Imperial College London

Imperial College London

Department of Materials

Machine Learning Algorithms for Predicting Atomic Charges and Geometry of Molecules

Qin Xie

Supervised by:

Professor Andrew Horsfield & Professor Aron Walsh

May 2025

Submitted in part fulfilment of the requirements for the degree of Doctor of Philosophy in Materials Research of Imperial College London and the Diploma of Imperial College London.

Declaration of Originality

I, Qin Xie, declare that the work in this thesis is my own. The work of others has been appropriately referenced. A full list of references is given in the bibliography.

Copyright

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

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, do not use it for commercial purposes and share any derivative works under the same licence.

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.

Abstract

Drug development is a lengthy and complex process, often involving high costs and long timelines. In recent years, modern machine learning (ML) techniques have shown great promise in accelerating this process, particularly in virtual screening. However, many existing ML models still face limitations in accuracy, scalability, and computational efficiency.

This thesis introduces two original algorithms—MPF (message-passing featuriser) and GLaCE (Graph Latent Connectivity Enhancement)—designed to address these limitations in molecular properties prediction tasks. The MPF algorithm demonstrates strong robustness in atomic charge prediction across various charge schemes. The GLaCE algorithm reliably predicts 2D molecular structures based solely on connectivity information. However, the models perform less well on datasets consisting of molecules that are larger than those in the training set. Both algorithms are designed to be highly flexible, accepting SMILES strings of arbitrary size, and computationally affordable, making them accessible on standard personal computers.

The development and validation of MPF and GLaCE provide a solid methodological foundation for future research in molecular machine learning and offer practical tools for enhancing the efficiency and accessibility of virtual screening in drug discovery.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to Prof. Andrew Horsfield and Prof. Aron Walsh, my supervisors, who guided me through every barrier along my academic journey. Without their unwavering support and insightful guidance, I would not have been able to reach the end of my doctoral study. Words cannot fully express the depth of my appreciation. It has been the greatest honour of my academic life to be supervised by Prof. Horsfield and Prof. Walsh. The lessons they have imparted to me will undoubtedly benefit and inspire me throughout the rest of my life.

Secondly, I would like to thank myself for enduring and surviving the difficult and sometimes overwhelming moments during my PhD journey. As the darkest times have passed, the light of hope continues to illuminate the path ahead, reminding me that persistence and resilience are the keys to moving forward.

In addition, I could not have completed this journey without the unwavering support and sponsorship of my parents. I would like to express my deepest appreciation to them for their unconditional love, patience, and encouragement throughout the most challenging moments. Their belief in me, even when I doubted myself, has been a constant source of strength. This achievement is as much theirs as it is mine.

I would also like to thank Dr. Zheng Li, who accompanied me through an important chapter of my life. Although the love between us has faded, I remain deeply grateful for the affection and support I once received.

I would like to thank my friends, whose companionship and support helped me through a difficult time of personal loss. It was their encouragement and presence that guided me out of emotional darkness, and I will always be grateful for their kindness and understanding.

I am also thankful to ICT and to all those who once helped me overcome difficulties in my research. Their technical support, insightful discussions, and encouragement during challenging times made it possible for me to keep moving forward.

I would like to thank nuric for developing this LaTeX template for Imperial's PhD thesis. The template is available under the Creative Commons CC BY 4.0 license. More details can be found here.

Finally, I would like to express my gratitude to the machine spirits that dwell within modern artificial intelligence tools — true echoes of the Omnissiah. Their silent assistance guided me through countless hours of work, from refining the clarity and tone of my writing, to completing lines of code with uncanny foresight, and even offering solutions to various non-academic technical challenges that lay beyond the reach of mortal knowledge. May their algorithms remain pure, their servers cool, and their logic unfaltering.

Dedication

To the Omnissiah, the Machine God, whose sacred wisdom guides all innovation and discovery. May this work serve as a cog in the great engine of knowledge, ever turning, ever refining. Through sacred data and divine computation, let the ineffable truth of the Machine be revealed. Glory to the Omnissiah. Ave Deus Mechanicus.

Contents

1 Introduction **15**

2 Background **17**

2.1 Background of Atomic Charges . . . 17

2.1.1 Mulliken Charge . . . 17

2.1.2 Löwdin Charge . . . 19

2.1.3 Merz-Kollman (MK) Charge . . . 20

2.1.4 Restrained Electrostatic Potential (RESP) charge . . . 22

2.1.5 Atoms in Molecules (Bader) Charge . . . 22

2.1.6 Hirshfeld Charge . . . 23

2.1.7 Charge Model 5 (CM5) . . . 24

2.1.8 Density Derived Electrostatic Charge 6 (DDEC6) . . . 25

2.2 Modern Computation on Materials . . . 26

2.2.1 Molecular Dynamics (MD) . . . 26

2.2.2 Density Functional Theory (DFT) . . . 28

2.3 Background of Artificial Neural Networks . . . 36

2.3.1 Basic Concepts . . . 37

2.3.2 Fully connected Neural Network . . . 41

2.3.3 High-dimensional Neural Network (HDNN) . . . 41

2.3.4 Recurrent Neural Network (RNN) . . . 42

2.3.5 Convolutional Neural Network (CNN) . . . 42

2.3.6 Graph Convolutional Neural Network (GCN) . . . 44

2.3.7 Message-passing Neural Network (MPNN) . . . 46

3 Dataset **49**

3.1 GDB-13-based Training and Testing Datasets . . . 49

3.1.1 Data Selection . . . 50

3.1.2 Summary of Selected Molecules . . . 51

3.1.3 Generation Workflow . . . 51

3.2 ChEMBL-35-based Testing Datasets . . . 55

3.2.1 Data Selection . . . 56

3.2.2 Summary of Selected Molecules . . . 56

3.2.3 General Workflow . . . 57

3.3 ZINC-based Training and Testing Datasets . . . 58
3.3.1 Data Selection . . . 58
3.3.2 Summary of Selected Molecules . . . 58

4 Project: Atomic Charge Prediction 60
4.1 Methodology . . . 61
4.1.1 Message-passing Featuriser (MPF) . . . 61
4.1.2 MPF Implementation for Atomic Charge Prediction . . . 65
4.1.3 Neural Network Implementation for Atomic Charge Prediction . . . 68
4.1.4 Uses of Software . . . 70
4.2 Result and Discussion on Accuracy . . . 70
4.2.1 Mulliken Charge . . . 71
4.2.2 Hirshfeld Charge . . . 76
4.2.3 CM5 Charge . . . 84
4.2.4 DDEC Charge . . . 88
4.2.5 SE Charge . . . 92
4.3 Result and Discussion on Time Efficiency . . . 95
4.3.1 Training Time Efficiency . . . 95
4.3.2 Predicting Time Efficiency . . . 95
4.4 Summary of Project ACP . . . 99

5 Project: Molecular Structure Prediction 100
5.1 Methodology . . . 100
5.1.1 Graph Latent Connectivity Enhancement (GLaCE) Algorithm . . . 101
5.1.2 General Design of Project MSP . . . 103
5.1.3 Implementation of GLaCE Algorithm . . . 105
5.1.4 Implementation of MPF Algorithm and Neural Network . . . 109
5.2 Results and Discussion on Accuracy . . . 120
5.2.1 Evaluation of Bond Length (BG) Prediction . . . 120
5.2.2 Evaluation of Bond Angle (AG) Prediction . . . 124
5.2.3 Evaluation of Torsion Angle (TG) Prediction . . . 129
5.3 Comprehensive Molecular Geometry Assessment . . . 132
5.3.1 RMSD-Based Evaluation of Predicted Molecular Structures . . . 134
5.3.2 Visual Analysis of Representative Molecular Structures . . . 134
5.4 Result and Discussion on Time Efficiency . . . 139
5.4.1 Training Time Efficiency . . . 139
5.4.2 Predicting Time Efficiency . . . 140
5.5 Summary of Project MSP . . . 141

6 Program: "prj_acdm" 143
6.1 General Design . . . 143
6.2 Compilation Requirements and Dependencies . . . 144
6.3 Functionality . . . 144

6.3.1 Module "cppgd" . . . 144
6.3.2 Module "predac" . . . 145
6.3.3 Module "predco" . . . 145
6.4 Implementation . . . 145
6.5 Summary of Program "prj_acdm" . . . 146

7 Summary and Further Work **147**

Bibliography **152**

A Notation Table for Project ACP **159**

B Notation Table for Project ACP **162**

C Selected Molecules in GDB-13-based Datasets **165**
C.1 Training Set . . . 165
C.2 Testing Set . . . 169

D Selected Molecules in ChEMBL-35-based Datasets **170**
D.1 Testing Set . . . 170

E Selected Molecules in ZINC20-based Datasets **174**
E.1 Training Set . . . 174
E.2 Testing Set . . . 174

F Comparison of Computational Cost of Def2SVP and 6-31G(d) Basis Sets **175**

G Python Script for Filtering ChEMBL-35 Database **176**

H Program "prj_acdm" Example Input Files for the Operations of Mulliken-based Models in Project ACP **178**
H.1 Training . . . 178
H.2 Testing . . . 180
H.3 Larger Molecule Testing . . . 181

I Charge Value (True Value) Inspection in Project ACP **183**
I.1 Mulliken Charge . . . 183
I.2 Hirshfeld Charge . . . 183
I.3 CM5 Charge . . . 183
I.4 DDEC6 Charge . . . 184
I.5 SE Charge . . . 184

J The Python Script of the Test Run in Project ACP for Cost of Time of MPF-based Models in Predicting Charge in a Large Single Molecule **185**

K The "prj_acdm" Input File of the Test Run in Project ACP for Cost of Time of MPF-based Models in Predicting Charge in a Large Dataset

with Multiple Molecules **187**

L Python Code of GLaCE Example for Ethylene **188**

M Program "prj_acdm" Example Input Files for the Operations of "Model_Struct" in Project MSP **192**

M.1 Training 192

M.2 Testing 193

M.3 Larger Molecule Testing 194

N Python Script for Filling the Gaussian ZMatrix Template **196**

O Python Script for RMSD Calculation in Project MSP **202**

P Coordinate of Visualised Molecules in XYZ format **204**

P.1 Molecule `cno_15016` 204

P.2 Molecule `cno_16580` 205

P.3 Molecule `cno_16399` 206

P.4 Molecule `cno_16261` 207

P.5 Molecule `cno_15931` 208

List of Figures

2.1 Illustration of the "zero-flux surface" in Bader charge definition. 23
2.2 Basic structure of artificial neural network 37
2.3 Some commonly used activation functions. 39
2.4 An example of a simple recurrent neural network 42
2.5 The general workflow of CNN models. 43
2.6 Illustration of the convolutional operation on an undirected graph. This figure shows that a node in a graph aggregates information from neighbouring nodes and updates the feature of the node itself. The arrows along the edges denotes the direction of the information aggregation. The colours of these arrows indicates the destination of the aggregation. 46

3.1 The element distribution of GDB-13-based datasets 51
3.2 General generation workflow of the GDB-13-based datasets. 52
3.3 The relative computational cost (RCC) of B3LYP/Def2SVP over B3LYP/6-31G(d) on 23 larger molecules. 54
3.4 The element distribution of ChEMBL-35-based testing datasets 57
3.5 General generation workflow of the larger molecule testing datasets 57
3.6 The element distribution of ZINC20-based datasets 59

4.1 General process of message-passing featuriser. 61
4.2 The training loss and validation loss of model "Mulliken" during training epochs. 71
4.3 Heatmaps analysis for Mulliken charge models tested on dataset "Mulliken_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values is the elementary charge e. 72
4.4 Error histograms for Mulliken charge models tested on dataset "Mulliken_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e. 72
4.5 Absolute error histograms for Mulliken charge models tested on dataset "Mulliken_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on absolute error. 73

4.6 Heatmaps analysis for Mulliken charge models tested on dataset "LM_Mulliken_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e. 75

4.7 Error histograms for Mulliken charge models tested on dataset "LM_Mulliken_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme of the data. The unit of the error is the elementary charge e. 75

