walker, G. C. Phospholipid Bilayers: Stability and Encapsulation of Nanoparticles. *Annual Review of Physical Chemistry* **2017**, *68*, 261–283, <https://doi.org/10.1146/annurev-physchem-040215-112634> (accessed 03/02/2023).
- (140) Kučerka, N.; Pencser, J.; Sachs, J. N.; Nagle, J. F.; Katsaras, J. Curvature Effect on the Structure of Phospholipid Bilayers. *Langmuir : the ACS journal of surfaces and colloids*

- 2007, 23, 1292–1299, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2720570/> (accessed 03/02/2023).
- (141) Jouhet, J. Importance of the Hexagonal Lipid Phase in Biological Membrane Organization. *Frontiers in Plant Science* **2013**, 4, <https://www.frontiersin.org/articles/10.3389/fpls.2013.00494> (accessed 03/02/2023).
- (142) Jacobson, K.; Papahadjopoulos, D. Phase Transitions and Phase Separations in Phospholipid Membranes Induced by Changes in Temperature, pH, and Concentration of Bivalent Cations. *Biochemistry* **1975**, 14, 152–161, <https://doi.org/10.1021/bi00672a026> (accessed 03/02/2023).
- (143) Ge, M.; Freed, J. H. Hydration, Structure, and Molecular Interactions in the Headgroup Region of Dioleoylphosphatidylcholine Bilayers: An Electron Spin Resonance Study. *Biophysical Journal* **2003**, 85, 4023–4040, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1303703/> (accessed 03/02/2023).
- (144) KATO, M.; HAYASHI, R. Effects of High Pressure on Lipids and Biomembranes for Understanding High-Pressure-Induced Biological Phenomena. *Bioscience, Biotechnology, and Biochemistry* **1999**, 63, 1321–1328, <https://doi.org/10.1271/bbb.63.1321> (accessed 03/02/2023).
- (145) Helfrich, W. Lyotropic Lamellar Phases. *Journal of Physics: Condensed Matter* **1994**, 6, A79, <https://dx.doi.org/10.1088/0953-8984/6/23A/009> (accessed 03/02/2023).
- (146) Vamparys, L.; Gautier, R.; Vanni, S.; Bennett, W. D.; Tieleman, D. P.; Antonny, B.; Etchebest, C.; Fuchs, P. F. Conical Lipids in Flat Bilayers Induce Packing Defects Similar to That Induced by Positive Curvature. *Biophysical Journal* **2013**, 104, 585–593, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3566444/> (accessed 03/02/2023).
- (147) Huang, Y.; Gui, S. Factors Affecting the Structure of Lyotropic Liquid Crystals and the Correlation between Structure and Drug Diffusion. *RSC Advances* **2018**, 8, 6978–6987, <https://pubs.rsc.org/en/content/articlelanding/2018/ra/c7ra12008g> (accessed 03/02/2023).

- (148) Seddon, J. M.; Conn, C. E., *Liposomes, Lipid Bilayers and Model Membranes: From Basic Research to Application*; Pabst, G., Kucerka, N., Nieh, M.-P., Katsaras, J., Eds.; CRC Press: Boca Raton, 2014.
- (149) Shearman, G. C.; Ces, O.; Templer, R. H.; Seddon, J. M. Inverse Lyotropic Phases of Lipids and Membrane Curvature. *Journal of Physics. Condensed Matter: An Institute of Physics Journal* **2006**, *18*, S1105–1124.
- (150) Seddon, J. M. Structure of the Inverted Hexagonal (HII) Phase, and Non-Lamellar Phase Transitions of Lipids. *Biochimica et Biophysica Acta (BBA) - Reviews on Biomembranes* **1990**, *1031*, 1–69, <https://www.sciencedirect.com/science/article/pii/030441579090002T> (accessed 03/02/2023).
- (151) Seddon, J.; Templer, R. In *Structure and Dynamics of Membranes*, Lipowsky, R., Sackmann, E., Eds.; Handbook of Biological Physics, Vol. 1; North-Holland: 1995, pp 97–160, <https://www.sciencedirect.com/science/article/pii/S1383812106800205>.
- (152) van Langen, H.; Schrama, C. A.; van Ginkel, G.; Ranke, G.; Levine, Y. K. Order and Dynamics in the Lamellar L Alpha and in the Hexagonal HII Phase. Dioleoylphosphatidylethanolamine Studied with Angle-Resolved Fluorescence Depolarization. *Biophysical Journal* **1989**, *55*, 937–947, <https://www.sciencedirect.com/science/article/pii/S0006349589828929> (accessed 03/02/2023).
- (153) Kranenburg, M.; Smit, B. Phase Behavior of Model Lipid Bilayers. *The Journal of Physical Chemistry B* **2005**, *109*, 6553–6563, <https://doi.org/10.1021/jp0457646> (accessed 03/02/2023).
- (154) Koynova, R.; Tenchov, B In *OA Biochemistry*, 2013; Vol. 1, <http://www.oapublishinglondon.com/article/602> (accessed 03/02/2023).
- (155) Brooks, N. J.; Ces, O.; Templer, R. H.; Seddon, J. M. Pressure Effects on Lipid Membrane Structure and Dynamics. *Chemistry and Physics of Lipids* **2011**, *164*, 89–98, <https://www.sciencedirect.com/science/article/pii/S0009308410003889> (accessed 03/02/2023).

- (156) Matsuki, H.; Goto, M.; Tada, K.; Tamai, N. Thermotropic and Barotropic Phase Behavior of Phosphatidylcholine Bilayers. *International Journal of Molecular Sciences* **2013**, *14*, 2282–2302, <https://www.mdpi.com/1422-0067/14/2/2282> (accessed 07/22/2023).
- (157) Rand, R. P.; Fuller, N. L.; Gruner, S. M.; Parsegian, V. A. Membrane Curvature, Lipid Segregation, and Structural Transitions for Phospholipids under Dual-Solvent Stress. *Biochemistry* **1990**, *29*, 76–87.
- (158) Rappolt, M.; Hickel, A.; Bringezu, F.; Lohner, K. Mechanism of the Lamellar/Inverse Hexagonal Phase Transition Examined by High Resolution X-Ray Diffraction. *Biophysical Journal* **2003**, *84*, 3111–3122, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1302872/> (accessed 03/02/2023).
- (159) Kirk, G. L.; Gruner, S. M.; Stein, D. L. A Thermodynamic Model of the Lamellar to Inverse Hexagonal Phase Transition of Lipid Membrane-Water Systems. *Biochemistry* **1984**, *23*, 1093–1102, <http://www.scopus.com/inward/record.url?scp=0000181355&partnerID=8YFLogxK> (accessed 03/02/2023).
- (160) Jahn, R.; Grubmüller, H. Membrane Fusion. *Current Opinion in Cell Biology* **2002**, *14*, 488–495, <https://www.sciencedirect.com/science/article/pii/S0955067402003563> (accessed 03/02/2023).
- (161) Meeks, W.; Pérez, J. The Classical Theory of Minimal Surfaces. *Bulletin of the American Mathematical Society* **2011**, *48*, 325–407, <https://www.ams.org/bull/2011-48-03/S0273-0979-2011-01334-9/> (accessed 03/02/2023).
- (162) Oka, T. Transformation between Inverse Bicontinuous Cubic Phases of a Lipid from Diamond to Primitive. *Langmuir* **2015**, *31*, 3180–3185, <https://doi.org/10.1021/la504295v> (accessed 03/02/2023).
- (163) Scriven, L. E. Equilibrium Bicontinuous Structure. *Nature* **1976**, *263*, 123–125, <https://www.nature.com/articles/263123a0> (accessed 03/02/2023).
- (164) Harper, P. E.; Gruner, S. M.; Lewis, R. N. a. H.; McElhaney, R. N. Electron Density Modeling and Reconstruction of Infinite Periodic Minimal Surfaces (IPMS) Based Phases in Lipid-Water Systems. II. Reconstruction of D Surface Based Phases. *The*

- European Physical Journal E* **2000**, *2*, 229–245, <http://dx.doi.org/10.1007/PL00013664> (accessed 03/02/2023).
- (165) Yan, H.; Voorhees, P. W.; Xin, H. L. Nanoscale X-Ray and Electron Tomography. *MRS Bulletin* **2020**, *45*, 264–271, <https://doi.org/10.1557/mrs.2020.90> (accessed 03/03/2023).
- (166) Li, T.; Senesi, A. J.; Lee, B. Small Angle X-ray Scattering for Nanoparticle Research. *Chemical Reviews* **2016**, *116*, 11128–11180.
- (167) SS, G. S. S. ASSESSMENT OF BRANCHING IN POLYMERS FROM SMALL ANGLE X-RAY SCATTERING (SAXS). SAXS FROM MODEL COMB-BRANCHED POLYSTYRENES. *POLYMER*; ISSN 0032-3861; GBR; DA. 1982; VOL. 23; NO 4; PP. 514-520; BIBL. 38 REF. **1982**.
- (168) Cullity, B.; Stock, S., *Elements of X-ray Diffraction, Third Edition*; Prentice-Hall: New York, 2001.
- (169) Bragg, W. H.; Bragg, W. L. The Reflection of X-rays by Crystals. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* **1997**, *88*, 428–438, <https://royalsocietypublishing.org/doi/10.1098/rspa.1913.0040> (accessed 03/03/2023).
- (170) Woolfson, M. M., *An Introduction to X-ray Crystallography*, 2nd ed.; Cambridge University Press: Cambridge, 1997.
- (171) Glazer, A. M., *A Journey into Reciprocal Space*; 2053-2571; Morgan & Claypool Publishers: 2017, <https://dx.doi.org/10.1088/978-1-6817-4621-0>.
- (172) *Fundamentals of Powder Diffraction and Structural Characterization of Materials*; Springer US: Boston, MA, 2009, <http://link.springer.com/10.1007/978-0-387-09579-0> (accessed 03/03/2023).
- (173) Yager, K. GISAXS, http://gisaxs.com/index.php/Main_Page (accessed 03/03/2023).