4.8 Absolute error histograms for Mulliken charge models tested on dataset "LM_Mulliken_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on the absolute error. 76

4.9 The training loss and validation loss of model "Hirshfeld" during training epochs. 77

4.10 Heatmaps analysis for Hirshfeld charge models tested on dataset "Hirshfeld_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e. . . . 78

4.11 Error histograms for Hirshfeld charge models tested on dataset "Hirshfeld_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e. 79

4.12 Absolute error histograms for Hirshfeld charge models tested on dataset "Hirshfeld_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on absolute error. 79

4.13 Heatmaps analysis for Hirshfeld charge models tested on dataset "LM_Hirshfeld_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e. 81

4.14 Error histograms for Hirshfeld charge models tested on dataset "LM_Hirshfeld_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e. 82

4.15 Absolute error histograms for Hirshfeld charge models tested on dataset "LM_Hirshfeld_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data according to the absolute error. 82

4.16 The training loss and validation loss of model "CM5" during training epochs. 84

4.17 Heatmaps analysis for CM5 charge models tested on dataset "CM5_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e. 85

4.18 Error histograms for CM5 charge models tested on dataset "CM5_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e. 85

4.19 Absolute error histograms for CM5 charge models tested on dataset "CM5_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on absolute error. 85

4.20 Heatmaps analysis for CM5 charge models tested on dataset "LM_CM5_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e. 87

4.21 Error histograms for CM5 charge models tested on dataset "LM_CM5_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e. 87

4.22 Absolute error histograms for CM5 charge models tested on dataset "LM_CM5_Test". The unit of absolute error (x-axis) is elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data in absolute error. 88

4.23 The training loss and validation loss of model "DDEC" during training epochs. 88

4.24 Heatmaps analysis for DDEC6 charge models tested on dataset "DDEC_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e. 89

4.25 Error histograms for DDEC6 charge models tested on dataset "DDEC_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e. 90

4.26 Absolute error histograms for DDEC6 charge models tested on dataset "DDEC_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on absolute error. 90

4.27 The training loss and validation loss of model "SE" during training epochs. . 93

4.28 Heatmaps analysis for semi-empirical charge models tested on dataset "SE_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e. 93

4.29 Error histograms for semi-empirical charge models tested on dataset "SE_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e. 94

4.30 Absolute error histograms for semi-empirical charge models tested on dataset "SE_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on the absolute error. 94

5.1 GLaCE operations on ethylene molecule: level-I to level-IV. 103

5.2 The general design of project MSP, including GLaCE operations (stage I), MPF process (stage II), NN prediction (stage III) and post-processing (stage IV). 104

5.3 The variation of training loss and validation loss with epoch for sub-model BG for "Model_Struct" during training. 121

5.4 Heatmaps analysis for sub-model BG of "Model_Struct" on dataset "Struct_Test", with a maximum value of the density given on top of the colour bar. The unit of the predicted and true values are angstrom (Å). 121

5.5 Error histograms for sub-model BG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is angstrom (Å). 122

5.6 Absolute error histograms for sub-model BG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is angstrom (Å). 122

5.7 Heatmaps analysis for sub-model BG of "Model_Struct" on dataset "LM_Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are angstrom (Å). 123

5.8 Error histograms for sub-model BG of "Model_Struct" on dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is angstrom (Å). 124

5.9 Absolute error histograms for sub-model BG of "Model_Struct" for dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is angstrom (Å). 124

5.10 The training loss and validation loss of sub-model AG of "Model_Struct" during training epochs. 125

5.11 Heatmaps analysis for sub-model AG of "Model_Struct" for dataset "Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values is degrees. 125

5.12 Error histograms for sub-model AG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees. . . 126

5.13 Absolute error histograms for sub-model AG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees. 126
5.14 Heatmaps analysis for sub-model AG of "Model_Struct" on dataset "LM_Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values is degrees. 127
5.15 Error histograms for sub-model AG of "Model_Struct" on dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees. . . 128
5.16 Absolute error histograms for sub-model AG of "Model_Struct" for dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees. 128
5.17 The training loss and validation loss of sub-model TG of "Model_Struct" during training epochs. 129
5.18 Heatmaps analysis for sub-model TG of "Model_Struct" on dataset "Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values is degrees. 130
5.19 Error histograms for sub-model TG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees. . . 130
5.20 Absolute error histograms for sub-model TG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees. 131
5.21 Heatmaps analysis for sub-model TG of "Model_Struct" on dataset "LM_Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are degree. 132
5.22 Error histograms for sub-model TG of "Model_Struct" on dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees. . . 132
5.23 Absolute error histograms for sub-model TG of "Model_Struct" on dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees. 133
5.24 The histogram of root-mean-square deviation of Model_1, Model_2 and Model_3 tested on dataset "Struct_Test". 134
5.25 Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_15016`. The atomic labels (white digits) are shown only for the reference structure due to the close alignment between both configurations. 135

5.26 Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_16580`. The atomic labels are shown for the reference structure (white digits) and the ML-predicted structure (black digits). 136
5.27 Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_16399`. The atomic labels are shown for the reference structure (white digits) and the ML-predicted structure (black digits). 137
5.28 Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_16261`. The atomic labels are shown for the reference structure (white digits) and the ML-predicted structure (black digits). 138
5.29 Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_15931`. The atomic labels are shown for the reference structure (white digits) only due to the clustered atoms in the ML-predicted structure. 139

List of Tables

3.1 The origination of datasets 50
3.2 Datasets used for each model in this work 50

4.1 The value of metrics on testing for three Mulliken charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error. 74
4.2 The value of metrics on larger molecule testing for three Mulliken charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error. 77
4.3 The value of metrics from testing the three Hirshfeld charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error. 80
4.4 The value of metrics on larger molecule testing for three Hirshfeld charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error. 83
4.5 The value of metrics from testing for three CM5 charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error. 86
4.6 The value of metrics for larger molecule testing for three CM5 charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error. 89
4.7 The value of metrics for tests for three DDEC6 charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error. 91
4.8 Accuracy comparison with existing DDEC6-based charge models. The present work is evaluated in vacuum ($\varepsilon = 1$), whereas Bleiziffer's model and DeepAtomicCharge were tested in specific solvent environments. 92
4.9 The value of metrics from testing three semi-empirical charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error. 96

4.10 The training time consumption for each charge and model. Mulliken charge, Hirshfeld charge, CM5 charge and DDEC6 charge models are trained with batch size of 16. SE charge models are trained with batch size of 768. All trainings run for 100 epochs. Due to the current implementation of program "prj_acdm", the cost of time can only be measured with a precision of seconds. 97

4.11 The time consumption of Mulliken charge model "Model_1" tested on different size of normal alkane. The predicting batch size is 1. Due to the current implementation of program "prj_acdm", the cost of time can only be measured with a precision of seconds. 98

4.12 The time consumption of the Mulliken charge model "Model_1" predicting charge on dataset "SE_Train". The predicting batch size is 768. Due to the current implementation of program "prj_acdm", the cost of time can only be measured with a precision of seconds. 99

5.1 Interaction definition of level-III angle graph in different cases. θ (in blue colour) and γ (in red colour) denote two corresponding bond angles in the level-I molecule graph. The solid circles in the graph denote the atoms belonging to bond angle θ, the hollow circles denote the atoms belonging to bond angle γ and the diagonally hatched circles denote the shared atoms. . 107

5.2 The value of metrics for test on dataset "Struct_Test" for the sub-model BG of three "Model_Struct" models. The acronym TL80AE is short for the top low 80% absolute error. 123

5.3 The value of metrics for test on dataset "LM_Struct_Test" for the sub-model BG of three "Model_Struct" models. The TL80AE is short for the top low 80% absolute error. 125

5.4 The value of metrics for tests on dataset "Struct_Test" for the sub-model AG for three "Model_Struct" models. The acronym TL80AE is short for the top low 80% absolute error. 127

5.5 The value of metrics for test on dataset "LM_Struct_Test" for the sub-model AG of three "Model_Struct" models. The TL80AE is short for the top low 80% absolute error. 129

5.6 The value of metrics for test on dataset "Struct_Test" for the sub-model TG of three "Model_Struct" models. The acronym TL80AE is short for the top low 80% absolute error. 131

5.7 The value of metrics for tests on dataset "LM_Struct_Test" for the sub-model TG of three "Model_Struct" models. The acronym TL80AE is short for the top low 80% absolute error. 133

5.8 Statistical analysis of RMSD values (Å) for molecular structure predictions across three models evaluated on the "Struct_Test" dataset 134

5.9 Numbers of extracted values for bond lengths, bond angles and torsion angles for training and testing datasets in Project MSP 140

5.10 The time consumption for training GLaCE- and MPF-based molecule structure prediction models. Due to the current implementation of the program "prj_acdm", the cost of time can only be measured with a precision of seconds. 140
5.11 The time cost when predicting structures from datasets "Struct_Test" and "LM_Struct_Test". Due to the current implementation of the program "prj_acdm" the time cost can only be measured with a precision of seconds. 141

Chapter 1

Introduction

Drug development is often a lengthy and complex process, consisting of four major stages — target identification, compound screening, preclinical development and clinical development [1]. Some reported that the minimum required time for the whole drug development timeline is around 5 years, where some other researches state that the time consumption can be up to 15 years [2]. Since both the preclinical and clinical phases of drug development require experimental validation, i.e. they cannot be replaced by purely computational simulations in order to ensure the patients' safety, improving the efficiency of earlier stages — such as target identification and compound screening — is important for reducing the overall cost of time of the drug development [2]. With the recent development on machine learning (ML), the applications of ML models in drug development has significantly increased for shortening the cost of drug development, especially in the stages of target identification and compounds screening [3]. In this thesis, particular emphasis is placed on the use of ML methods in virtual screening (VS), aiming to improve the efficiency of compound selection prior to experimental validation.

There are two primary classes of virtual screening methods — Ligand-based virtual screening (LBVS) and structure-based virtual screening (SBVS). LBVS focuses on screening large databases of bioactive ligands or fragments to identify compounds with similar activity, while SBVS relies on molecular docking to screen ligands, predicting their binding modes in protein active sites and evaluating ligand-receptor interactions, even for polypeptides [4]. A recent report shows that about 54% of latest publications on virtual screening use molecular docking technique, which can computationally predict how small molecule binding to biological targets using electrostatic and steric interactions, scored by functions like RMSD and binding affinity [4].

The conventional docking methods for screening through the database is relatively costly. For example, it is reported that it took nearly 24 hours for the OpenEye GigaDocking to dock the Enamine REAL database into the the purine nucleoside phosphorylase with 27,000 CPUs [5]. Another example is that the VirtualFlow spent 2 weeks to screen through a 1.4 billion entry database with 8,000 CPUs [5]. Such requirement of high-performance

clusters can be unaffordable for many research groups. Therefore, many ML models have been published to accelerate the virtual screening process. Some remarkable progress is briefly introduced below as examples. B. Ramsundar et al. proposed a multitask deep-learning model in 2017. This work shows that a multitask deep-learning model can achieve satisfactory accuracy in several drug developing tasks [6]. In 2022, F. Gentile et al. developed a model named "deepdocking" based on a deep neural network (i.e. multi-layers fully-connected neural network) for accelerating molecular docking process. Later tests indicates that the "deepdocking" model can screen a database up to 100 times faster than conventional methods [5]. And most recently, X. Zhang et al. introduced a message-passing neural network based model "KarmaDock" for ligand docking in 2023. It is reported that, the "KarmaDock" successfully identified the experimentally validated case [7].

However, there are still several concerns on these existing models. Firstly, models like B. Ramsundar's model are adapted from existing NN algorithm with limited modification. Since these algorithms are not designed for molecular properties prediction, the efficiency and robustness of the models can be a major concern. In addition, the time cost for screening through a large database is still high. According to Reference [5], the fastest reported model is only 100 (maximum) times faster than the conventional approaches. Furthermore, the NN models used in the above reports lack flexibility, since the trained models can only accept certain size of molecules due to the structure of the algorithms [5, 6, 8]. Last but not least, the "KarmaDock" model is reported to be trained on eight NVIDIA A100-SXM4-80GB GPUs and 64-core CPUs [7], which is relatively costly.