- (174) Statistical Expectation Value of the Debye-Waller Factor and $E(\text{Hkl})$ Values for Macromolecular Crystals. *Acta Crystallographica. Section D Biological Crystallography* **1996**, *52*, 257–266, <https://eurekamag.com/research/009/454/009454230.php> (accessed 03/03/2023).
- (175) AlOmar, A. S. Line Width at Half Maximum of the Voigt Profile in Terms of Gaussian and Lorentzian Widths: Normalization, Asymptotic Expansion, and Chebyshev Approximation. *Optik* **2020**, *203*, 163919, <https://www.sciencedirect.com/science/article/pii/S0030402619318170> (accessed 03/03/2023).
- (176) Baddour, N. In *Advances in Imaging and Electron Physics*, Hawkes, P. W., Ed.; Advances in Imaging and Electron Physics, Vol. 165; Elsevier: 2011, pp 1–45, <https://www.sciencedirect.com/science/article/pii/B9780123858610000014>.
- (177) In *Special Functions*; John Wiley & Sons, Ltd: 1996; Chapter 1, pp 1–27, eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118032572.ch1>, <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118032572.ch1>.
- (178) Siegman, A. E. Quasi Fast Hankel Transform. *Optics Letters* **1977**, *1*, 13–15, <https://opg.optica.org/ol/abstract.cfm?URI=ol-1-1-13>.
- (179) Fraser, S. J.; Mulet, X.; Hawley, A.; Separovic, F.; Polyzos, A. Controlling Nanostructure and Lattice Parameter of the Inverse Bicontinuous Cubic Phases in Functionalised Phytantriol Dispersions. *Journal of Colloid and Interface Science* **2013**, *408*, 117–124, <https://www.sciencedirect.com/science/article/pii/S0021979713006425> (accessed 03/03/2023).
- (180) Gandy, P. J. F.; Klinowski, J. Exact Computation of the Triply Periodic G (‘Gyroid’) Minimal Surface. *Chemical Physics Letters* **2000**, *321*, 363–371, <https://www.sciencedirect.com/science/article/pii/S0009261400003730> (accessed 03/03/2023).
- (181) Garstecki, P.; Hołyst, R. Scattering on Triply Periodic Minimal Surfaces—the Effect of the Topology, Debye–Waller, and Molecular Form Factors. *The Journal of Chemical Physics* **2000**, *113*, 3772–3779, <https://aip.scitation.org/doi/10.1063/1.1287426> (accessed 03/03/2023).

- (182) Garstecki, P.; Holyst, R. Scattering Patterns of Self-Assembled Gyroid Cubic Phases in Amphiphilic Systems. *The Journal of Chemical Physics* **2001**, *115*, 1095–1099, <https://aip.scitation.org/doi/abs/10.1063/1.1379326> (accessed 03/03/2023).
- (183) Garstecki, P.; Holyst, R. Scattering Patterns of Self-Assembled Cubic Phases. 1. The Model. *Langmuir* **2002**, *18*, 2519–2528, <https://doi.org/10.1021/la011298p> (accessed 03/03/2023).
- (184) Garstecki, P.; Holyst, R. Scattering Patterns of Self-Assembled Cubic Phases. 2. Analysis of the Experimental Spectra. *Langmuir* **2002**, *18*, 2529–2537, <https://doi.org/10.1021/la011299h> (accessed 03/03/2023).
- (185) Bruinsma, R. Elasticity and Excitations of Minimal Crystals. *Journal de Physique II* **1992**, *2*, 425–451, <http://dx.doi.org/10.1051/jp2:1992142> (accessed 03/03/2023).
- (186) Lorensen, W. E.; Cline, H. E. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *ACM SIGGRAPH Computer Graphics* **1987**, *21*, 163–169, <https://doi.org/10.1145/37402.37422> (accessed 03/03/2023).
- (187) Lewiner, T.; Lopes, H.; Vieira, A. W.; Tavares, G. Efficient Implementation of Marching Cubes' Cases with Topological Guarantees. *Journal of Graphics Tools* **2003**, *8*, 1–15, <https://doi.org/10.1080/10867651.2003.10487582> (accessed 03/03/2023).
- (188) Levenberg, K. A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quarterly of Applied Mathematics* **1944**, *2*, 164–168, <https://www.ams.org/qam/1944-02-02/S0033-569X-1944-10666-0/> (accessed 03/03/2023).
- (189) Marquardt, D. W. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics* **1963**, *11*, 431–441, <https://epubs.siam.org/doi/10.1137/0111030> (accessed 03/03/2023).
- (190) Wells, R. J. Rapid Approximation to the Voigt/Faddeeva Function and Its Derivatives. *Journal of Quantitative Spectroscopy and Radiative Transfer* **1999**, *62*, 29–48, <https://www.sciencedirect.com/science/article/pii/S0022407397002318> (accessed 03/03/2023).

- (191) Aty, H. A.; Strutt, R.; McIntyre, N.; Allen, M.; Barlow, N. E.; Páez-Pérez, M.; Seddon, J. M.; Brooks, N.; Ces, O.; Gould, I. R. Machine Learning Platform for Determining Experimental Lipid Phase Behaviour from Small Angle X-ray Scattering Patterns by Pre-Training on Synthetic Data. *Digital Discovery* **2022**, *1*, 98–107, <https://pubs.rsc.org/en/content/articlelanding/2022/dd/d1dd00025j> (accessed 06/23/2022).
- (192) Grinberg, M., *Flask Web Development: Developing Web Applications with Python*; "O'Reilly Media, Inc.": 2018.
- (193) Bokeh Development Team Bokeh: Python Library for Interactive Visualization, Manual, 2014, <http://www.bokeh.pydata.org>.
- (194) Dyer, S.; Dyer, J. Cubic-Spline Interpolation. 1. *IEEE Instrumentation & Measurement Magazine* **2001**, *4*, 44–46.
- (195) Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017, arXiv: 1704.04861 [cs], <http://arxiv.org/abs/1704.04861> (accessed 03/03/2023).
- (196) Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks, 2019, arXiv: 1801.04381 [cs], <http://arxiv.org/abs/1801.04381> (accessed 03/03/2023).
- (197) He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition, 2015, arXiv: 1512.03385 [cs], <http://arxiv.org/abs/1512.03385> (accessed 03/03/2023).
- (198) Lin, M.; Chen, Q.; Yan, S. Network In Network, 2014, arXiv: 1312.4400 [cs], <http://arxiv.org/abs/1312.4400> (accessed 03/03/2023).
- (199) GeeksforGeeks Depth Wise Separable Convolutional Neural Networks, 2018, <https://www.geeksforgeeks.org/depth-wise-separable-convolutional-neural-networks/> (accessed 03/04/2023).

- (200) Fabian, B.; Edlich, T.; Gaspar, H.; Segler, M.; Meyers, J.; Fiscato, M.; Ahmed, M. Molecular Representation Learning with Language Models and Domain-Relevant Auxiliary Tasks, 2020, arXiv: 2011.13230 [cs], <http://arxiv.org/abs/2011.13230> (accessed 03/04/2023).
- (201) Jin, W.; Coley, C.; Barzilay, R.; Jaakkola, T. Predicting Organic Reaction Outcomes with Weisfeiler-Lehman Network. *Advances in neural information processing systems* **2017**, *30*.
- (202) Kearnes, S.; McCloskey, K.; Berndl, M.; Pande, V.; Riley, P. Molecular Graph Convolutions: Moving beyond Fingerprints. *Journal of Computer-Aided Molecular Design* **2016**, *30*, 595–608, <https://doi.org/10.1007/s10822-016-9938-8> (accessed 03/04/2023).
- (203) Winter, R.; Montanari, F.; Noé, F.; Clevert, D.-A. Learning Continuous and Data-Driven Molecular Descriptors by Translating Equivalent Chemical Representations. *Chemical Science* **2019**, *10*, 1692–1701, <https://pubs.rsc.org/en/content/articlelanding/2019/sc/c8sc04175j> (accessed 03/04/2023).
- (204) Coley, C. W.; Barzilay, R.; Green, W. H.; Jaakkola, T. S.; Jensen, K. F. Convolutional Embedding of Attributed Molecular Graphs for Physical Property Prediction. *Journal of Chemical Information and Modeling* **2017**, *57*, 1757–1772, <https://doi.org/10.1021/acs.jcim.6b00601> (accessed 03/04/2023).
- (205) Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; Adams, R. P. In *Advances in Neural Information Processing Systems*, ed. by Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M.; Garnett, R., Curran Associates, Inc.: 2015; Vol. 28, <https://proceedings.neurips.cc/paper/2015/file/f9be311e65d81a9ad8150a6Paper.pdf>.
- (206) Landrum, G. et al. Rdkit/Rdkit: 2021_03_1 (Q1 2021) Release, Zenodo, 2021, <https://zenodo.org/record/4639764> (accessed 03/04/2023).
- (207) Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; Soricut, R. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, 2020, arXiv: 1909.11942 [cs], <http://arxiv.org/abs/1909.11942> (accessed 03/04/2023).

- (208) Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; Zettlemoyer, L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension, 2019, arXiv: 1910.13461 [cs, stat], <http://arxiv.org/abs/1910.13461> (accessed 03/04/2023).
- (209) Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language Models Are Unsupervised Multitask Learners. **2019**, <https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf> (accessed 03/07/2023).
- (210) Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P. J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, 2020, arXiv: 1910.10683 [cs, stat], <http://arxiv.org/abs/1910.10683> (accessed 03/04/2023).
- (211) Workshop, B. et al. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model, 2022, arXiv: 2211.05100 [cs], <http://arxiv.org/abs/2211.05100> (accessed 03/04/2023).
- (212) Weyssow, M.; Sahraoui, H.; Syriani, E. Recommending Metamodel Concepts during Modeling Activities with Pre-Trained Language Models. *Software and Systems Modeling* **2022**, *21*, 1071–1089, <https://doi.org/10.1007/s10270-022-00975-5> (accessed 07/16/2023).
- (213) Sennrich, R.; Haddow, B.; Birch, A. Neural Machine Translation of Rare Words with Subword Units, 2016, arXiv: 1508.07909 [cs], <http://arxiv.org/abs/1508.07909> (accessed 03/04/2023).
- (214) Kudo, T. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates, 2018, arXiv: 1804.10959 [cs], <http://arxiv.org/abs/1804.10959> (accessed 03/04/2023).
- (215) Abdel-Aty, H.; Gould, I. R. Large-Scale Distributed Training of Transformers for Chemical Fingerprinting. *Journal of Chemical Information and Modeling* **2022**, *62*, 4852–4862, <https://doi.org/10.1021/acs.jcim.2c00715> (accessed 03/04/2023).

- (216) Ehmki, E. S. R.; Schmidt, R.; Ohm, F.; Rarey, M. Comparing Molecular Patterns Using the Example of SMARTS: Applications and Filter Collection Analysis. *Journal of Chemical Information and Modeling* **2019**, *59*, 2572–2586, <https://doi.org/10.1021/acs.jcim.9b00249> (accessed 03/04/2023).
- (217) Gaulton, A.; Bellis, L. J.; Bento, A. P.; Chambers, J.; Davies, M.; Hersey, A.; Light, Y.; McGlinchey, S.; Michalovich, D.; Al-Lazikani, B.; Overington, J. P. ChEMBL: A Large-Scale Bioactivity Database for Drug Discovery. *Nucleic Acids Research* **2012**, *40*, D1100–D1107, <https://doi.org/10.1093/nar/gkr777> (accessed 03/05/2023).
- (218) Blum, L. C.; Reymond, J.-L. 970 Million Druglike Small Molecules for Virtual Screening in the Chemical Universe Database GDB-13. *Journal of the American Chemical Society* **2009**, *131*, 8732–8733, <https://doi.org/10.1021/ja902302h> (accessed 03/05/2023).
- (219) PubChem in 2021: New Data Content and Improved Web Interfaces — *Nucleic Acids Research* — Oxford Academic, <https://academic.oup.com/nar/article/49/D1/D1388/5957164> (accessed 03/05/2023).
- (220) Sterling, T.; Irwin, J. J. ZINC 15 – Ligand Discovery for Everyone. *Journal of Chemical Information and Modeling* **2015**, *55*, 2324–2337, <https://doi.org/10.1021/acs.jcim.5b00559> (accessed 03/05/2023).
- (221) Lowe, D. Chemical Reactions from US Patents (1976-Sep2016). *Figshare* <https://doi.org/10.6084/m9.figshare.5104873>, v1.
- (222) Fisher, R. A.; Yates, F., *Statistical Tables for Biological, Agricultural and Medical Research*, Edited by R.A. Fisher and F. Yates. 6th Ed. Edinburgh: Oliver and Boyd: 1963, <https://digital.library.adelaide.edu.au/dspace/handle/2440/10701> (accessed 03/05/2023).
- (223) Salle, A. Terashuf, 2023, <https://github.com/alexandres/terashuf> (accessed 03/05/2023).
- (224) Siebenmann, Chris The History of Unix’s Confusing Set of Low-Level Ways to Allocate Memory, <https://utcc.utoronto.ca/~cks/space/blog/unix/SbrkVersusMmap> (accessed 03/05/2023).

- (225) NCCL, NVIDIA Corporation, 2023, <https://github.com/NVIDIA/ncccl> (accessed 03/05/2023).
- (226) Yang, C. Tree-based Allreduce Communication on MXNet. **2019**, <https://www.semanticscholar.org/paper/Tree-based-Allreduce-Communication-on-MXNet-Yang/3371346d73b4cdfdeb3e0895e59cc626aad75ba8> (accessed 07/23/2023).
- (227) Glorot, X.; Bengio, Y. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ed. by Teh, Y. W.; Titterton, M., PMLR: Chia Laguna Resort, Sardinia, Italy, 2010; Vol. 9, pp 249–256, <https://proceedings.mlr.press/v9/glorot10a.html>.
- (228) Wolf, T. et al. HuggingFace’s Transformers: State-of-the-art Natural Language Processing, 2020, arXiv: 1910.03771 [cs], <http://arxiv.org/abs/1910.03771> (accessed 03/05/2023).
- (229) Truchon, J.-F.; Bayly, C. I. Evaluating Virtual Screening Methods: Good and Bad Metrics for the “Early Recognition” Problem. *Journal of Chemical Information and Modeling* **2007**, *47*, 488–508, <https://doi.org/10.1021/ci600426e> (accessed 03/05/2023).
- (230) McGaughey, G. B.; Sheridan, R. P.; Bayly, C. I.; Culberson, J. C.; Kreatsoulas, C.; Lindsley, S.; Maiorov, V.; Truchon, J.-F.; Cornell, W. D. Comparison of Topological, Shape, and Docking Methods in Virtual Screening. *Journal of chemical information and modeling* **2007**, *47*, 1504–1519.
- (231) Sheridan, R. P.; Singh, S. B.; Fluder, E. M.; Kearsley, S. K. Protocols for Bridging the Peptide to Nonpeptide Gap in Topological Similarity Searches. *Journal of chemical information and computer sciences* **2001**, *41*, 1395–1406.
- (232) Glide: A New Approach for Rapid, Accurate Docking and Scoring. 2. Enrichment Factors in Database Screening — Journal of Medicinal Chemistry, <https://pubs.acs.org/doi/10.1021/jm030644s> (accessed 03/05/2023).
- (233) Chen, D.; Zheng, J.; Wei, G.-W.; Pan, F. Extracting Predictive Representations from Hundreds of Millions of Molecules. *The Journal of Physical Chemistry Letters* **2021**,

- 12, 10793–10801, <https://doi.org/10.1021/acs.jpcclett.1c03058> (accessed 03/05/2023).
- (234) Martins, I. F.; Teixeira, A. L.; Pinheiro, L.; Falcao, A. O. A Bayesian Approach to in Silico Blood-Brain Barrier Penetration Modeling. *Journal of Chemical Information and Modeling* **2012**, *52*, 1686–1697, <https://doi.org/10.1021/ci300124c> (accessed 03/05/2023).
- (235) AIDS Antiviral Screen Data - NCI DTP Data - NCI Wiki, <https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data> (accessed 03/05/2023).
- (236) Gayvert, K. M.; Madhukar, N. S.; Elemento, O. A Data-Driven Approach to Predicting Successes and Failures of Clinical Trials. *Cell Chemical Biology* **2016**, *23*, 1294–1301.
- (237) Integrated Deep Learned Transcriptomic and Structure-Based Predictor of Clinical Trials Outcomes — bioRxiv, <https://www.biorxiv.org/content/10.1101/095653v2> (accessed 03/05/2023).
- (238) Van der Maaten, L.; Hinton, G. Visualizing Data Using T-SNE. *Journal of Machine Learning Research* **2008**, *9*, 2579–2605, <http://jmlr.org/papers/v9/vandermaaten08a.html> (accessed 03/05/2023).
- (239) Delaney, J. S. ESOL: Estimating Aqueous Solubility Directly from Molecular Structure. *Journal of Chemical Information and Computer Sciences* **2004**, *44*, 1000–1005.
- (240) Mobley, D. L.; Guthrie, J. P. FreeSolv: A Database of Experimental and Calculated Hydration Free Energies, with Input Files. *Journal of Computer-Aided Molecular Design* **2014**, *28*, 711–720.
- (241) Kendall, A.; Gal, Y.; Cipolla, R. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics, 2018, arXiv: 1705.07115 [cs], <http://arxiv.org/abs/1705.07115> (accessed 03/05/2023).
- (242) McCloskey, M.; Cohen, N. J. In *Psychology of Learning and Motivation*, Bower, G. H., Ed.; Academic Press: 1989; Vol. 24, pp 109–165, <https://www.sciencedirect.com/science/article/pii/S0079742108605368> (accessed 03/05/2023).

- (243) Ratcliff, R. Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions. *Psychological Review* **1990**, *97*, 285–308.
- (244) Schlimmer, J. C.; Fisher, D. H. In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*, ed. by Kehler, T., Morgan Kaufmann: 1986, pp 496–501, <http://www.aaai.org/Library/AAAI/1986/aaai86-083.php>.
- (245) Aldeghi, M.; Heifetz, A.; Bodkin, M. J.; Knapp, S.; Biggin, P. C. Accurate Calculation of the Absolute Free Energy of Binding for Drug Molecules. *Chemical Science* **2015**, *7*, 207–218, <https://pubs.rsc.org/en/content/articlelanding/2016/sc/c5sc02678d> (accessed 03/05/2023).
- (246) Dickson, M.; Gagnon, J. P. Key Factors in the Rising Cost of New Drug Discovery and Development. *Nature Reviews. Drug Discovery* **2004**, *3*, 417–429.
- (247) Drews, J. Drug Discovery: A Historical Perspective. *Science (New York, N.Y.)* **2000**, *287*, 1960–1964.
- (248) Kitchen, D. B.; Decornez, H.; Furr, J. R.; Bajorath, J. Docking and Scoring in Virtual Screening for Drug Discovery: Methods and Applications. *Nature Reviews Drug Discovery* **2004**, *3*, 935–949, <https://www.nature.com/articles/nrd1549> (accessed 03/05/2023).
- (249) Sliwoski, G.; Kothiwale, S.; Meiler, J.; Lowe, E. W. Computational Methods in Drug Discovery. *Pharmacological Reviews* **2014**, *66*, 334–395.
- (250) Stank, A.; Kokh, D. B.; Fuller, J. C.; Wade, R. C. Protein Binding Pocket Dynamics. *Accounts of Chemical Research* **2016**, *49*, 809–815, <https://doi.org/10.1021/acs.accounts.5b00516> (accessed 03/05/2023).
- (251) de Vries, R. M. J. M.; Meijer, F. A.; Doveston, R. G.; Leijten-van de Gevel, I. A.; Brunsveld, L. Cooperativity between the Orthosteric and Allosteric Ligand Binding Sites of ROR γ t. *Proceedings of the National Academy of Sciences* **2021**, *118*, e2021287118, <https://www.pnas.org/doi/10.1073/pnas.2021287118> (accessed 03/05/2023).