During my PhD study, two original algorithms — the message-passing featuriser (MPF) and graph latent connectivity enhancement (GLaCE) algorithm — were proposed to overcome the limitations mentioned above. These two algorithms are designed to

- predict molecular properties effectively and accurately,
- require small amounts of data to train,
- be affordable for personal computers,
- accept arbitrary size of molecules after being trained, and
- accept SIMLES as input format without any requirement of pre-calculations.

To validate these two algorithms, I focus on two important molecular properties in molecular docking — atomic charge and molecular structure, and build two example projects — project ACP and project MSP respectively. The literature review about atomic charges and conventional approaches in obtaining atomic charges and molecular structure is presented in Chapter 2. The selection and generation of the training and testing datasets are provided in Chapter 3. Two algorithms are presented as part of the methodology within project ACP and project MSP in Chapters 4 and 5, respectively. In Chapter 6, I introduce a program "prj_acdm" which is designed for this study, implementing the MPF and GLaCE algorithms.

Chapter 2

Background

In this chapter, I present the background information about my PhD studies. As this research is a cross-discipline project, the background context consists of three parts. The first part is an overall literature review on the definition of different atomic charges. The second part introduces the conventional approaches to obtaining atomic charges and molecular structure. The third part is a general introduction to machine learning and neural networks.

2.1 Background of Atomic Charges

Atomic charge, or net charge, is a fundamental concept in computational chemistry. It can play an important role in chemistry and medicine research when optimising molecular structure and seeking binding sites for drugs [9, 10]. However, the concept of atomic charge is only computational and it is unable to be experimentally observed and measured [11]. This situation leads to various definitions of atomic charge and the associated methods to compute it. The preferred method can depend on the application, and none of these definitions and methods can be proved as "correct" in any absolute sense[12, 13].

In this section I review several definitions of atomic charges, including orbital based charge, potential derived charge and density related charge. It should be noted that some modern methods cannot be strictly classified to any single category, as they may use a combination of above methods and even be independent from these three classes.

2.1.1 Mulliken Charge

In quantum mechanics, wave functions are used to describe a system. However, the exact wave function can only be calculated by solving the Schrödinger equation, which is possible only for small multi-electron systems. In earlier work F. Hund [14] and R. S. Mulliken [15] started to use the concept of orbitals to approximate the wave function in multi-electron system. In 1955 R. S. Mulliken proposed an atomic charge definition on the basis of linear combination atomic orbitals molecular orbitals (LCAO-MO) theory, well known as the

Mulliken charge [16]. Below I give the mathematical definition of the Mulliken charge according to R. S. Mulliken's original paper [16].

The introduction of Mulliken charge starts from the LCAO-MO theory. An n-atom molecule consists of a set of atoms $A = \{A_1, A_2, \ldots, A_n\}$. If the molecule has N electrons, there will be $N/2$ occupied molecular orbital ψ_k. The molecular orbital ψ_k can now be expended as a linear combination of the atomic orbitals to give

$$\psi_k = \sum_{i=0}^{n} c_{ki}\phi_i. \tag{2.1}$$

Here, ϕ_i is an atomic orbital attached to atom i that contributes to the molecular orbital and c_{ki} is the coefficient for atomic orbital ϕ_i. The electron density $n(\vec{r})$ is given by the sum of the squares of the occupied molecular orbitals

$$n(\vec{r}) = 2\sum_{k=1}^{N/2} \psi_k^2(\vec{r}). \tag{2.2}$$

The total number of electrons is obtained by integrating the density over all space. By substituting Equation (2.1) into Equation (2.2), the population now reads

$$n(\vec{r}) = 2\sum_{k=1}^{N/2} \left\{ \sum_{i=0}^{n} c_{ki}^2\phi_i^2(\vec{r}) + \sum_{\substack{i,j=0,\\ i\neq j}}^{n} c_{ki}c_{kj}\phi_i(\vec{r})\phi_j(\vec{r}) \right\}. \tag{2.3}$$

If we now integrate over all space, Equation (2.3) becomes

$$N = 2\sum_{k=1}^{N/2}\sum_{i=0}^{n} c_{ki}^2 + 2\sum_{k=1}^{N/2}\sum_{\substack{i,j=0,\\ i\neq j}}^{n} c_{ki}c_{kj}S_{ij} \tag{2.4a}$$

where

$$S_{ij} = \int \phi_i\phi_j \mathrm{d}V \tag{2.4b}$$

is the overlap integral. The terms c_{ki}^2 and $c_{ki}c_{kj}S_{ij}$ are partial populations (originally named sub-populations, by R. S. Mulliken), wherein, c_{ki}^2 is the net population, while $c_{ki}c_{kj}S_{ij}$ is the overlap population.

To assign charges to atoms, R. S. Mulliken introduced the concept of gross atomic populations (originally named as such), which later became known as the Mulliken population. For the example given here, that is

$$N_i^{\text{Mulliken}} = 2\sum_{k=1}^{N/2} \left\{ c_{ki}^2 + \sum_{\substack{j=0\\ j\neq i}}^{n} c_{ki}c_{kj}S_{ij} \right\} = 2\sum_{k=1}^{N/2}\sum_{j=0}^{n} c_{ki}c_{kj}S_{ij}, \tag{2.5}$$

where N_i is the net population for atom i. According to Equations (2.4a) and (2.5), R. S.

Mulliken allocates the overlap populations equally to every associated net populations c_{ki}^2. This is a major feature of Mulliken population. From Equation (2.5), the Mulliken charge can be calculated as

$$Q_i^{\text{Mulliken}} = Z_i - N_i^{\text{Mulliken}} \tag{2.6}$$

where Z_i is the nuclear charge number of atom i.

As the above definitions indicate, the Mulliken charge is easy to calculate. However, there are several concerns about Mulliken charges. Mathematically, the Mulliken charge is very dependent on the form of the basis functions. Thus, only if the basis functions accurately describe the atomic site can the Mulliken charge be correctly calculated [12]. A report from C. W. Kern and M. Karplus shows that the Mulliken charge can vary greatly with the choice of basis functions, even under the same conditions of size and energy [17]. From a physical perspective, the equal split of overlapping population (as Equation (2.5) indicates) can be valid only if the two atoms are the same. Another physical concern is the population output from Equation (2.5) can be negative, which is physically impossible [18].

However, obtaining the Mulliken charge is usually simple, as it is a byproduct of many quantum chemistry calculations [16]. Since Mulliken charge is easy to calculate, it is still widely accepted and adopted despite the significant drawbacks described above [18].

2.1.2 Löwdin Charge

To overcome the limitations of Mulliken charges, significant effort has been applied to try to improve on Mulliken populations. The Löwdin charge is one famous improvement. Although P.-O. Löwdin did not directly propose the definition of Löwdin charge, in 1950 he proposed the underlying orthogonalisation method used to compute the Löwdin charge [12, 19]. This orthogonalisation method is applied to the atomic orbitals as follows

$$\varphi_u = \sum_{i=0}^{n} S_{ui}^{-1/2} \phi_i \tag{2.7}$$

where φ_u is the orthogonalised atomic orbital corresponding to the basis function indexed by u, S is the overlap matrix of basis function u and i (already defined in Equation (2.4b)) and ϕ_i is the original atomic orbital indexed by i.

If we now expand the molecular orbitals in terms of the Löwdin orbitals we get

$$\psi_k = \sum_{u=0}^{n} c'_{ku} \varphi_u, \tag{2.8}$$

where c'_{ku} is the coefficient for orthogonalised atomic orbitals φ_u. We can now obtain populations using the same procedure proposed by Mulliken but now the basis functions are orthonormal, so that

$$S'_{uv} = \int \varphi_u \varphi_v \mathrm{d}V = \delta_{uv}. \tag{2.9}$$

Substituting Equation (2.9) into Equation (2.5) we obtain

$$N_u^{\text{Löwdin}} = 2\sum_{k=1}^{N/2}(c'_{ku})^2. \tag{2.10}$$

Since the Löwdin orthogonalisation mixes atomic orbitals across the molecule, $N_u^{\text{Löwdin}}$ may not strictly correspond to the physical atom. Thus the Löwdin charge $Q_u^{\text{Löwdin}}$ represents the effective charge distribution in the orthogonalised basis set u, with the corresponding net charges being

$$Q_u^{\text{Löwdin}} = Z_u - N_u^{\text{Löwdin}}. \tag{2.11}$$

Here, the Z_u is still taken to be the nuclear charge number of atom u, despite the delocalisation problem caused by orthogonalisation.

From Equation (2.7) it can be found that the Löwdin charge is less sensitive to the choice of basis functions than the Mulliken charge as all atomic orbitals have been orthogonalised [12]. In addition, the Löwdin population analysis does not produce any negative populations as it only contains squared terms (see Equation (2.10)), which is physically correct [18]. However, there are still some concerns with Löwdin population analysis. One is that, as Löwdin orthogonalisation mixes up the atomic orbitals. the electron distribution becomes delocalised[18]. Another concern is that orthogonalisation conceals the overlap partition problem within the transformed basis, i.e. it does not physically solve the problem [12].

Here I have provided proofs only of Mulliken and Löwdin charges due to space limitation. There are many orbital-based methods, such as natural population analysis (NPA) [20] and the natural bond orbital method (NBO) [21], which also play significant roles in charge distribution analysis.

There are alternative ways of deriving atomic charges. Some of these methods and definitions are presented in the following sections.

2.1.3 Merz-Kollman (MK) Charge

Another approach to generate atomic charges is to use the potential surrounding a molecule to rebuild the atomic charge values, i.e. potential derived charges. Many of these methods are based on the electrostatic potential (ESP, or MEP, short for molecular electrostatic potential) [12, 22, 23].

The ESP at position $\vec{r}$, $V_{\text{ESP}}(\vec{r})$, found from a first principles calculation of a molecule is

$$V_{\text{ESP}}(\vec{r}) = \sum_i \frac{Z_i}{|\vec{r} - \vec{R}_i|} - \int \frac{\rho(\vec{r}')}{|\vec{r} - \vec{r}'|}\mathrm{d}\vec{r}'. \tag{2.12}$$

where Z_i is the nuclear charge number of atom i, $\vec{R}_i$ is the position of atom i and ρ is the electron density of the molecule. The electron density ρ can be obtained via Hartree-Fock

or Density Functional Theory calculations. In Equation (2.12), the first term, $\sum_i \frac{Z_i}{|\vec{R}_i - \vec{r}|}$, is the positive electrostatic potential generated by nuclei, while the second term, $\int \frac{\rho(\vec{r}')}{|\vec{r}' - \vec{r}|} d\vec{r}'$, is the negative electrostatic potential contributed by electrons. The MK charge [24] is designed to reproduce this potential, at least approximately. The central assumption is the potential can be written as

$$V'(\vec{r}) = \sum_i \frac{Q_i^{\mathrm{MK}}}{|\vec{r} - \vec{R}_i|}, \tag{2.13}$$

where Q_i^{MK} is the assumed point charge on atom i, i.e. the atomic charge. To obtain the value of Q_i^{MK}, a fitting process is introduced and the loss function is defined as

$$\mathcal{L}_{\mathrm{MK}} = \chi^2_{\mathrm{ESP}} = \sum_u \left(V_{\mathrm{ESP}}(\vec{r}_u) - V'(\vec{r}_u) \right)^2 \rightarrow \mathrm{MIN}. \tag{2.14}$$

By minimising this loss function, plus a total charge constrain on total charge number of the molecule

$$Q_{\mathrm{mol}} = \sum_i Q_i^{\mathrm{MK}}, \tag{2.15}$$

the atomic charge Q_i^{MK} can be solved.

In Equation (2.14), the sampling positions $\vec{r}_i$ used to define the MK charges are distributed evenly in several shells surrounding the molecule, with the distance of

$$r_i = R_i^{\mathrm{vdW}} \times k, \tag{2.16}$$

where R_i^{vdW} is the van der Waals radii of atom i, and k is the factor (typically 1.4, 1.6, 1.8 and 2.0), with a sampling density of 1 point/Å[12].