- (252) Liu, Z.; Su, M.; Han, L.; Liu, J.; Yang, Q.; Li, Y.; Wang, R. Forging the Basis for Developing Protein–Ligand Interaction Scoring Functions. *Accounts of Chemical Research* **2017**, *50*, 302–309, <https://doi.org/10.1021/acs.accounts.6b00491> (accessed 03/06/2023).
- (253) Wang, R.; Fang, X.; Lu, Y.; Wang, S. The PDBbind Database: Collection of Binding Affinities for Protein-Ligand Complexes with Known Three-Dimensional Structures. *Journal of Medicinal Chemistry* **2004**, *47*, 2977–2980.
- (254) Wang, R.; Fang, X.; Lu, Y.; Yang, C.-Y.; Wang, S. The PDBbind Database: Methodologies and Updates. *Journal of Medicinal Chemistry* **2005**, *48*, 4111–4119.
- (255) Su, M.; Yang, Q.; Du, Y.; Feng, G.; Liu, Z.; Li, Y.; Wang, R. Comparative Assessment of Scoring Functions: The CASF-2016 Update. *Journal of Chemical Information and Modeling* **2019**, *59*, 895–913, <https://doi.org/10.1021/acs.jcim.8b00545> (accessed 03/06/2023).
- (256) Li, Y.; Han, L.; Liu, Z.; Wang, R. Comparative Assessment of Scoring Functions on an Updated Benchmark: 2. Evaluation Methods and General Results. *Journal of Chemical Information and Modeling* **2014**, *54*, 1717–1736, <https://doi.org/10.1021/ci500081m> (accessed 03/06/2023).
- (257) Halgren, T. A.; Murphy, R. B.; Friesner, R. A.; Beard, H. S.; Frye, L. L.; Pollard, W. T.; Banks, J. L. Glide: A New Approach for Rapid, Accurate Docking and Scoring. 2. Enrichment Factors in Database Screening. *Journal of Medicinal Chemistry* **2004**, *47*, 1750–1759, <https://doi.org/10.1021/jm030644s> (accessed 03/05/2023).
- (258) Jones, G.; Willett, P.; Glen, R. C.; Leach, A. R.; Taylor, R. Development and Validation of a Genetic Algorithm for Flexible docking¹¹Edited by F. E. Cohen. *Journal of Molecular Biology* **1997**, *267*, 727–748, <https://www.sciencedirect.com/science/article/pii/S0022283696908979> (accessed 03/05/2023).
- (259) Morris, G. M.; Huey, R.; Lindstrom, W.; Sanner, M. F.; Belew, R. K.; Goodsell, D. S.; Olson, A. J. AutoDock4 and AutoDockTools4: Automated Docking with Selective Re-

- ceptor Flexibility. *Journal of Computational Chemistry* **2009**, *30*, 2785–2791, <https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21256> (accessed 03/05/2023).
- (260) Ravindranath, P. A.; Forli, S.; Goodsell, D. S.; Olson, A. J.; Sanner, M. F. AutoDockFR: Advances in Protein-Ligand Docking with Explicitly Specified Binding Site Flexibility. *PLOS Computational Biology* **2015**, *11*, e1004586, <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004586> (accessed 03/05/2023).
- (261) Trott, O.; Olson, A. J. AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading. *Journal of Computational Chemistry* **2010**, *31*, 455–461, <https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21334> (accessed 03/05/2023).
- (262) Muegge, I.; Rarey, M. In *Reviews in Computational Chemistry*; John Wiley & Sons, Ltd: 2001; Chapter 1, pp 1–60, <https://onlinelibrary.wiley.com/doi/abs/10.1002/0471224413.ch1> (accessed 03/05/2023).
- (263) Cang, Z.; Mu, L.; Wei, G.-W. Representability of Algebraic Topology for Biomolecules in Machine Learning Based Scoring and Virtual Screening. *PLOS Computational Biology* **2018**, *14*, e1005929, <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005929> (accessed 03/05/2023).
- (264) Feinberg, E. N.; Sur, D.; Wu, Z.; Husic, B. E.; Mai, H.; Li, Y.; Sun, S.; Yang, J.; Ramsundar, B.; Pande, V. S. PotentialNet for Molecular Property Prediction. *ACS Central Science* **2018**, *4*, 1520–1530, <https://doi.org/10.1021/acscentsci.8b00507> (accessed 03/05/2023).
- (265) Hassan-Harrirou, H.; Zhang, C.; Lemmin, T. RosENet: Improving Binding Affinity Prediction by Leveraging Molecular Mechanics Energies with an Ensemble of 3D Convolutional Neural Networks. *Journal of Chemical Information and Modeling* **2020**, *60*, 2791–2802, <https://doi.org/10.1021/acs.jcim.0c00075> (accessed 03/05/2023).
- (266) Jiménez, J.; Škalič, M.; Martínez-Rosell, G.; De Fabritiis, G. KDEEP: Protein–Ligand Absolute Binding Affinity Prediction via 3D-Convolutional Neural Networks. *Journal of*

- Chemical Information and Modeling* **2018**, *58*, 287–296, <https://doi.org/10.1021/acs.jcim.7b00650> (accessed 03/05/2023).
- (267) McNutt, A. T.; Francoeur, P.; Aggarwal, R.; Masuda, T.; Meli, R.; Ragoza, M.; Sunseri, J.; Koes, D. R. GNINA 1.0: Molecular Docking with Deep Learning. *Journal of Cheminformatics* **2021**, *13*, 43, <https://doi.org/10.1186/s13321-021-00522-2> (accessed 03/05/2023).
- (268) Ragoza, M.; Hochuli, J.; Idrobo, E.; Sunseri, J.; Koes, D. R. Protein–Ligand Scoring with Convolutional Neural Networks. *Journal of Chemical Information and Modeling* **2017**, *57*, 942–957, <https://doi.org/10.1021/acs.jcim.6b00740> (accessed 03/05/2023).
- (269) Stepniewska-Dziubinska, M. M.; Zielenkiewicz, P.; Siedlecki, P. Development and Evaluation of a Deep Learning Model for Protein–Ligand Binding Affinity Prediction. *Bioinformatics* **2018**, *34*, 3666–3674, <https://doi.org/10.1093/bioinformatics/bty374> (accessed 03/05/2023).
- (270) Muegge, I.; Martin, Y. C. A General and Fast Scoring Function for Protein–Ligand Interactions: A Simplified Potential Approach. *Journal of Medicinal Chemistry* **1999**, *42*, 791–804.
- (271) Trott, O.; Olson, A. J. AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading. *Journal of Computational Chemistry* **2010**, *31*, 455–461, <https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21334> (accessed 03/05/2023).
- (272) Morris, G. M.; Huey, R.; Lindstrom, W.; Sanner, M. F.; Belew, R. K.; Goodsell, D. S.; Olson, A. J. AutoDock4 and AutoDockTools4: Automated Docking with Selective Receptor Flexibility. *Journal of computational chemistry* **2009**, *30*, 2785–2791, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2760638/> (accessed 03/05/2023).
- (273) Jones-Hertzog, D. K.; Jorgensen, W. L. Binding Affinities for Sulfonamide Inhibitors with Human Thrombin Using Monte Carlo Simulations with a Linear Response Method. *Journal of Medicinal Chemistry* **1997**, *40*, 1539–1549.

- (274) Jones, G.; Willett, P.; Glen, R. C. Molecular Recognition of Receptor Sites Using a Genetic Algorithm with a Description of Desolvation. *Journal of Molecular Biology* **1995**, *245*, 43–53.
- (275) Shin, W.-H.; Kim, J.-K.; Kim, D.-S.; Seok, C. GalaxyDock2: Protein-Ligand Docking Using Beta-Complex and Global Optimization. *Journal of Computational Chemistry* **2013**, *34*, 2647–2656.
- (276) Shin, W.-H.; Seok, C. GalaxyDock: Protein-Ligand Docking with Flexible Protein Side-Chains. *Journal of Chemical Information and Modeling* **2012**, *52*, 3225–3232.
- (277) Li, H.; Peng, J.; Sidorov, P.; Leung, Y.; Leung, K.-S.; Wong, M.-H.; Lu, G.; Ballester, P. J. Classical Scoring Functions for Docking Are Unable to Exploit Large Volumes of Structural and Interaction Data. *Bioinformatics (Oxford, England)* **2019**, *35*, 3989–3995.
- (278) Eldridge, M. D.; Murray, C. W.; Auton, T. R.; Paolini, G. V.; Mee, R. P. Empirical Scoring Functions: I. The Development of a Fast Empirical Scoring Function to Estimate the Binding Affinity of Ligands in Receptor Complexes. *Journal of Computer-Aided Molecular Design* **1997**, *11*, 425–445.
- (279) Verdonk, M. L.; Cole, J. C.; Hartshorn, M. J.; Murray, C. W.; Taylor, R. D. Improved Protein-Ligand Docking Using GOLD. *Proteins* **2003**, *52*, 609–623.
- (280) Quiroga, R.; Villarreal, M. A. Vinardo: A Scoring Function Based on Autodock Vina Improves Scoring, Docking, and Virtual Screening. *PloS One* **2016**, *11*, e0155183.
- (281) Guedes, I. A.; Barreto, A.; Marinho, D.; Krempser, E.; Kuenemann, M. A.; Sperandio, O.; Dardenne, L. E.; Miteva, M. A. New Machine Learning and Physics-Based Scoring Functions for Drug Discovery. *Scientific reports* **2021**, *11*, 1–19.
- (282) de Magalhães, C. S.; Almeida, D. M.; Barbosa, H. J. C.; Dardenne, L. E. A Dynamic Niching Genetic Algorithm Strategy for Docking Highly Flexible Ligands. *Information Sciences* **2014**, *289*, 206–224.

- (283) Pan, X.; Wang, H.; Zhang, Y.; Wang, X.; Li, C.; Ji, C.; Zhang, J. Z. AA-score: A New Scoring Function Based on Amino Acid-Specific Interaction for Molecular Docking. *Journal of Chemical Information and Modeling* **2022**, *62*, 2499–2509.
- (284) Muegge, I.; Martin, Y. C. A General and Fast Scoring Function for Protein-Ligand Interactions: A Simplified Potential Approach. *Journal of Medicinal Chemistry* **1999**, *42*, 791–804.
- (285) DeWitte, R. S.; Shakhnovich, E. I. SMOG: De Novo Design Method Based on Simple, Fast, and Accurate Free Energy Estimates. 1. Methodology and Supporting Evidence. *Journal of the American Chemical Society* **1996**, *118*, 11733–11744.
- (286) DeWitte, R. S.; Ishchenko, A. V.; Shakhnovich, E. I. SMOG: De Novo Design Method Based on Simple, Fast, and Accurate Free Energy Estimates. 2. Case Studies in Molecular Design. *Journal of the American Chemical Society* **1997**, *119*, 4608–4617.
- (287) Dittrich, J.; Schmidt, D.; Pfleger, C.; Gohlke, H. Converging a Knowledge-Based Scoring Function: DrugScore2018. *Journal of Chemical Information and Modeling* **2019**, *59*, 509–521.
- (288) Gohlke, H.; Hendlich, M.; Klebe, G. Knowledge-Based Scoring Function to Predict Protein-Ligand Interactions. *Journal of Molecular Biology* **2000**, *295*, 337–356.
- (289) Neudert, G.; Klebe, G. DSX: A Knowledge-Based Scoring Function for the Assessment of Protein-Ligand Complexes. *Journal of Chemical Information and Modeling* **2011**, *51*, 2731–2745.
- (290) Velec, H. F. G.; Gohlke, H.; Klebe, G. DrugScore(CSD)-Knowledge-Based Scoring Function Derived from Small Molecule Crystal Data with Superior Recognition Rate of near-Native Ligand Poses and Better Affinity Prediction. *Journal of Medicinal Chemistry* **2005**, *48*, 6296–6303.
- (291) Huang, S.-Y.; Zou, X. An Iterative Knowledge-Based Scoring Function to Predict Protein-Ligand Interactions: I. Derivation of Interaction Potentials. *Journal of Computational Chemistry* **2006**, *27*, 1866–1875.

- (292) Huang, S.-Y.; Zou, X. An Iterative Knowledge-Based Scoring Function to Predict Protein-Ligand Interactions: II. Validation of the Scoring Function. *Journal of Computational Chemistry* **2006**, *27*, 1876–1882.
- (293) Zheng, Z.; Merz, K. M. Development of the Knowledge-Based and Empirical Combined Scoring Algorithm (KECSA) to Score Protein-Ligand Interactions. *Journal of Chemical Information and Modeling* **2013**, *53*, 1073–1083.
- (294) Yang, C.-Y.; Wang, R.; Wang, S. M-Score: A Knowledge-Based Potential Scoring Function Accounting for Protein Atom Mobility. *Journal of medicinal chemistry* **2006**, *49*, 5903–5911.
- (295) Chen, P.; Ke, Y.; Lu, Y.; Du, Y.; Li, J.; Yan, H.; Zhao, H.; Zhou, Y.; Yang, Y. DLI-GAND2: An Improved Knowledge-Based Energy Function for Protein–Ligand Interactions Using the Distance-Scaled, Finite, Ideal-Gas Reference State. *Journal of cheminformatics* **2019**, *11*, 1–11.
- (296) Kadukova, M.; Machado, K. D. S.; Chacón, P.; Grudinin, S. KORP-PL: A Coarse-Grained Knowledge-Based Scoring Function for Protein-Ligand Interactions. *Bioinformatics (Oxford, England)* **2021**, *37*, 943–950.
- (297) Meli, R.; Morris, G. M.; Biggin, P. C. Scoring Functions for Protein-Ligand Binding Affinity Prediction Using Structure-based Deep Learning: A Review. *Frontiers in Bioinformatics* **2022**, *2*, <https://www.frontiersin.org/articles/10.3389/fbinf.2022.885983> (accessed 03/06/2023).
- (298) Maguire, J. B.; Haddox, H. K.; Strickland, D.; Halabiya, S. F.; Coventry, B.; Griffin, J. R.; Pulavarti, S. V. S. R. K.; Cummins, M.; Thieker, D. F.; Klavins, E.; Szyperski, T.; DiMaio, F.; Baker, D.; Kuhlman, B. Perturbing the Energy Landscape for Improved Packing during Computational Protein Design. *Proteins* **2021**, *89*, 436–449.
- (299) Zhang, H.; Saravanan, K. M.; Lin, J.; Liao, L.; Ng, J. T.-Y.; Zhou, J.; Wei, Y. Deep-BindPoc: A Deep Learning Method to Rank Ligand Binding Pockets Using Molecular Vector Representation. *PeerJ* **2020**, *8*, e8864, <https://peerj.com/articles/8864> (accessed 03/06/2023).

- (300) Zhu, F.; Zhang, X.; Allen, J. E.; Jones, D.; Lightstone, F. C. Binding Affinity Prediction by Pairwise Function Based on Neural Network. *Journal of Chemical Information and Modeling* **2020**, *60*, 2766–2772.
- (301) Li, Y.; Liu, Z.; Li, J.; Han, L.; Liu, J.; Zhao, Z.; Wang, R. Comparative Assessment of Scoring Functions on an Updated Benchmark: 1. Compilation of the Test Set. *Journal of Chemical Information and Modeling* **2014**, *54*, 1700–1716.
- (302) Behler, J. Neural Network Potential-Energy Surfaces in Chemistry: A Tool for Large-Scale Simulations. *Physical Chemistry Chemical Physics* **2011**, *13*, 17930–17955, <https://pubs.rsc.org/en/content/articlelanding/2011/cp/c1cp21668f> (accessed 03/06/2023).
- (303) Bartók, A. P.; De, S.; Poelking, C.; Bernstein, N.; Kermode, J. R.; Csányi, G.; Michele Ceriotti Machine Learning Unifies the Modeling of Materials and Molecules. *Science Advances* **2017**, *3*, e1701816, eprint: <https://www.science.org/doi/pdf/10.1126/sciadv.1701816>, <https://www.science.org/doi/abs/10.1126/sciadv.1701816>.
- (304) Bartók, A. P.; Kondor, R.; Csányi, G. On Representing Chemical Environments. *Physical Review B* **2013**, *87*, 184115, <https://link.aps.org/doi/10.1103/PhysRevB.87.184115>.
- (305) Bartók, A. P.; Payne, M. C.; Kondor, R.; Csányi, G. Gaussian Approximation Potentials: The Accuracy of Quantum Mechanics, without the Electrons. *Physical Review Letters* **2010**, *104*, 136403, <https://link.aps.org/doi/10.1103/PhysRevLett.104.136403>.
- (306) Behler, J.; Parrinello, M. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Physical Review Letters* **2007**, *98*, 146401, <https://link.aps.org/doi/10.1103/PhysRevLett.98.146401>.
- (307) Schütt, K. T.; Kindermans, P.-J.; Sauceda, H. E.; Chmiela, S.; Tkatchenko, A.; Müller, K.-R. SchNet: A Continuous-Filter Convolutional Neural Network for Modeling Quantum Interactions, 2017, arXiv: 1706.08566 [physics, stat], <http://arxiv.org/abs/1706.08566> (accessed 03/06/2023).

- (308) 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* **2017**, *8*, 3192–3203, <https://pubs.rsc.org/en/content/articlelanding/2017/sc/c6sc05720a> (accessed 03/06/2023).
- (309) 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* **2018**, *148*, 241733, <https://aip.scitation.org/doi/10.1063/1.5023802> (accessed 03/06/2023).
- (310) 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* **2020**, *60*, 3408–3415, <https://doi.org/10.1021/acs.jcim.0c00451> (accessed 03/06/2023).
- (311) Chai, J.-D.; Head-Gordon, M. Systematic Optimization of Long-Range Corrected Hybrid Density Functionals. *The Journal of Chemical Physics* **2008**, *128*, 084106, <https://aip.scitation.org/doi/10.1063/1.2834918> (accessed 03/06/2023).
- (312) Ditchfield, R.; Hehre, W. J.; Pople, J. A. Self-Consistent Molecular-Orbital Methods. IX. An Extended Gaussian-Type Basis for Molecular-Orbital Studies of Organic Molecules. *The Journal of Chemical Physics* **1971**, *54*, 724–728, <https://aip.scitation.org/doi/10.1063/1.1674902> (accessed 03/06/2023).
- (313) Fink, T.; Reymond, J.-L. Virtual Exploration of the Chemical Universe up to 11 Atoms of C, N, O, F: Assembly of 26.4 Million Structures (110.9 Million Stereoisomers) and Analysis for New Ring Systems, Stereochemistry, Physicochemical Properties, Compound Classes, and Drug Discovery. *Journal of Chemical Information and Modeling* **2007**, *47*, 342–353, <https://doi.org/10.1021/ci600423u> (accessed 03/06/2023).
- (314) Fink, T.; Bruggesser, H.; Reymond, J.-L. Virtual Exploration of the Small-Molecule Chemical Universe below 160 Daltons. *Angewandte Chemie International Edition* **2005**, *44*, 1504–1508, <https://onlinelibrary.wiley.com/doi/abs/10.1002/anie.200462457> (accessed 03/06/2023).

- (315) Devereux, C.; Smith, J. S.; Huddleston, K. K.; Barros, K.; Zubatyuk, R.; Isayev, O.; Roitberg, A. E. Extending the Applicability of the ANI Deep Learning Molecular Potential to Sulfur and Halogens. *Journal of Chemical Theory and Computation* **2020**, *16*, 4192–4202, <https://doi.org/10.1021/acs.jctc.0c00121> (accessed 03/06/2023).
- (316) Ryckaert, J.-P.; Ciccotti, G.; Berendsen, H. J. Numerical Integration of the Cartesian Equations of Motion of a System with Constraints: Molecular Dynamics of n-Alkanes. *Journal of computational physics* **1977**, *23*, 327–341.
- (317) Smith, J. S.; Isayev, O.; Roitberg, A. E. ANI-1, A Data Set of 20 Million Calculated off-Equilibrium Conformations for Organic Molecules. *Scientific Data* **2017**, *4*, 170193, <https://www.nature.com/articles/sdata2017193> (accessed 04/27/2023).
- (318) Su, M.; Feng, G.; Liu, Z.; Li, Y.; Wang, R. Tapping on the Black Box: How Is the Scoring Power of a Machine-Learning Scoring Function Dependent on the Training Set? *Journal of Chemical Information and Modeling* **2020**, *60*, 1122–1136, <https://doi.org/10.1021/acs.jcim.9b00714> (accessed 03/06/2023).
- (319) Lipman, D. J.; Pearson, W. R. Rapid and Sensitive Protein Similarity Searches. *Science* **1985**, *227*, 1435–1441, <https://www.science.org/doi/10.1126/science.2983426> (accessed 03/06/2023).
- (320) Roe, D. R.; Cheatham, T. E. I. PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data. *Journal of Chemical Theory and Computation* **2013**, *9*, 3084–3095, <https://doi.org/10.1021/ct400341p> (accessed 03/06/2023).
- (321) Yeturu, K.; Chandra, N. PocketMatch: A New Algorithm to Compare Binding Sites in Protein Structures. *BMC Bioinformatics* **2008**, *9*, 543, <https://doi.org/10.1186/1471-2105-9-543> (accessed 03/06/2023).
- (322) Wallach, I.; Heifets, A. Most Ligand-Based Classification Benchmarks Reward Memorization Rather than Generalization. *Journal of Chemical Information and Modeling* **2018**, *58*, 916–932, <https://doi.org/10.1021/acs.jcim.7b00403> (accessed 03/06/2023).