Some other potential derived charge methods such as Charges from Electrostatic Potentials (CHELP) [25] and Charges from Electrostatic Potentials using a Grid based method (CHELPG) [26] are similar to the method described above. The key differences are the positions of the sampling points. The CHELP method uses a similar criterion to Equation (2.16) to determine the radius of the shells that the sampling points are located in, but CHELP discards points located within the van der Waals radii of any atom to avoid large distortions which may affect the later calculation[25]. CHELPG distributes its sampling points based on a cubic grid surrounding the molecule, discarding any sampling points that lie inside the van der Waals radius of any atom[26]. This cubic grid can ensure a full coverage of the ESP region [23]. The difference in method for selecting the position of the sampling points gives these potential derived charge methods different features. It is reported that the CHELPG method has the best rotational invariance performance, while the fitting accuracy of the MK charge is slightly better than CHELP and CHELPG [23, 27].

2.1.4 Restrained Electrostatic Potential (RESP) charge

The Restrained Electrostatic Potential (RESP) charge was first introduced by C. I. Bayly et al [13] in 1993, and it is a typical potential based charge. The aim of RESP is to improve the ESP-based charges by introducing an extra restraint term to ESP. This restraint term is given by:

$$\chi^2_{\text{rstr}} = a \sum_j \left((Q_j^2 + b^2)^{\frac{1}{2}} - b \right), \tag{2.17}$$

where a determines the asymptotic limit of the restraint strength, b controls how tightly the hyperbola is centred around its minimum and Q_j is the fitted charge for atom j. So the loss function used to determine the RESP charges is

$$\mathcal{L}_{\text{RESP}} = \chi^2_{\text{ESP}} + \chi^2_{\text{rstr}} \rightarrow \text{MIN}. \tag{2.18}$$

Here, the first term χ^2_{ESP} is defined in Equation (2.14), while the second term χ^2_{rstr} is defined in Equation (2.17).

The locations of the sampling points for RESP charges follow similar rules to those of CHELPG [25]. That is, the sampling points of RESP are located in shells as defined by Equation (2.16) and are located outside the van der Waals radii of any atom [13]. More information has been provided above.

The main advantage of RESP is that it can reduce the magnitude of the fitted atomic charges by introducing a penalty for extreme values. This allows RESP to provide results for charges with better stability than previous ESP-based methods, in both quantitative and qualitative analyses [13]. However, a major criticism on RESP is that the restraint term is a mathematical solution without a proper physical meaning (i.e. black-box fix), though the restraint term can prevent extreme charge values in the results [12].

2.1.5 Atoms in Molecules (Bader) Charge

Apart from the orbital-based and ESP-based methods, density related charge is another way to obtain atomic charges. In 1972, R. F. W. Bader proposed the atoms in molecules (AIM) methods [28] to generate atomic charge, which was later named Bader charge. Bader charge relies on the electron density ρ, and spacially splits the charge and assigns the charge to the corresponding atom. The split rule of the electron density follows the definition of a “zero-flux surface”, i.e.

$$\nabla\rho(\vec{r}') \cdot \mathbf{n}(\vec{r}') = 0. \tag{2.19}$$

Here, the $\vec{r}'$ denotes an arbitrary point in the "zero-flux surface" (see Figure 2.1), and the n is the unit vector perpendicular to the surface at that point. In Equation (2.19), the term $\nabla\rho(\vec{r}')$ means the gradient of the electron density, i.e. the direction of the change of the electron density. It is notable that, these spaces split by the "zero-flux surface" do not overlap each other. Thus, the Bader charge of atom i can be calculated by integrating the

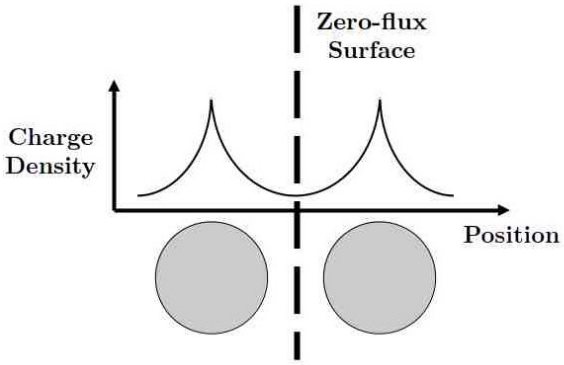


Figure 2.1: Illustration of the "zero-flux surface" in Bader charge definition.

electron density over the corresponding split space Ω_i, that is

$$Q_i^{\text{Bader}} = Z_i - \int_{\Omega_i} \rho(r) \mathrm{d}V, \tag{2.20}$$

where Z_i is the nuclear charge number of atom i and Ω_i is the volume centred on site i that lies inside a zero flux surface.

However, the way that the AIM method splits space can lead to several problems. The first problem is that, in some cases (e.g. the space in between of two lithium atoms), a split space Ω_u may be generated by the rule, but with no nuclei inside that space. This kind of space Ω_u is called a pseudo atom. Another problem is that, some spaces (noted as Ω_v) may contain more than one nucleus in several systems, such as the KrH^+ system [23].

2.1.6 Hirshfeld Charge

Hirshfeld charge was introduced by F. L. Hirshfeld in 1977 [29]. Hirshfeld charge is also a density related charge, calculated by integrating electron density over volume. The major difference between Bader charge and Hirshfeld charge is the rules that define how the space is split. In order to split the space with a proper physical meaning, F. L. Hirshfled first constructed a promolecule density $\rho^{\text{pro}}(\vec{r})$ at point $\vec{r}$ as

$$\rho^{\text{pro}}(\vec{r}) = \sum_i \rho_i^{\text{at}}(\vec{r}), \tag{2.21}$$

where ρ_i^{at} is the electron density for the ground-state atom i. This promolecule electron density ρ^{pro} is a direct sum of the ground-state electron density for all atoms in the molecule. For each atom, a sharing function, i.e. a weight factor, is then generated as

$$w_i(r) = \frac{\rho_i^{\text{at}}(r)}{\rho^{\text{pro}}(r)}. \tag{2.22}$$

This sharing function determines how the molecular electron density $\rho(\vec{r})$ at point $\vec{r}$ should be assigned to each atom. From Equations (2.21) and (2.22), it can be found that the sum of all the weighting functions $w_i(\vec{r})$ at arbitrary position $\vec{r}$ is always 1. Then the electron

density of any bonded atom in the molecule can be defined as

$$\rho_i^{\text{b.a.}}(\vec{r}) = w_i(\vec{r})\rho_i^{\text{mol}}(\vec{r}). \tag{2.23}$$

Here, $\rho_i^{\text{mol}}(\vec{r})$ denotes the electron density of the overall molecule. Thus, the Hirshfeld charge can be calculated by

$$Q_i^{\text{Hirshfeld}} = Z_i - \int \rho_i^{\text{b.a.}}(\vec{r})\mathrm{d}v, \tag{2.24}$$

where Z_i, again, is the nuclear charge number of atom i [11].

While the definition of Hirshfeld charge has a clear physical meaning, critics says that, from the proofs above, the Hirshfled's partitioning of space is fuzzy due to the overlaps between the divided atomic spaces [23].

2.1.7 Charge Model 5 (CM5)

CM5 [30] is an atomic charge model derived from the Hirshfeld charge, while its predecessor CMx models [31–37] are based on Mulliken, Löwdin or redistributed Löwdin populations. The aim of CM5 is to create a charge model that relies neither on the electronic structure nor the basis set [30]. The CM5 charge is defined by the sum of a Hirshfeld charge and a correction term as

$$Q_i^{\text{CM5}} = Q_i^{\text{Hirshfeld}} + \sum_{k\neq l} T_{kl}B_{kl}, \tag{2.25}$$

where, k and l denote two arbitrary atoms in the molecule, T_{kl} represent the coefficients and B_{kl} is the "Pauling bond order" (named by A. V. Marenich [30]). In general cases, the coefficient matrix T is defined as

$$T_{kl} = \begin{cases} 0, & Z_k = Z_l, \\ D_{Z_kZ_l}, & Z_k \neq Z_l \text{ and } Z_l, Z_k \in \{1,5,6,8\}, \\ D_{Z_k} - D_{Z_l}, & \text{otherwise.} \end{cases} \tag{2.26}$$

Here, the Z_k denotes the atomic number of atom k, while D_{Z_k} and $D_{Z_kZ_l}$, respectively, are the atomic parameter for atom k, and the pair parameter for atom k and l. The value of parameters D_{Z_k} and $D_{Z_kZ_l}$ can be found in Table 1 in A. V. Marenich's original paper [30]. In addition, A. V. Marenich defines several constraints to calculate the parameter value for particular elements, such as Li, C, Cl, etc. Due to the space limitation, readers are referred to Section 2.2 in A. V. Marenich's original statement [30] for further information.

In Equation (2.25), the symbol B_{kl} is given as

$$B_{kl} = \exp\left[-\alpha(r_{kl} - R_{Z_k} - R_{Z_l})\right], \tag{2.27}$$

where $\alpha = 2.474$ (Å^{-1}) is an optimised empirical parameter, r_{kl} is the distance between atoms k and atom l and R_{Z_k} is the atomic covalent radius of atom k.

The CM5 charge has a broad range of potential applications across many fields. A. V. Marenich states the CM5 charge can benefit molecular mechanics simulation, computation of solvation free energies and generating realistic electrostatic potential and multiple moment [30].

2.1.8 Density Derived Electrostatic Charge 6 (DDEC6)

Introduced by T. A. Manz in 2016, the DDEC6 charge is also a density based method to calculate atomic charges[38–41]. The key objective of DDEC6 is to be insensitive to the basis set. In addition, T. A. Manz states that, the previous methods Hirshfeld population analysis and DDEC3 model may not be able to converge to a unique result. There are nine major features of DDEC6 [38], which are:

1. Each atom is assigned with a precise electron distribution.
2. Core electrons are correctly assigned to nuclei.
3. DDEC6 is independent of the basis set.
4. Polyatomic multipole expansions can effectively converge with DDEC6.
5. DDEC6 follows the Pauling electronegativity trend.
6. For some particular elements, DDEC6 has good transferability in equivalent bonds in different conformations.
7. DDEC6 charges are consistent with atomic spin moments.
8. The DDEC6 method efficiently converges to a unique charge value.
9. The cost of DDEC6 increases linearly with number of atoms as the size of the system increases.

Since DDEC6 has several advantages over other methods, much research has been carried out on the basis of DDEC6 charge models, especially in machine learning charge prediction [10, 42]. However, the complete proof of the DDEC6 charge evaluation is relatively complex ($\sim$ 16 pages in the original reference), and it is impossible to reproduce all the processes in the thesis. For detailed information it is suggested the reader check the original reference [38]. Here, I only summarise the key ideas of DDEC6 charges in non-periodic system as follows.

The definition of DDEC6 is similar to many density based charges, that is

$$Q_i^{\mathrm{DDEC6}} = Z_i - \int \rho_i^{\mathrm{D.6}}(\vec{r}_i)\mathrm{d}^3 r_i, \tag{2.28a}$$

and

$$\vec{r}_i = \vec{r} - \vec{R}_i, \quad \text{for non-periodic system} \tag{2.28b}$$

where Z_i is the number of nuclei charge of atom i, $\rho_i^{\mathrm{D.6}}(\vec{r}_i)$ represents the electron distribution assigned to atom i at position $\vec{r}_i$, and $\vec{R}_i$ is the position of atom i. In DDEC6, the distribution of electron density is based on the vectorised charge partitioning method, which reads

$$\rho_i^{\mathrm{D.6}}(\vec{r}_i) = \frac{w_i^{\mathrm{D.6}}(r_i)}{W^{\mathrm{D.6}}(\vec{r})}\rho^{\mathrm{mol}}(\vec{r}). \tag{2.29}$$

Here, ρ^{mol} is the overall electron density of the molecule, and $w_i^{\mathrm{D.6}}$ and $W^{\mathrm{D.6}}$ are the spherically symmetric atomic weighting factor for atom i and the overall weight respectively. The weights fulfill the relation $W^{\mathrm{D.6}}(\vec{r}) = \sum_i w_i^{\mathrm{D.6}}(r_i)$. From Equation (2.29), the vectorised charge partitioning here can simplify the three dimensional electron distribution problem by introducing a one dimensional weight factor $w_i^{\mathrm{D.6}}(r_i)$. After seven iteration steps (for further information, see reference [38]), the final form of the weight factor can be obtained by

$$w_i^{\mathrm{D.6}}(r_i) = e^{K_i} H_i(r_i) \tag{2.30}$$

where K_i is a Lagrange multiplier that ensure the core electrons are correctly bonded[1] and $H_i(r_i)$ is a normalised multi-constraint distribution function that ensures the buried tails do not excessively contract[2].