- (323) Sehnal, D.; Bittrich, S.; Deshpande, M.; Svobodová, R.; Berka, K.; Bazgier, V.; Velankar, S.; Burley, S. K.; Koča, J.; Rose, A. S. Mol* Viewer: Modern Web App for 3D Visualization and Analysis of Large Biomolecular Structures. *Nucleic Acids Research* **2021**, *49*, W431–W437, <https://doi.org/10.1093/nar/gkab314> (accessed 03/07/2023).
- (324) Zhu, F.; Zhang, X.; Allen, J. E.; Jones, D.; Lightstone, F. C. Binding Affinity Prediction by Pairwise Function Based on Neural Network. *Journal of Chemical Information and Modeling* **2020**, *60*, 2766–2772, <https://doi.org/10.1021/acs.jcim.0c00026> (accessed 03/07/2023).
- (325) Chen, L.; Cruz, A.; Ramsey, S.; Dickson, C. J.; Duca, J. S.; Hornak, V.; Koes, D. R.; Kurtzman, T. Hidden Bias in the DUD-E Dataset Leads to Misleading Performance of Deep Learning in Structure-Based Virtual Screening. *PLOS ONE* **2019**, *14*, e0220113, <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0220113> (accessed 03/07/2023).
- (326) Francoeur, P. G.; Masuda, T.; Sunseri, J.; Jia, A.; Iovanisci, R. B.; Snyder, I.; Koes, D. R. Three-Dimensional Convolutional Neural Networks and a Cross-Docked Data Set for Structure-Based Drug Design. *Journal of Chemical Information and Modeling* **2020**, *60*, 4200–4215.
- (327) Sieg, J.; Flachsenberg, F.; Rarey, M. In Need of Bias Control: Evaluating Chemical Data for Machine Learning in Structure-Based Virtual Screening. *Journal of Chemical Information and Modeling* **2019**, *59*, 947–961, <https://doi.org/10.1021/acs.jcim.8b00712> (accessed 03/07/2023).
- (328) Wallach, I.; Heifets, A. Most Ligand-Based Classification Benchmarks Reward Memorization Rather than Generalization. *Journal of Chemical Information and Modeling* **2018**, *58*, 916–932, <https://doi.org/10.1021/acs.jcim.7b00403> (accessed 03/07/2023).
- (329) Goodsell, D. S.; Autin, L.; Olson, A. J. Illustrate: Software for Biomolecular Illustration. *Structure* **2019**, *27*, 1716–1720.e1, <https://www.sciencedirect.com/science/article/pii/S0969212619302825> (accessed 03/07/2023).

- (330) Weiser, J.; Shenkin, P. S.; Still, W. C. Approximate atomic surfaces from linear combinations of pairwise overlaps (LCPO). *Journal of Computational Chemistry* **1999**, *20*, 217–230, <https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291096-987X%2819990130%2920%3A2%3C217%3A%3AAID-JCC4%3E3.0.CO%3B2-A> (accessed 03/07/2023).
- (331) He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, 2015, arXiv: 1502.01852 [cs], <http://arxiv.org/abs/1502.01852> (accessed 07/23/2023).
- (332) Snoek, J.; Larochelle, H.; Adams, R. P. Practical Bayesian Optimization of Machine Learning Algorithms, 2012, arXiv: 1206.2944 [cs, stat], <http://arxiv.org/abs/1206.2944> (accessed 03/09/2023).
- (333) Ericksen, S. S.; Wu, H.; Zhang, H.; Michael, L. A.; Newton, M. A.; Hoffmann, F. M.; Wildman, S. A. Machine Learning Consensus Scoring Improves Performance Across Targets in Structure-Based Virtual Screening. *Journal of Chemical Information and Modeling* **2017**, *57*, 1579–1590, <https://doi.org/10.1021/acs.jcim.7b00153> (accessed 03/07/2023).
- (334) Ganaie, M. A.; Hu, M.; Malik, A. K.; Tanveer, M.; Suganthan, P. N. Ensemble Deep Learning: A Review. *Engineering Applications of Artificial Intelligence* **2022**, *115*, 105151, <https://www.sciencedirect.com/science/article/pii/S095219762200269X> (accessed 07/23/2023).
- (335) Rose, A. S.; Bradley, A. R.; Valasatava, Y.; Duarte, J. M.; Prlić, A.; Rose, P. W. NGL Viewer: Web-Based Molecular Graphics for Large Complexes. *Bioinformatics* **2018**, *34*, 3755–3758, <https://doi.org/10.1093/bioinformatics/bty419> (accessed 03/31/2023).
- (336) Boyles, F.; Deane, C. M.; Morris, G. M. Learning from the Ligand: Using Ligand-Based Features to Improve Binding Affinity Prediction. *Bioinformatics* **2020**, *36*, 758–764, <https://doi.org/10.1093/bioinformatics/btz665> (accessed 03/07/2023).

- (337) Afifi, K.; Al-Sadek, A. F. Improving Classical Scoring Functions Using Random Forest: The Non-Additivity of Free Energy Terms' Contributions in Binding. *Chemical Biology & Drug Design* **2018**, *92*, 1429–1434, <https://onlinelibrary.wiley.com/doi/abs/10.1111/cbdd.13206> (accessed 03/07/2023).
- (338) Ballester, P. J.; Mitchell, J. B. O. A Machine Learning Approach to Predicting Protein–Ligand Binding Affinity with Applications to Molecular Docking. *Bioinformatics* **2010**, *26*, 1169–1175, <https://doi.org/10.1093/bioinformatics/btq112> (accessed 03/07/2023).
- (339) Durrant, J. D.; McCammon, J. A. NNScore 2.0: A Neural-Network Receptor–Ligand Scoring Function. *Journal of Chemical Information and Modeling* **2011**, *51*, 2897–2903, <https://doi.org/10.1021/ci2003889> (accessed 03/07/2023).
- (340) Li, Y.; Rezaei, M. A.; Li, C.; Li, X.; Wu, D. DeepAtom: A Framework for Protein–Ligand Binding Affinity Prediction, 2019, arXiv: 1912.00318 [cs, q-bio], <http://arxiv.org/abs/1912.00318> (accessed 07/23/2023).
- (341) Kwon, Y.; Shin, W.-H.; Ko, J.; Lee, J. AK-Score: Accurate Protein-Ligand Binding Affinity Prediction Using the Ensemble of 3D-Convolutional Neural Network, 2020, <https://chemrxiv.org/engage/chemrxiv/article-details/60c7490a4c8919d4f9ad3012> (accessed 03/07/2023).
- (342) Nguyen, D. D.; Wei, G.-W. AGL-Score: Algebraic Graph Learning Score for Protein–Ligand Binding Scoring, Ranking, Docking, and Screening. *Journal of Chemical Information and Modeling* **2019**, *59*, 3291–3304, <https://doi.org/10.1021/acs.jcim.9b00334> (accessed 03/07/2023).
- (343) Koes, D. R.; Baumgartner, M. P.; Camacho, C. J. Lessons Learned in Empirical Scoring with Smina from the CSAR 2011 Benchmarking Exercise. *Journal of Chemical Information and Modeling* **2013**, *53*, 1893–1904, <https://doi.org/10.1021/ci300604z> (accessed 03/07/2023).

Appendix A

Appendix

A.1 Expectation-Maximization Algorithm for SentencePiece

The SentencePiece training loss is defined as the likely reduction in marginal log-likelihood of the training corpus should the subword be removed. The marginal log-likelihood is defined as follows:

$$\mathcal{L} = \sum_{s=1}^{|D|} \log P(X^s) = \sum_{s=1}^{|D|} \log \left(\sum_{\mathbf{x} \in S(\mathbf{x})} P(\mathbf{x}) \right) \quad (\text{A.1})$$

where D is the training corpus, $|D|$ is the number of sentences in the corpus, $S(\mathbf{x})$ is the set of all possible segmentations/sequences $\mathbf{x}$, and $P(\mathbf{x})$ are the unigram sequences and probabilities. Given that these are all hidden variables (since we only have the un-tokenized training corpus), expectation-maximization (EM) is used to find the maximum likelihood estimate given the unknowns. In this subword regularisation algorithm however, the EM steps are not performed in the traditional way. Instead, the EM algorithm is used to find the maximum likelihood estimate of the unigram sequences and probabilities given the unknown segmentations (i.e it is

done in reverse). The algorithm is as follows:

1. Initialize the unigram sequences and probabilities $P(\mathbf{x})$ randomly.
2. For each sentence s in the corpus, find the most likely segmentation $\mathbf{x}^s$ using the current $P(\mathbf{x})$.
3. Update the unigram sequences and probabilities $P(\mathbf{x})$ using the current $\mathbf{x}^s$.
4. Repeat steps 2 and 3 until convergence.

During training it is theoretically expected for the log-likelihood to monotonically increase then plateau.

A.2 Finding Optimal Segmentations

The Viterbi algorithm is useful when given hidden variables and a set of possible states. In this case, the hidden variables are the unigram sequences and probabilities $P(\mathbf{x})$ and the possible states are the segmentations $\mathbf{x}$. The Viterbi algorithm can be used to find the most likely segmentation $\mathbf{x}^s$ given the unigram sequences and probabilities $P(\mathbf{x})$. The issue is not all subwords are of equal length, so the Viterbi algorithm cannot be used directly.

Consider the following example, suppose we have the following set of subwords for tokenizing the word hello:

$$\{\text{"he"}, \text{"h"}, \text{"ll"}, \text{"e"}, \text{"o"}, \text{"hell"}\}$$

A trellis-like diagram can be used to find the most likely segmentation. The diagram is a directed graph where the nodes are the characters and the edges are the possible subword segmentations. Figure A.1 a) shows the trellis-like diagram for the word hello where the transitions are unknown. Each arrow is a transition from the start-of-sentence token, considering

the arrows as a combination of the unigram probabilities from its head to its tail, the objective is to maximise this probability through a substructure search algorithm. One such algorithm is bottom-up dynamic programming, which is applied efficiently to a Trie data structure. Figure A.1 b) shows the Trie for our "hello" example:

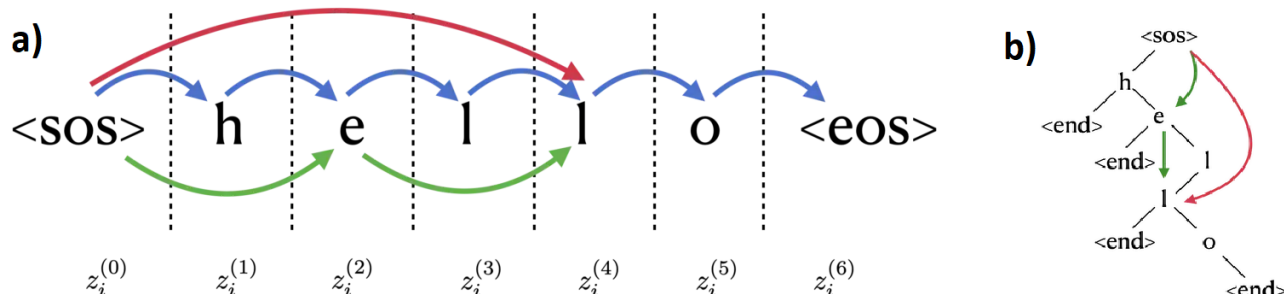


Figure A.1: a) Trellis-like diagram for the word "hello" where the transitions are unknown. b) Trie for the word "hello". Adapted from <https://towardsdatascience.com/sentencepiece-tokenizer-demystified-d0a3aac19b15>

This is a bottom-up dynamic programming algorithm, where the objective is to find the most optimal path through the graph to a particular state, p_i given all the possible paths to that state. In other words:

$$p_i = \max_{j < i} (p_j p_{j \rightarrow i}) \quad (\text{A.2})$$

further to finding the individual optimal paths, SentencePiece also finds multiple optimal paths and performs subword sampling by taking the top n paths. To ensure sampling results in good quality tokenization the Forward Filtering Backwards Sampling (FFBS) algorithm is used. This completes the subword regularisation algorithm.

A.3 RDKit's benchmarking platform filtered target set and MFBERT scores

The RDKit benchmarking platform filtered target set is a set of 69 targets that are filtered from the original set based on difficulty. Table A.2 shows the filtered target set and the MFBERT

scores for each target. The final MFBERT benchmark scores are the average of the scores for each of the 69 targets.

AUCROC is the area under the receiver operating characteristic curve, which is a plot of the true positive rate against the false positive rate. BEDROC20 is the Boltzmann-enhanced discrimination of receiver operating characteristic, which is a metric that promotes early recognition of active compounds by assigning higher weights to the early part of the ROC curve. The BEDROC20 score is calculated as follows:

$$BEDROC = \frac{e^{\alpha R_a} - R_i}{e^{\alpha R_a} - 1} - \frac{R_i}{1 - e^{-\alpha R_i}} \quad (\text{A.3})$$

where R_a is the ratio of actives, $R_a = \frac{n}{N}$, R_i is the ratio of inactives $R_i = \frac{N-n}{N}$, and N is the total number of compounds. $\alpha = 20$ is a parameter that controls the degree of early recognition. In this case the top 20% of the ranked list is weighed more heavily than the bottom 80%.

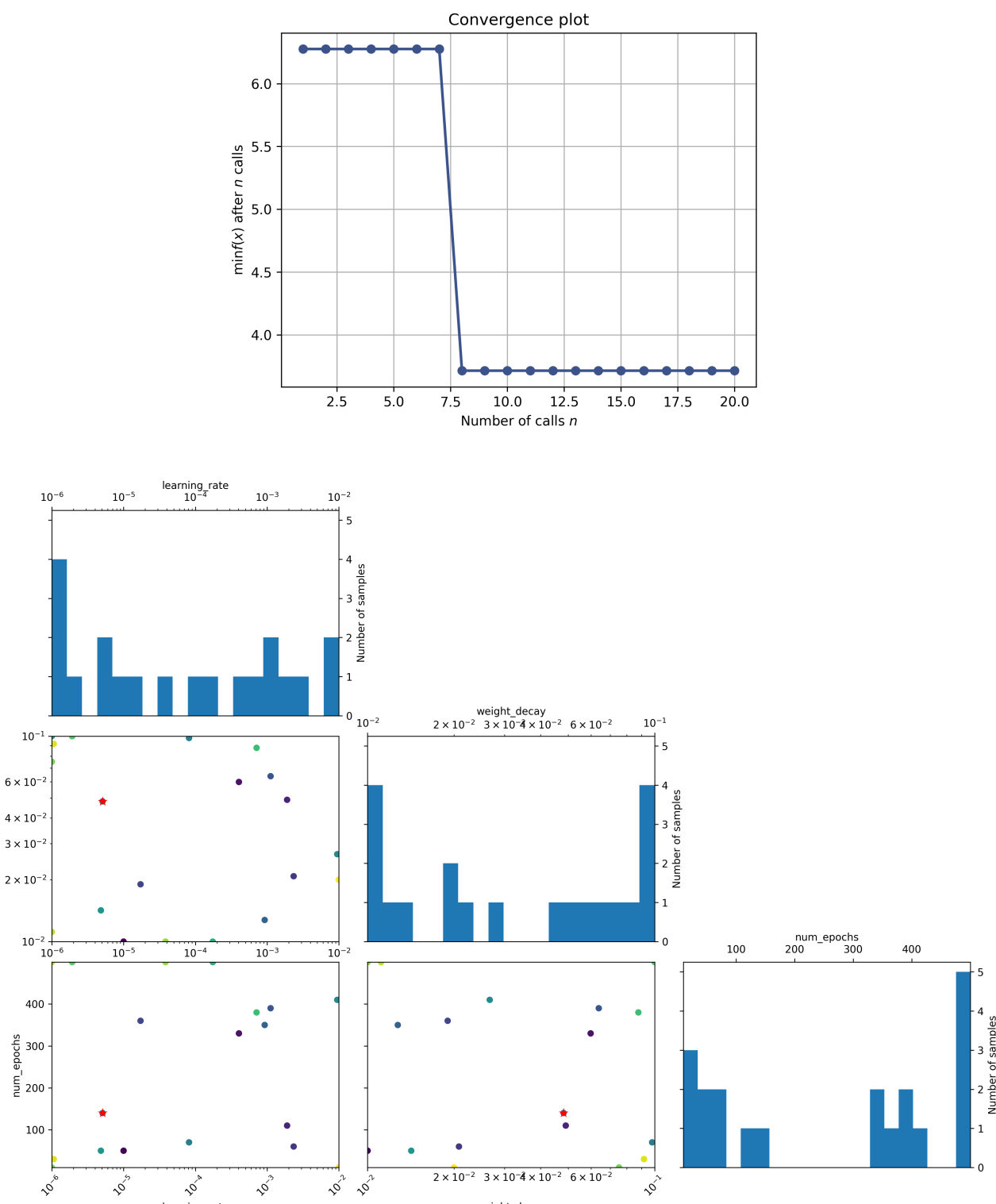
Target	AUCROC	BEDROC20
ChEMBL_target_100	0.99 +/- 0.028	0.974 +/- 0.039
ChEMBL_target_100579	0.975 +/- 0.045	0.945 +/- 0.073
ChEMBL_target_10188	0.966 +/- 0.091	0.937 +/- 0.148
ChEMBL_target_10193	0.999 +/- 0.002	0.994 +/- 0.008
ChEMBL_target_10260	0.967 +/- 0.054	0.925 +/- 0.095
ChEMBL_target_10280	0.96 +/- 0.118	0.932 +/- 0.158
ChEMBL_target_10378	0.999 +/- 0.005	0.994 +/- 0.011
ChEMBL_target_104	0.786 +/- 0.293	0.67 +/- 0.33
ChEMBL_target_10434	0.994 +/- 0.02	0.982 +/- 0.027
ChEMBL_target_10498	1.0 +/- 0.0	0.999 +/- 0.002
ChEMBL_target_105	1.0 +/- 0.0	0.999 +/- 0.001
ChEMBL_target_107	0.972 +/- 0.059	0.937 +/- 0.089
ChEMBL_target_108	0.926 +/- 0.153	0.87 +/- 0.192
ChEMBL_target_10980	0.983 +/- 0.042	0.969 +/- 0.056
ChEMBL_target_11140	1.0 +/- 0.0	0.998 +/- 0.002
ChEMBL_target_11359	0.958 +/- 0.087	0.931 +/- 0.099
ChEMBL_target_11365	0.975 +/- 0.06	0.952 +/- 0.076
ChEMBL_target_114	0.96 +/- 0.089	0.922 +/- 0.135
ChEMBL_target_11489	0.91 +/- 0.182	0.837 +/- 0.224
ChEMBL_target_11534	0.998 +/- 0.006	0.994 +/- 0.016
ChEMBL_target_11575	0.993 +/- 0.019	0.981 +/- 0.03
ChEMBL_target_11631	0.992 +/- 0.018	0.977 +/- 0.036
ChEMBL_target_121	0.973 +/- 0.082	0.952 +/- 0.109
ChEMBL_target_12209	0.961 +/- 0.108	0.934 +/- 0.143
ChEMBL_target_12252	0.987 +/- 0.038	0.977 +/- 0.045
ChEMBL_target_12261	0.992 +/- 0.015	0.981 +/- 0.02
ChEMBL_target_126	0.876 +/- 0.219	0.807 +/- 0.229
ChEMBL_target_12670	0.99 +/- 0.02	0.963 +/- 0.039
ChEMBL_target_12911	0.939 +/- 0.074	0.882 +/- 0.097
ChEMBL_target_12952	0.987 +/- 0.081	0.969 +/- 0.127
ChEMBL_target_130	0.999 +/- 0.002	0.994 +/- 0.009
ChEMBL_target_13001	0.98 +/- 0.055	0.95 +/- 0.069
ChEMBL_target_15	0.999 +/- 0.003	0.994 +/- 0.012
ChEMBL_target_165	0.968 +/- 0.117	0.952 +/- 0.15
ChEMBL_target_17045	0.985 +/- 0.052	0.959 +/- 0.066
ChEMBL_target_19905	0.999 +/- 0.007	0.994 +/- 0.011
ChEMBL_target_219	0.975 +/- 0.091	0.96 +/- 0.11
ChEMBL_target_25	0.999 +/- 0.004	0.994 +/- 0.011
ChEMBL_target_259	0.996 +/- 0.013	0.985 +/- 0.026
ChEMBL_target_36	0.996 +/- 0.01	0.988 +/- 0.019
ChEMBL_target_43	0.732 +/- 0.225	0.585 +/- 0.237
ChEMBL_target_51	0.996 +/- 0.006	0.983 +/- 0.028
ChEMBL_target_52	0.997 +/- 0.006	0.988 +/- 0.017
ChEMBL_target_61	0.997 +/- 0.008	0.99 +/- 0.027
ChEMBL_target_65	0.971 +/- 0.06	0.934 +/- 0.094
ChEMBL_target_72	0.995 +/- 0.013	0.989 +/- 0.019
ChEMBL_target_8	0.989 +/- 0.031	0.971 +/- 0.038
ChEMBL_target_87	0.934 +/- 0.129	0.888 +/- 0.154
ChEMBL_target_90	0.996 +/- 0.013	0.985 +/- 0.029
ChEMBL_target_93	0.989 +/- 0.041	0.967 +/- 0.056

DUD_target_cdk2	0.989 +/- 0.03	0.966 +/- 0.066
DUD_target_hivrt	0.883 +/- 0.101	0.8 +/- 0.141
DUD_target_vegfr2	0.974 +/- 0.063	0.922 +/- 0.136
MUV_target_466	0.542 +/- 0.13	0.199 +/- 0.107
MUV_target_548	0.799 +/- 0.302	0.574 +/- 0.272
MUV_target_600	0.378 +/- 0.227	0.132 +/- 0.151
MUV_target_644	0.505 +/- 0.257	0.256 +/- 0.209
MUV_target_652	0.771 +/- 0.127	0.419 +/- 0.156
MUV_target_689	0.45 +/- 0.287	0.209 +/- 0.199
MUV_target_692	0.385 +/- 0.17	0.094 +/- 0.084
MUV_target_712	0.637 +/- 0.251	0.436 +/- 0.23
MUV_target_713	0.387 +/- 0.232	0.148 +/- 0.136
MUV_target_733	0.422 +/- 0.218	0.17 +/- 0.122
MUV_target_737	0.736 +/- 0.14	0.464 +/- 0.159
MUV_target_810	0.653 +/- 0.257	0.335 +/- 0.177
MUV_target_832	0.547 +/- 0.23	0.318 +/- 0.223
MUV_target_852	0.868 +/- 0.134	0.65 +/- 0.209
MUV_target_858	0.39 +/- 0.31	0.185 +/- 0.23
MUV_target_859	0.537 +/- 0.263	0.274 +/- 0.229

Figure A.2: RDKit's benchmarking platform filtered target set and MFBERT scores.