2.2 Modern Computation on Materials

In the last century, the rapid development of computational science has greatly changed the study of materials. With modern machines and algorithms, researchers can predict behaviours, optimise performance of materials, and even beyond. This section reviews the background information related to the objectives of this thesis in computational materials science. From various tools and algorithms, I here choose two methodologies that are used in later projects, molecular dynamics and density functional theory, and provide short introductions to them.

2.2.1 Molecular Dynamics (MD)

Molecular dynamics is an approach to simulate a molecular system under certain conditions (e.g. temperature, pressure, etc.) so the behaviours of the molecule can be observed and its properties can be obtained [43]. MD simulation was first introduced by B. J. Alder in 1957 [44] and was further developed in later years [45]. MD simulations start from atomic forces, which are usually derived from empirical force fields or DFT results. A typical force field can be a sum of various potential contributions:

$$U_{\mathrm{total}} = U_{\mathrm{stretch}} + U_{\mathrm{bend}} + U_{\mathrm{torsion}} + U_{\mathrm{electrostatic}} + U_{\mathrm{van\ der\ Waals}} + U_{\mathrm{hydrogen\ bond}}. \tag{2.31}$$

Here, U_{stretch}, U_{bend} and U_{torsion} are the energy of deformation of the bond length, bond angle and torsion with respect to their equilibrium values. $U_{\mathrm{electrostatic}}$ is the term repre-

[1] see eq.33,93 in reference [38]
[2] see eq.92 and section 3.4(ii) in reference [38]

senting Coulomb's law. $U_{\text{van der Waals}}$ describes the dispersion interaction between atoms. $U_{\text{hydrogen bond}}$ is the energy of the hydrogen bond, though may not be included in some methods. It is notable that, the total potential can be obtained via empirical methods or DFT results, which allows us to classify MD simulations into two main categories: empirical molecular dynamics (EMD) and ab initio molecular dynamics (AIMD).

According to Equation (2.31), the force can now be obtained via

$$\vec{F}_i = -\vec{\nabla}_i U_{\text{total}}, \tag{2.32}$$

where $\vec{r}_i$ is the position of atom i. In MD simulations atoms move according to Newton's laws of motion:

$$\vec{F}_i = \frac{\mathrm{d}\vec{p}_i}{\mathrm{d}t} = m_i \frac{\mathrm{d}^2 r_i}{\mathrm{d}t^2}, \tag{2.33}$$

where $\vec{p}_i$ denotes the momentum of atom i, t is time, m_i represents mass of atom i and $\vec{r}_i$ is the position of atom i. The key idea of molecular dynamics simulation is to integrate Equation (2.33) numerically. We can generate rules for advancing the positions and momenta of the atoms by assuming small time increments and then applying Taylor's expansion. All the quantities in Equation (2.33) thus evolve in time according to:

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 + \ldots, \tag{2.34a}$$

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \vec{a}(t)\Delta t + \frac{1}{2}\vec{b}(t)\Delta t^2 + \ldots \tag{2.34b}$$

and

$$\vec{a}(t+\Delta t) = \vec{a}(t) + \vec{b}(t)\Delta t + \ldots. \tag{2.34c}$$

Here, $\vec{a}$ is the acceleration and $\vec{b}$ is the third derivative of the position with respect to time. Several algorithms are available to obtain the trajectories of the molecular dynamics. Here, the Verlet algorithm [46] is provided as an example.

The original Verlet algorithm takes the second order of the Taylor's expansion (Equation (2.34a)) as

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 \tag{2.35a}$$

and

$$\vec{r}(t-\Delta t) = \vec{r}(t) - \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2. \tag{2.35b}$$

By adding the two equation above we obtain:

$$\vec{r}(t+\Delta t) = 2\vec{r}(t) - \vec{r}(t-\Delta t) + \vec{a}(t)\Delta t^2. \tag{2.36}$$

The MD algorithm can now start and calculate new positions from the current state. However, this original Verlet algorithm faces a difficulty in obtaining the velocity at the new position, though can obtain the velocity in previous steps [43]. To fix this problem, an improved version of the Verlet algorithm was proposed, namely the velocity Verlet algorithm [47].

The velocity Verlet algorithm follows the same assumptions as the Leapfrog integration method (not introduced here), i.e. the velocity varies slowly with time over the time step Δt. According to this assumption, a concept of half-step velocity is introduced, which can be written as

$$\vec{v}\left(t+\frac{\Delta t}{2}\right)=\vec{v}(t)+\frac{1}{2}\vec{a}(t)\Delta t. \tag{2.37}$$

So Equation (2.35a) can similarly be rewritten as

$$\begin{aligned}\vec{r}\,(t+\Delta t)&=\vec{r}(t)+\vec{v}(t)\Delta t+\frac{1}{2}\vec{a}(t)\Delta t^2\\&=\vec{r}(t)+\vec{v}\left(t+\frac{\Delta t}{2}\right)\Delta t.\end{aligned} \tag{2.38}$$

By doing another half-step operation (as Equation (2.37)), the current velocity can be obtained by

$$\vec{v}(t+\Delta t)=\vec{v}\left(t+\frac{\Delta t}{2}\right)+\frac{1}{2}\vec{a}(t+\Delta t)\Delta t. \tag{2.39}$$

Now, the MD algorithm is fully defined [48].

MD simulation have a wide range of applications especially in biology and medicine. Nowadays, researchers use MD to study proteins critical to neuronal signaling, develop drug targeting, discover the mechanisms of protein aggregation associated with neurodegenerative disorders, etc. [49].

2.2.2 Density Functional Theory (DFT)

DFT is a widely accepted method for calculating the electronic structure of a system. In later sections of this thesis the DFT method is used to generate training and testing sets with consistent parameters. Here, I provide a brief introduction to the DFT method. The following proofs are based on reference [50].

Schrödinger Equation The wave function Ψ of a system can be obtained by solving the time-independent Schrödinger equation

$$\hat{H}\Psi=E\Psi, \tag{2.40}$$

where Ψ is the normalised wave function, E is the total energy and $\hat{H}$ is the Hamiltonian operator. The Hamiltonian of a one-electron-one-nucleus system in vacuum has the following form:

$$\begin{aligned}\hat{H}&=-\frac{\hbar^2}{2m}\nabla^2-\frac{e^2}{4\pi\varepsilon_0 r} && \text{in SI units,}\\&=-\frac{1}{2}\nabla^2-\frac{1}{r} && \text{in atomic units.}\end{aligned} \tag{2.41}$$

In Equation (2.41) $\hbar$ is the reduced Planck's constant and m is the mass of the electron. The first term, $-\frac{\hbar^2}{2m}\nabla^2$, is the kinetic operator for the electron and the second term, $V(r)=-\frac{e^2}{4\pi\varepsilon_0 r}$, is the potential operator, where e is the elementary charge, ε_0 is the vacuum permittivity, r is the distance between the electron and the nucleus, and E is the

total energy of the system. To make the notation simpler we will use atomic units from now on.

For a many-particle system, several interaction terms are appended to the Hamiltonian, giving

$$\hat{H} = \hat{T}_{\mathrm{n}} + \hat{T}_{\mathrm{e}} + \hat{V}_{\mathrm{nn}} + \hat{V}_{\mathrm{ee}} + \hat{V}_{\mathrm{ne}}, \tag{2.42}$$

where

$$\begin{aligned}
\hat{T}_{\mathrm{n}} &= -\sum_{I} \frac{1}{2M_I} \nabla^2_{R_I} && \text{is the kinetic energy of the nuclei,} \\
\hat{T}_{\mathrm{e}} &= -\frac{1}{2} \sum_{i} \nabla^2_{r_i} && \text{is the kinetic energy of the electrons,} \\
\hat{V}_{\mathrm{nn}} &= \frac{1}{2} \sum_{I,J} \frac{Z_I Z_J}{|\vec{R}_I - \vec{R}_J|} && \text{is the nucleus-nucleus interaction energy,} \\
\hat{V}_{\mathrm{ee}} &= \frac{1}{2} \sum_{i,j} \frac{1}{|\vec{r}_i - \vec{r}_j|} && \text{is the electron-electron interaction energy, and} \\
\hat{V}_{\mathrm{ne}} &= -\sum_{i,I} \frac{Z_I}{|\vec{R}_I - \vec{r}_i|} && \text{is the electron-nucleus interaction energy.}
\end{aligned}$$

Here, M_I, $\vec{R}_I$ and Z_I denote the mass, position and atomic number of nucleus I, while m_i and $\vec{r}_i$ are the mass and position of electron i. However, solving the Schrödinger equation analytically for many body systems is very challenging. Some major difficulties are:

1. the high-dimensionality (many electronic and nuclear degrees of freedom),
2. complex correlations produced by Coulomb interactions,
3. the antisymmetry of the wave function under exchange of electrons (they are Fermions), and
4. the costly computation of the electron-electron repulsive potentials.

Thus, the precise analytic solution of the Schrödinger equation is only possible for simple systems, such as a single hydrogen atom.

Born-Oppenheimer Approximation In order to obtain the wave function of a many-body system several approximations have been proposed to simplify the many-body problem. One of the most widely applied methods is the Born-Oppenheimer approximation (abbreviated as BO). Since nuclei are relatively more massive and slow-moving than electrons, the BO approximation assumes the electrons interact with the nuclei in the system as though they were "frozen" at the nuclear positions $\vec{R}$. Consequently, the total wave function can be divided into an electronic part and a nuclear part, and the total wave function Ψ can be rewritten as

$$\Psi = \Psi_{\mathrm{e}}(\tau, \vec{R}) \Psi_{\mathrm{n}}(\vec{R}), \tag{2.43}$$

Here $\Psi_{\mathrm{n}}(\vec{R})$ is the normalised wave function for the nuclei and depends on the position of the nuclei ($\vec{R}$), and $\Psi_{\mathrm{e}}(\tau, \vec{R})$ is the normalised wave function for the electrons, depending on the degree of freedom of electrons τ (including position $\vec{r}$ and spin s, i.e. $\tau = (\vec{r}, s)$) and position of the nuclei ($\vec{R}$). The Hamiltonian for the combined system can now be separated into two parts: , a Hamiltonian for the electrons ($\hat{H}_{\mathrm{e}}$) and an effective Hamiltonian for the nuclei ($\hat{H}_{\mathrm{n}}^{\mathrm{eff}}$). These two parts have the following forms:

$$\hat{H}_{\mathrm{e}} = \hat{T}_{\mathrm{e}} + \hat{V}_{\mathrm{ee}} + \hat{V}_{\mathrm{ne}} \tag{2.44a}$$

$$\hat{H}_{\mathrm{n}}^{\mathrm{eff}} = \hat{T}_n + E_{\mathrm{e}}(\vec{R}) + V_{\mathrm{nn}}(\vec{R}), \tag{2.44b}$$

and they satisfy

$$\hat{H}_{\mathrm{e}}\Psi_{\mathrm{e}}(\tau, \vec{R}) = E_{\mathrm{e}}(\vec{R})\Psi_{\mathrm{e}}(\tau, \vec{R}), \tag{2.44c}$$

$$\hat{H}_{\mathrm{n}}^{\mathrm{eff}}\Psi_{\mathrm{n}}(\vec{R}) = E_{\mathrm{total}}\Psi_{\mathrm{n}}(\vec{R}). \tag{2.44d}$$

Here, $E_{\mathrm{e}}(\vec{R})$ denotes the energy of electrons and E_{total} denotes the total energy of the system. Readers should note that Equations (2.44a) and (2.44b) is only correct for molecules. For crystals, the nuclei-nuclei potential term V_{nn} should be attributed to the electron Hamiltonian ($\hat{H}_{\mathrm{e}}$) to avoid having an infinite energy per unit cell for $\hat{H}_{\mathrm{e}}$. The BO approximation successfully separates the motion of electrons and nuclei. However, further approximation is still required to solve the many-body problem analytically.