A.4 Meta Optimization of Hyperparameters

Below are the convergence, objective and evaluation plots for the meta-optimization of (some of) the hyperparameters of the MFBERT-ANI combination model. The hyperparameters are the learning rate, weight decay constant and the number of epochs. The hyperparameters are optimized using the Bayesian Optimization algorithm, which is a sequential model-based optimization algorithm that uses a Gaussian process to model the objective function.



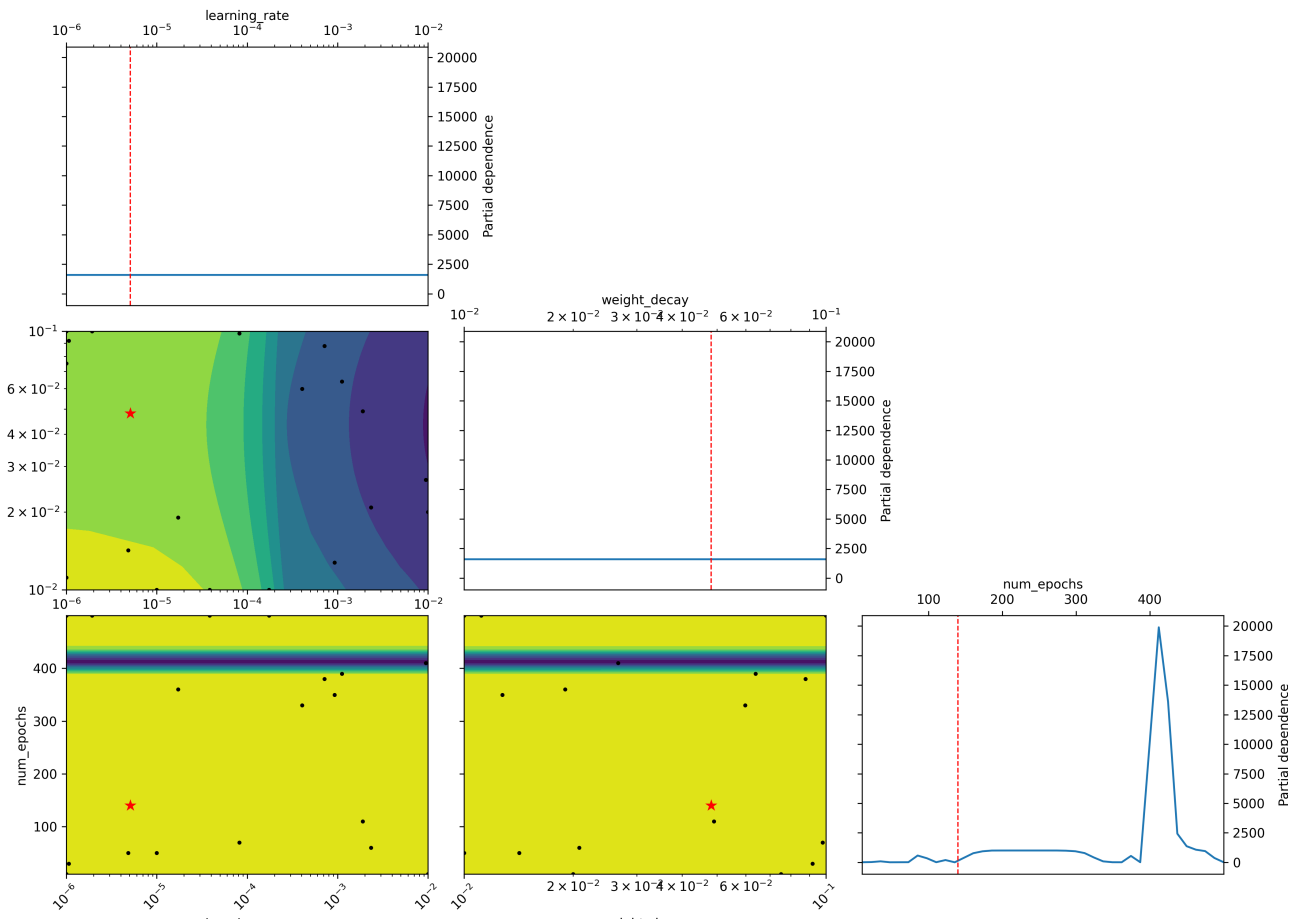


Figure A.3: Convergence, evaluation and objective plots for the meta-optimization of the hyperparameters of the MFBERT-ANI combination model.

A.5 Copyright Licenses and Permissions

Permission to reproduce figure 1.6.

 IUCr Journals

Terms and conditions of use of the article titled
Processing two-dimensional X-ray diffraction and small-angle scattering data in *DAWN 2*

[Filik *et al.* (2017). *J. Appl. Cryst.* **50**, 959-966
<https://doi.org/10.1107/S1600576717004708>]

Permission to reproduce or reuse in whole or part the above article is granted to

Hisham Abdel Aty

provided that the reused material is accompanied by an attribution statement according the terms of the open-access licence under which the original article was published, and that permission has been obtained from any other organizations or individuals where any parts of the material are subject to explicit statements of copyright or prior reproduction permission from such third parties.

 Peter Strickland
Executive Managing Editor, IUCr Journals

Permission to reproduce figure 3.3.

25/03/2023, 22:52

RightsLink Printable License

ELSEVIER LICENSE
TERMS AND CONDITIONS

Mar 25, 2023

This Agreement between Mr. Hisham Abdel Aty ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

License Number	5516160526055
License date	Mar 25, 2023
Licensed Content Publisher	Elsevier
Licensed Content Publication	Elsevier Books
Licensed Content Title	Handbook of Biological Physics
Licensed Content Author	J.M. Seddon,R.H. Templer
Licensed Content Date	Jan 1, 1995
Licensed Content Pages	64
Start Page	97
End Page	160
Type of Use	reuse in a thesis/dissertation
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	both print and electronic

Permission to reproduce figure 5.1.

26/03/2023, 00:28 Rightslink® by Copyright Clearance Center



TorchANI: A Free and Open Source PyTorch-Based Deep Learning Implementation of the ANI Neural Network Potentials

Author: Xiang Gao, Farhad Ramezanghorbani, Olexandr Isayev, et al
Publication: Journal of Chemical Information and Modeling
Publisher: American Chemical Society
Date: Jul 1, 2020

Copyright © 2020, American Chemical Society

PERMISSION/LICENSE IS GRANTED FOR YOUR ORDER AT NO CHARGE

This type of permission/license, instead of the standard Terms and Conditions, is sent to you because no fee is being charged for your order. Please note the following:

- Permission is granted for your request in both print and electronic formats, and translations.
- If figures and/or tables were requested, they may be adapted or used in part.
- Please print this page for your records and send a copy of it to your publisher/graduate school.
- Appropriate credit for the requested material should be given as follows: "Reprinted (adapted) with permission from {COMPLETE REFERENCE CITATION}. Copyright {YEAR} American Chemical Society." Insert appropriate information in place of the capitalized words.
- One-time permission is granted only for the use specified in your RightsLink request. No additional uses are granted (such as derivative works or other editions). For any uses, please submit a new request.

If credit is given to another source for the material you requested from RightsLink, permission must be obtained from that source.

[BACK](#) [CLOSE WINDOW](#)

© 2023 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | Privacy statement | Data Security and Privacy
| For California Residents | Terms and ConditionsComments? We would like to hear from you. E-mail us at customer care@copyright.com

<https://s100.copyright.com/AppDispatchServlet> 1/1

Permission to reproduce figure 4.1.

16/07/2023, 16:22

RightsLink - Your Account

SPRINGER NATURE LICENSE TERMS AND CONDITIONS

Jul 16, 2023

This Agreement between Mr. Hisham Abdel Aty ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

License Number	5590840250656
License date	Jul 16, 2023
Licensed Content Publisher	Springer Nature
Licensed Content Publication	Software & Systems Modeling
Licensed Content Title	Recommending metamodel concepts during modeling activities with pre-trained language models
Licensed Content Author	Martin Weyssow et al
Licensed Content Date	Feb 12, 2022
Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	print and electronic
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Will you be translating?	no
Circulation/distribution	1000 - 1999
Author of this Springer Nature content	no
Title	Contextual Representations of the Chemical Space for Task Agnostic Machine Learning Methods
Institution name	Imperial College London
Expected presentation date	Jul 2023
Portions	fig. 1
Requestor Location	Mr. Hisham Abdel Aty B.014 WoodLane studios 80 WoodLane White City London, London W12 0BZ United Kingdom Attn: Mr. Hisham Abdel Aty
Total	0.00 GBP

Terms and Conditions

Springer Nature Customer Service Centre GmbH Terms and Conditions

The following terms and conditions ("Terms and Conditions") together with the terms specified in your [RightsLink] constitute the License ("License") between you as Licensee and Springer Nature Customer Service Centre GmbH as Licensor. By clicking 'accept' and completing the transaction for your use of the material ("Licensed Material"), you confirm your acceptance of and obligation to be bound by these Terms and Conditions.

1. Grant and Scope of License

- 1.1. The Licensor grants you a personal, non-exclusive, non-transferable, non-sublicensable, revocable, world-wide License to reproduce, distribute, communicate to the public, make available, broadcast, electronically transmit or create derivative works using the Licensed Material for the purpose(s) specified in your RightsLink Licence Details only. Licenses are granted for the specific use requested in the order and for no other use, subject to these Terms and Conditions. You acknowledge and agree that the rights granted to you under this License do not include the right

All other reproduced figures are under creative common licenses that do not require additional permissions.