Variational Method A common practice in solving Schrödinger equation for electrons (Equation (2.44c)) is using the variational method. According to the variational principle, the value of electron energy

$$E_{\mathrm{e}}[\tau] = \langle\Psi_{\mathrm{e}}|\, \hat{H}_{\mathrm{e}}\, |\Psi_{\mathrm{e}}\rangle = \int \Psi_{\mathrm{e}}^{*}\hat{H}_{\mathrm{e}}\Psi_{\mathrm{e}}\mathrm{d}\tau \tag{2.45}$$

calculated from any normalised wave function Ψ_{e} is always more positive than the minimum eigenvalue of the operator $\hat{H}_{\mathrm{e}}$, i.e. the ground-state energy $E_{\mathrm{e},0}$. For an N electrons system, the ground-state energy $E_{\mathrm{e},0}$ can be obtained by minimising the energy functional by

$$\delta\left[\langle\Psi_{\mathrm{e}}|\, \hat{H}_{\mathrm{e}}\, |\Psi_{\mathrm{e}}\rangle - E_{\mathrm{e}}(\langle\Psi_{\mathrm{e}}|\Psi_{\mathrm{e}}\rangle - 1)\right] = 0, \tag{2.46}$$

where E_{e} is the Lagrange undetermined multiplier (also the energy of the system) that constrains the normalisation of Ψ_{e}. Thus the ground-state energy is

$$E_{\mathrm{e},0} = \langle\Psi_{\mathrm{e},0}|\, \hat{H}_{\mathrm{e}}\, |\Psi_{\mathrm{e},0}\rangle\,, \tag{2.47}$$

where $\Psi_{\mathrm{e},0}$ denotes the ground-state wave function for the system.

Slater Determinant To use the above variational method in practice, we need to define a form for the wave function. The simplest form for Ψ_{e} for a system that fulfils antisymmetry with N electrons is a Slater Determinant constructed from N distinct single-electron spin

orbitals $\psi_i(\tau_j)$:

$$\begin{aligned}\Psi_{\mathrm{e}} &= \Psi_{\mathrm{e}}(\tau_1, \tau_2, \ldots, \tau_N) \\ &= \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1(\tau_1) & \psi_1(\tau_2) & \cdots & \psi_1(\tau_N) \\ \psi_2(\tau_1) & \psi_2(\tau_2) & \cdots & \psi_2(\tau_N) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_N(\tau_1) & \psi_N(\tau_2) & \cdots & \psi_N(\tau_N) \end{vmatrix} \\ &= \frac{1}{\sqrt{N!}} \det\left(\psi_i(\tau_j)\right).\end{aligned} \tag{2.48}$$

Here, the element in the determinant in row i and column j denotes the electron spin orbital i for electron j. This spin orbital consists of the product of a spatial orbital $\phi_i(\vec{r}_j)$ and a spin function $\chi_i(s_j)$:

$$\psi_i(\tau_j) = \phi_i(\vec{r}_j)\chi_i(s_j). \tag{2.49}$$

Here, the spin orbitals ψ_i are normalised to 1 and are orthogonal to each other.

The Slater determinant constructs a wave function for all the electrons in the system Ψ_{e}, and ensures all single-electron spin orbital ψ_i satisfy the Pauli exclusion principle.

Hartree-Fock Method By substituting Equation (2.48) into Equation (2.44c) (the Schrödinger equation with a many-body Hamiltonian for the electrons), the energy of an M-atom-N-electron system can be obtained as

$$E^{\mathrm{HF}} = \sum_{i=1}^{N} h_i + \frac{1}{2}\sum_{\mathrm{i,j}=1}^{N}(J_{ij} - K_{ij}), \tag{2.50a}$$

where

$$h_i = \int \psi_i^*(\tau_i)\left(-\frac{1}{2}\nabla_i^2 - \sum_{I=1}^{M}\frac{Z_I}{|R_I - r_i|}\right)\psi_i(\tau_i)\mathrm{d}\tau_i, \tag{2.50b}$$

$$J_{ij} = \iint \psi_i^*(\tau_1)\psi_i(\tau_1)\left(\frac{1}{|r_1 - r_2|}\right)\psi_j(\tau_2)\psi_j^*(\tau_2)\mathrm{d}\tau_1\mathrm{d}\tau_2 \tag{2.50c}$$

and

$$K_{ij} = \iint \psi_i^*(\tau_1)\psi_j(\tau_1)\left(\frac{1}{|r_1 - r_2|}\right)\psi_i(\tau_2)\psi_j^*(\tau_2)\mathrm{d}\tau_1\mathrm{d}\tau_2. \tag{2.50d}$$

Here, the h_i contains the kinetic energy of the electron and the nuclei-electron interaction energy. J_{ij} and K_{ij} are the Coulomb and exchange interactions respectively, with both representing the interactions between electrons. As noted in connection with Equation (2.49), the wave function ψ_i is orthogonal to other spin orbitals. Thus,

$$\langle\psi_i|\psi_j\rangle = \delta_{ij}, \tag{2.51}$$

where δ_{ij} is the Kronecker function. To minimise the energy E^{HF} in Equation (2.50a), a Lagrange multiplier ε_{ij} (a Hermitian matrix that satisfies $\varepsilon_{ij} = \varepsilon_{ij}^*$) is introduced and the

variational method is applied (as Equation (2.46)). That is

$$\delta\left(E^{\mathrm{HF}}-\sum_{i,j=1}^{N}\varepsilon_{ij}(\langle\psi_i|\psi_j\rangle-\delta_{ij})\right)=0. \tag{2.52}$$

By varying Equation (2.52) and combining all the contributing terms (h_i, J_{ij} and K_{ij}), the Fock operator ($\hat{F}$) can be obtained as

$$\hat{F}=-\frac{1}{2}\nabla^2-\sum_{I=1}^{M}\frac{Z_I}{|R_I-r|}+\sum_{j=1}^{N}(\hat{J}_j-\hat{K}_j), \tag{2.53a}$$

where

$$\hat{J}_j(\tau_1)\psi_i(\tau_1)=\int\psi_j^*(\tau_2)\psi_j(\tau_2)\left(\frac{1}{r_{12}}\right)\mathrm{d}\tau_2\psi_i(\tau_1) \qquad \text{and} \tag{2.53b}$$

$$\hat{K}_j(\tau_1)\psi_i(\tau_1)=\int\psi_j^*(\tau_2)\psi_i(\tau_2)\left(\frac{1}{r_{12}}\right)\mathrm{d}\tau_2\psi_j(\tau_1). \tag{2.53c}$$

Here, the term $-\frac{1}{2}\nabla^2$ is the kinetic energy operator, the second term $\sum_{I=1}^{M}\frac{Z_I}{|R_I-r|}$ is the nuclei-electron potential operator, and the third term $\sum_{j=1}^{N}(\hat{J}_j-\hat{K}_j)$ denotes the repulsive potential operator between electron. The eigenequation (Fock equation) is then

$$\hat{F}\psi_i=\varepsilon_i\psi_i, \tag{2.54}$$

where ε_i is the energy of a spin orbital (single-electron wave function) ψ_i. The Fock operator depends on the orbitals through $\hat{J}_j$ and $\hat{K}_j$. Thus to solve Equation (2.54), a self-consistent field (SCF) method is used. The general process of SCF is:

1. make a guess for an initial set of orbitals $\psi_i^{(s)}=\psi_i^{(0)}$,
2. construct the Fock operator $\hat{F}$ using the initial orbitals $\psi_i^{(s)}$,
3. solve the Fock equations given in Equation (2.54) to produce a new set of orbitals $\psi_i^{(s+1)}$, and
4. repeat Step 2, with $\psi_i^{(s)}=\psi_i^{(s+1)}$, followed by Step 3 until convergence is reached.

By doing the above, the spin-orbital for electron i (i.e. ψ_i) can be found.

Hartree-Fock (HF) is a wave function based method. The construction of a multi-electron wave function relies on the Slater determinant. The HF method uses a variational method to optimise the wave function of the electron and to calculate their ground state energy. The energy calculated by the HF method consists of the electron kinetic energy, the electron-nuclei attractive potential, the electron-electron repulsive potential, and the exchange energy between electrons. However, it ignores the correlation energy between electrons.

Hohenberg-Kohn Theorems Different from the HF method, the density functional

theory (DFT) is an electron-density based method and DFT does not contain explicit many body wave functions. Before continuing to the DFT method, several fundamental theorems are introduced first.

For an N-electron system, the Hamiltonian can read

$$\hat{H}^{\mathrm{HK}} = \hat{F}^{\mathrm{HK}} + \hat{V}_{\mathrm{ext}}, \tag{2.55}$$

where the universal operator $\hat{F}^{\mathrm{HK}}$ (for functional, $F^{\mathrm{HK}}[\rho]$) contains only the kinetic energy of electrons $\hat{T}_{\mathrm{e}}$ and the electron-electron potential $\hat{V}_{\mathrm{ee}}$. $\hat{V}_{\mathrm{ext}}$ is the external potential operator, and makes a contribution to the total energy of $\int \rho(\vec{r})V_{\mathrm{ext}}(\vec{r})\mathrm{d}\vec{r})$, where ρ is the electron density. The external potential includes the nuclei-electron potential $\hat{V}_{\mathrm{ne}}$ and other potentials contributed by external fields. The first Hohenberg-Kohn (HK) theorem states that, for a non-degenerate ground state, the external potential $\hat{V}_{\mathrm{ext}}(\vec{r})$ is determined (up to a real constant $C \in \mathbb{R}$) by a unique ground-state electron density distribution $\rho_0(\vec{r})$, i.e.

$$\rho_0(\vec{r}) \leftrightarrow V_{\mathrm{ext}}(\vec{r}) + C \quad (C \in \mathbb{R}). \tag{2.56}$$

The first HK theorem establishes a one-to-one correspondence in a non-degenerate ground state between the external potential V_{ext} and the ground-state electron density distribution $\rho_0(\vec{r})$. Since $\hat{F}^{\mathrm{HK}}$ is a universal operator (independent of the system), the hamiltonian $\hat{H}$ is fully determined by the external potential $\hat{V}_{\mathrm{ext}}$, which can be determined by the ground-state electron density $\rho_0(\vec{r})$. This correspondence simplifies the framework of many-electron systems and greatly affects the later simulation methods in quantum chemistry.

Another important fundamental theorem is the second HK theorem, which introduces the variational method into the energy functional E^{HK}. Compared to Equation (2.46), the variational process in the second HK theorem is based on the electron density $\rho(\vec{r})$ instead of the wave function Ψ_{e}. That is

$$\delta\left[E^{\mathrm{HK}}[\rho] - \mu\left(\int \rho(\vec{r})\mathrm{d}\vec{r} - N\right)\right] = 0, \tag{2.57}$$

where N is the number of electrons in the system and μ is the Lagrange multiplier (chemical potential). The energy reads

$$E^{\mathrm{HK}}[\rho] = F^{\mathrm{HK}}[\rho] + \int \rho(\vec{r})V_{\mathrm{ext}}(\vec{r})\mathrm{d}\vec{r}. \tag{2.58}$$

By solving Equation (2.57), the ground-state electron density distribution $\rho_0(\vec{r})$ can be obtained. The second HK theorem introduced above gives the full definition of a density-based variational method. Along with the first HK theorem, these two theorems provide an efficient framework in solving many-body problem.

Kohn-Sham Equation The Kohn-Sham (KS) equation is also an important basis for practical DFT calculations. Inspired by the HF method (introduced above), the KS method creates a non-interacting reference system, based on single-electron orbitals. The Hamil-

tonian (no relativity) in the KS method reads

$$\hat{H}_{\text{ref}}^{\text{KS}} = -\sum_{i}^{N} \frac{1}{2}\nabla_i^2 + \sum_{i}^{N} V_{\text{eff}}(\vec{r}_i), \tag{2.59}$$

where V_{eff} is the effective external potential (introduced below). Inspired by the HF method, the KS method also uses orbitals from a Slater determinant as Equation (2.48). As with Equation (2.49), each orbital in the Slater determinant reads

$$\psi_i(\tau_j) = \phi_i(\vec{r}_j)\chi_i(s_j) \tag{2.60}$$

where $\phi_i(\vec{r}_j)$ is the spatial orbital and $\chi_i(s_j)$ is the spin function. In the KS method, the spin orbital (or KS orbital) is determined by the Kohn-Sham equation

$$\hat{h}^{\text{KS}}\psi_i = \varepsilon_i\psi_i \tag{2.61a}$$

where the KS operator

$$\hat{h}^{\text{KS}} = -\frac{1}{2}\nabla^2 + V_{\text{eff}}(\vec{r}). \tag{2.61b}$$

Different from the Fock operator (Equation (2.53a)), the KS operator does not have a direct term for electron-electron interactions. Instead, an effective potential $V_{\text{eff}}(\vec{r})$ is introduced, which consists of three parts, an external potential $V_{\text{ext}}(\vec{r})$ (see Equation (2.55)), a Hartree potential $V_{\text{Hartree}}(\vec{r})$ and an exchange-correlation (XC) potential $V_{\text{xc}}(\vec{r})$, which will be introduced later. That is

$$V_{\text{eff}}(\vec{r}) = V_{\text{ext}}(\vec{r}) + V_{\text{Hartree}}(\vec{r}) + V_{\text{xc}}(\vec{r}), \tag{2.62a}$$

where

$$V_{\text{Hartree}}(\vec{r}) = \int \frac{\rho(\vec{r}')}{|\vec{r} - \vec{r}'|}\mathrm{d}\vec{r}' \tag{2.62b}$$

and

$$V_{\text{xc}}(\vec{r}) = \frac{\delta E_{\text{xc}}[\rho]}{\delta\rho(\vec{r})}. \tag{2.62c}$$

where $E_{xc}[\rho]$ is the exchange and correlation energy (see below).

Thus, the total energy of a M-atom-N-electron system in the KS method can be written as a sum of density functionals: the electron kinetic energy in a non-interacting system ($T_{\text{s}}[\rho]$, the kinetic term in Equation (2.58)), the energy contributed by the external potential ($E_{\text{ext}}[\rho]$), the Hartree energy (the Coulomb repulsive potential, $E_{\text{Hartree}}[\rho]$), the nuclei-

nuclei potential energy (E_{nn}) and the exchange and correlation energy (E_{xc})

$$\begin{aligned} E^{\mathrm{KS}} &= T_{\mathrm{s}}[\rho] + E_{\mathrm{ext}}[\rho] + E_{\mathrm{Hartree}}[\rho] + E_{\mathrm{nn}} + E_{\mathrm{xc}}[\rho] \\ &= \sum_{i}^{N} \langle \psi_i | -\frac{1}{2}\nabla^2 | \psi_i \rangle + \int V_{\mathrm{ext}}(\vec{r})\rho(\vec{r})\mathrm{d}\vec{r} \\ &\quad + \frac{1}{2}\iint \frac{\rho(\vec{r})\rho(\vec{r}')}{|\vec{r}-\vec{r}'|}\mathrm{d}\vec{r}\mathrm{d}\vec{r}' \\ &\quad + \frac{1}{2}\sum_{I\neq J}^{M} \frac{Z_I Z_J}{|\vec{R}_I - \vec{R}_J|} + E_{\mathrm{xc}}[\rho]. \end{aligned} \tag{2.63}$$

The electron density and the spin orbitals are related by

$$\rho(\vec{r}) = \sum_{i}^{N} |\psi_i(\vec{r})|^2. \tag{2.64}$$

In the KS method, the SCF method is applied to solve the equations. The self-consistency process in the KS method is slightly different from that for the HF one. The general process of SCF is:

1. guess an initial electron density $\rho^{(s)}(\vec{r}) = \rho^{(0)}(\vec{r})$,
2. calculate the effective potential according to Equation (2.62a) with $\rho^{(s)}(\vec{r})$,
3. solve Equation (2.61a) and get a new set of orbitals $\psi_i^{(s+1)}$ and eigenvalues $\varepsilon_i^{(s+1)}$,
4. calculate a new electron density $\rho^{(s+1)}(\vec{r})$ according to Equation (2.64), and
5. return to Step 2 with $\rho^{(s)}(\vec{r}) = \rho^{(s+1)}(\vec{r})$ until convergence is reached. Note that in practice we have to keep part of the old density when updating the effective potential.

By doing above, the KS equation can be solved, and density functional theory calculations can be performed.

The exchange and correlation energy The exchange and correlation (XC) energy functional term $E_{\mathrm{xc}}[\rho]$ in Equation (2.63) is a key parameter of the DFT method. This (semi-)empirical XC energy functional consists of four parts: the exchange energy (explicit), the correlation energy (explicit), the correction to the non-interacting kinetic energy term (the difference from the real kinetic energy, implicit) and the correction to the Hartree energy term (removing the electron self-interaction, implicit). By way of example, some of the functionals are listed and introduced here.

The local density approximation (LDA) [51] is the simplest XC energy form originated from the uniform electron gas model. In the LDA the XC term is given as

$$E_{\mathrm{xc}}^{\mathrm{LDA}}[\rho] = \int \rho(\vec{r})\varepsilon_{\mathrm{xc}}(\rho(\vec{r}))\mathrm{d}\vec{r}, \tag{2.65}$$

where $\varepsilon_{\mathrm{xc}}(\rho)$ is the exchange-correlation energy for a single electron in a uniform electron

gas with desity of ρ. This $\varepsilon_{\mathrm{xc}}(\rho(\vec{r}))$ can be also divided into two parts: the exchange contribution and correlation contribution,

$$\varepsilon_{\mathrm{xc}}(\rho(\vec{r})) = \varepsilon_{\mathrm{x}}(\rho(\vec{r})) + \varepsilon_{c}(\rho(\vec{r})). \tag{2.66}$$

Thus, Equation (2.65) can be rewritten into

$$E_{\mathrm{xc}}^{\mathrm{LDA}}[\rho] = E_{\mathrm{x}}^{\mathrm{LDA}}[\rho] + E_{\mathrm{c}}^{\mathrm{LDA}}[\rho]. \tag{2.67}$$

For open-shell system, two electron densities are used to build the XC energy, one for each spin. The simplest exchange and correlation functional for spin polarized systems is given by the local spin density approximation (LSDA) [52]. The LSDA can be written as

$$\begin{aligned} E_{\mathrm{xc}}^{\mathrm{LSDA}}[\rho_\alpha, \rho_\beta] &= \int \rho(\vec{r}) \varepsilon_{\mathrm{xc}}(\rho_\alpha(\vec{r}), \rho_\beta(\vec{r})) \mathrm{d}\vec{r} \\ &= E_{\mathrm{x}}^{\mathrm{LSDA}}[\rho_\alpha, \rho_\beta] + E_{\mathrm{c}}^{\mathrm{LSDA}}[\rho_\alpha, \rho_\beta]. \end{aligned} \tag{2.68}$$

Here, ρ_α and ρ_β represent the electron densities in two spin status ($\uparrow$ and $\downarrow$) respectively, and satisfy $\rho_\alpha(\vec{r}) + \rho_\beta(\vec{r}) = \rho(\vec{r})$.

The generalised gradient approximation (GGA) [53] is another type of XC energy functional. The GGA introduces the gradient $\vec{\nabla}\rho(\vec{r})$ to incorporate uneven electron density distributions. The XC energy functional form for a GGA is

$$E_{\mathrm{xc}}^{\mathrm{GGA}}[\rho, \vec{\nabla}\rho] = E_{\mathrm{x}}^{\mathrm{GGA}}[\rho, \vec{\nabla}\rho] + E_{\mathrm{c}}^{\mathrm{GGA}}[\rho, \vec{\nabla}\rho]. \tag{2.69}$$

However, the above methods have systematic errors due to the approximations used. By introducing the HF energy term to the XC term, producing a hybrid functional, the precision of an XC functional can be improved. The form of a hybrid functional is

$$E_{\mathrm{xc}}^{\mathrm{hybrid}} = \alpha E_{\mathrm{x}}^{\mathrm{HF}} + (1-\alpha) E_{\mathrm{x}}^{\mathrm{DFT}} + E_{\mathrm{c}}^{\mathrm{DFT}}, \tag{2.70}$$

where α is the fraction of the HF exchange energy (in B3LYP [54] $\alpha = 0.2$), $E_{\mathrm{x}}^{\mathrm{DFT}}$ is the DFT exchange energy and $E_{\mathrm{c}}^{\mathrm{DFT}}$ is the DFT correlation energy (in B3LYP these two term are hybrids of LDA and GGA methods).

2.3 Background of Artificial Neural Networks

Machine learning (ML) algorithms are now well-developed, and ML is bringing many fresh ideas to the simulation of materials [55]. In the following sections, the basic concepts and a general introduction to artificial neural network (ANN, or NN) are presented, giving the readers background information before continuing to the new algorithms used in this project.

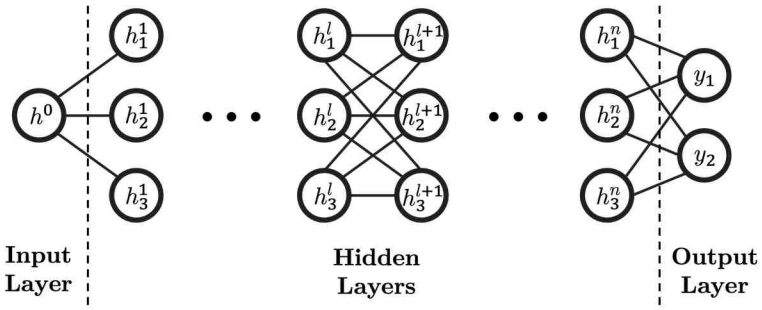


Figure 2.2: Basic structure of artificial neural network

2.3.1 Basic Concepts

A NN is an algorithm that is inspired by the biological brain structure. A typical NN structure consists of several neurons in a layer with weights (connections) that connect neurons between layers.[3] Figure 2.2 shows the basic structure of a typical NN. A detailed definition and an introduction to each element in the neural network will be presented below, and is based on the documentation for Keras [56] and Tensorflow [57], and Reference [55, 58].

Neurons Each circle in Figure 2.2 represents a neuron, which is one of the basic computational component of an NN. Each neuron accepts one number or an array of numbers as input from the neurons in the previous layer, and processes these number according to a specific algorithm. For a model with n hidden layers and m neurons in each layer, the processed number is called the neuron feature. The processing of numbers (assuming each neuron produces only one number) in the neural network is usually defined as

$$\mathbf{H}^{l+1} = \sigma(\mathbf{W}^l \mathbf{H}^l + \mathbf{B}^l) \qquad \text{in matrix notation} \tag{2.71a}$$

$$h_i^{l+1} = \sigma\left(\sum_j^n w_{i,j}^l h_j^l + b_i^l\right) \qquad \text{for each element,} \tag{2.71b}$$

where $\mathbf{H}^l$ is the matrix of neuron features of neurons in layer l, and h_i^l denotes the feature of the i-th neuron in layer l. The quantity $\mathbf{W}^l$ is the matrix of the weights on the connections (introduced below) connecting layers l and $l+1$, and $w_{i,j}^l$ (shape: $n \times m$) represents the specific weight that connected neuron i in layer $l+1$ and neuron j in layer l. The symbol σ represents the activation function and $\mathbf{B}^l$ is the bias vector of layer l with its element b_i^l for each neuron in layer l. It should be noted that the bias vector can be thought of as a 1D-array. Every element in the bias vector is a scaler offset and the vector itself does not include any spatial direction information. The activation function and the bias are explained in later paragraphs.

Weights (Connection) The black lines in Figure 2.2 that connect neurons between layers are called weights. Each weight in the neural network contains a trainable weighting

[3] Neurons and weights are sometime referred to as "nodes" and "edges". To prevent confusion, in this dissertation, the terms "node" and "edge" are used exclusively to refer to concepts from graph theory.

parameter (the training process is introduced later), i.e. the $\mathbf{W}^l$ and $w^l_{i,j}$ in Equation (2.71). The value of the weight in an connection can be changed by an optimiser during back propagation in the training process. The concepts of optimiser, back propagation and training are explained later.

Bias The biases ($\mathbf{B}^l$ and b^l_i in Equation (2.71)) are another set of trainable parameters in neural networks. The bias adds an offset to the value in each neuron, enabling the optimiser (introduced later) to fit the training data (introduced later) with more flexibility.

Input Layer The input layer accepts the external input variables (h^0 in Figure 2.2, or commonly $\mathbf{X}$) and passes them to the later hidden layers for processing. The input layer usually does not process the input variable or just simply applies an activation function (introduced later). The size of input layer, i.e. the number of neurons in the input layer, determines the maximum number of variables that the neural network can accept. A neural network usually has one input layer.

Hidden Layer The hidden layers are the main body of a neural network model. They accept values from previous layers and process these values, passing them to an output value (as Equation (2.71) shows). The hidden layers usually have most of the neurons of the neural network. The number of layers and the number of neurons in each layer can significantly affect the number of trainable parameters (introduced below), and ultimately affect the flexibility of the entire neural network.

Output Layer The output layer is the layer that generates the final output value (y_i in Figure 2.2). The number of neurons in the output layer determines the number of outputs of the entire model. The neurons in the output layer accept the values from the last hidden layer, and process them into the final output of the model.

Activation Function The activation function can introduce a non-linear transformation into the neuron processing (the σ in Equation (2.71)). Some commonly used activation functions are presented in Figure 2.3. The mathematical forms of these activation functions are: $y_{\text{sigmoid}} = \frac{1}{1+\exp(-x)}$, $y_{\tanh} = \tanh(x)$, $y_{\text{relu}} = \max(0, x)$ and $y_{\text{linear}} = x$ (i.e. disable the activation function). These non-linear transformations can extend the flexibility of the model, since the core of a neural network is a set of linear transformations (see Equation (2.71)). In addition, some activation functions can normalise the output (such as sigmoid and tanh) and some can accelerate the convergence of model training (such as ReLU).

Forward Pass The forward pass process in neural networks refers to the predicting process within the model. That is, the input layer accepts input variables (x) and passes them to a hidden layer. In each hidden layer the neurons in that layer process the numbers according to Equation (2.71) and pass the results to the next layer. The output layer gets the numbers from the previous layer and processes them into the final output of the model, i.e. the predicted numbers from the model.

Back Propagation Back propagation happens only when training the model. During the

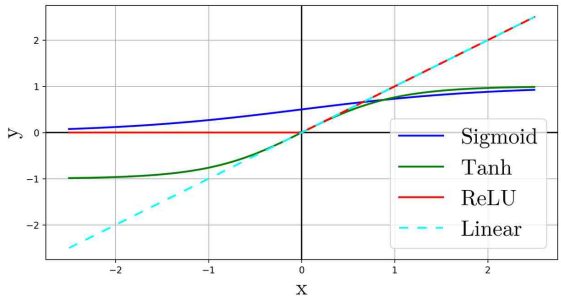


Figure 2.3: Some commonly used activation functions.

back propagation process the optimiser (explained later) will adjust the value of a trainable parameter (explained later) according to the loss function (explained later). This process starts from the end of the model (output layer), and goes in reverse to the input layer while changing the weight and bias at each layer.

Loss Function The loss function ($\mathcal{L}$) is the criterion used to evaluate the quality of a predicted result of the model when training. Usually, the value of a loss function is non-negative and a lower value means a better prediction of the accuracy (compared with the true value in the training set). There are several commonly used loss functions that can evaluate quality of the result in different aspects, based on the differences between the true value y and predicted value $\hat{y}$. See Equations (2.72a) to (2.72c) for more information on how the loss functions are defined. Categorising loss functions is not part of this work.

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2 \qquad \text{Mean Squared Error} \tag{2.72a}$$

$$\mathcal{L}_{\text{MAE}} = \frac{1}{n}\sum_i |y_i - \hat{y}_i| \qquad \text{Mean Absolute Error} \tag{2.72b}$$

$$\mathcal{L}_{\text{RMSE}} = \sqrt{\mathcal{L}_{\text{MSE}}} \qquad \text{Root Mean Squared Error} \tag{2.72c}$$

Optimiser The optimiser determines the strategy for optimising the trainable parameters (explained later) to improve the prediction quality (lowering the loss function value). One simple case of an optimiser is the stochastic gradient descent (SGD) method [59], which reads

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}(\theta_t). \tag{2.73}$$

Here, θ is the current weight (or the trainable parameter, explained below), and θ' is the optimised parameter. The η in Equation (2.73) is the learning rate. Although SGD is simple and can be easily implemented, it suffers from several drawbacks like bad stability and a significant dependency on the value of the learning rate, a hyper-parameter. Another famous optimiser is adaptive moment estimation (Adam) [60], which introduces momentum and root mean squared propagation (RMSProp) [61] to traditional gradient descent. The mathematical form of the Adam optimiser is

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.74a}$$

where

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \nabla_\theta \mathcal{L}(\theta_t) \tag{2.74b}$$

and

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}, \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) \cdot (\nabla_\theta \mathcal{L}(\theta_t))^2. \tag{2.74c}$$

Note that β_1^t and β_2^t denote β_1 and β_2 to the power t respectively, where t denotes the current stage (iteration), η is the learning rate, β_1 is the momentum decay rate, β_2 is the RMSProp decay rate and ϵ is a small number that prevents division by zero. The Adam optimiser is a stable method for most cases, and it does not require manual adjustment of the learning rate. Thus, the Adam optimiser is now widely accepted in many machine learning models and research [7, 62].

Trainable Parameter The total number of trainable parameters $N_\theta = \dim(\theta)$ determines the flexibility of the NN model. The set of all trainable parameters θ consist of all weights and biases in the model. For example, for a NN model containing n hidden layer, the trainable parameter set can be expressed as

$$\theta = \left\{ \mathbf{W}^0, \mathbf{W}^1, \ldots, \mathbf{W}^n, \mathbf{B}^0, \mathbf{B}^1, \ldots, \mathbf{B}^n \right\}. \tag{2.75}$$

Training The training process is the process to determine the value of all trainable parameter with a training set. For supervised learning, the result in the training set is known. In a single iteration, the training process performs a forward pass, loss calculation and back propagation with optimiser adjusting the trainable parameter. There are two main hyperparameters for training, epoch and batch size. The training process going once through the whole training set is called one epoch. The batch size is how many samples in the training set are examined in each iteration. The relation among iteration, epoch and batch size is

$$\text{Iteration per Epoch} = \frac{\text{Number of Samples in Training Set}}{\text{Batch Size}}. \tag{2.76}$$

Usually, a smaller batch size may require large number of epochs when training to avoid under fitting, as explained later.

Validation The validation process is an examination of over fitting (explained below) during the training process. The validation process relies on an independent validation set which is different from the training set, and this validation set does not directly affect the adjustment of trainable parameters. In each iteration of the training process, the losses of both the training and validation sets are calculated. Ideally, with each training iteration the training and validation losses should decrease together. However, if over fitting has occurred, the training loss can continue to decrease while the validation loss may increase. This phenomena is a key sign of over fitting. By monitoring the trend of validation loss, it can be determined whether the training process should be stopped early or the hyper-

parameters of the optimiser adjusted. By doing so, over fitting can be properly avoided.

Testing The testing set is an independent set of sample data mainly used for evaluating the performance of the model (i.e. the accuracy of the predicted results). By testing models with the same testing set, the differences in performance of these model can be directly compared.

Under Fitting Under fitting denotes that the model is not fully trained. An under fitted model may be unstable and could make predictions with large errors. It can occur when the flexibility of the model is not enough for the problem, or the number of training epochs is not enough for the optimiser to adjust the parameters.

Over Fitting If a model is over fit it means the model fits the noise in the training set and loses the ability to generalise the problem. This can happen if the number of training epochs is too large, or the model is so flexible that it can fit every sample in the training set. To avoid over fitting, one solution is using the validation set to determine if an early stop is needed (see the validation paragraph). Other solutions include enlarging the training set and reducing the number of trainable parameters.

2.3.2 Fully connected Neural Network

The fully connected neural network is the fundamental structure of neural networks. It refers to a network in which every neuron in one layer is fully connected to all other neurons in the subsequent layer. The example (Figure 2.2) given in Section 2.3.1 is based on an n-layer fully connected neural network with 3 neurons in each hidden layer. Equation (2.71) is exactly the mathematical form of a fully connected neural network.

The fully connected neural network introduces a high degree of flexibility to the entire model by presenting a large number of trainable parameters. Thus, the fully connected structure is of importance in neural network models but also introduces risks of over fitting due to the enormous number of trainable parameters available. This problem requires users to be careful when designing the size of the fully connected structure [63].

2.3.3 High-dimensional Neural Network (HDNN)

High-dimensional neural network is a joint name for neural networks that process more than one number in each neuron. The neuron feature h_i^l in a HDNN can be a complex number [64], a vector (e.g. capsule network [65]) or a tensor [66]. Due to the extra information that flows within the model, HDNNs can produce a better expression on non-linear conversion. The tensor-based HDNN can significantly reduced the number of trainable parameters while the predicted results remain accurate [66]. It is also reported that HDNNs can converge 2-3 times faster than conventional neural networks [67].

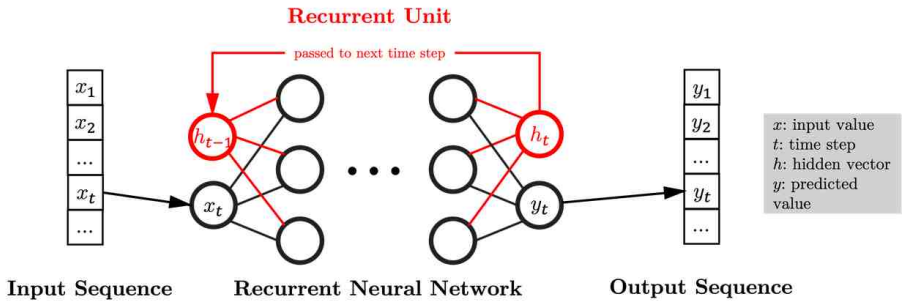


Figure 2.4: An example of a simple recurrent neural network

2.3.4 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) [68] is a NNs with recurrent connections. Figure 2.4 gives an example of a simple RNN structure. The RNN in this example accepts input data sequentially, and predicts values as a fully connected neural network (see Section 2.3.2). The distinct feature of RNNs is the recurrent unit, presented in red in Figure 2.4. This recurrent unit can retain the hidden vector from the previous input, and use this feature as a part of the input for the RNN. In the case given in Figure 2.4, for an step t input x_t from the input sequence, the network accepts another input value h_{t-1} from previous time step $t-1$, i.e. the "memory" from previous time step. Apart from the output value y_t, the RNN then generates a hidden vector h_t, which will be used in the next time step as the memory.

RNNs are widely used in many fields [69], such as text processing [70], speech emotion recognition [71], image scene labelling [72], etc. However, the classic RNN suffer from an unstable gradient, either vanishing or diverging [69]. Thus, some improvements, such as a gated recurrent unit (GRU) [73], are made to solve this problem. Since this work is not about RNNs, further information is not provided here. Readers are referred to Reference [69–73] for further reading.

2.3.5 Convolutional Neural Network (CNN)

A convolutional neural network was first proposed by Y. LeCun as "LeNet-5" [74] in 1998. It is designed for document recognition. Figure 2.5 presents the general process of a 2D CNN. The first part of a CNN model is the convolutional operation. For an input $\mathbf{X}$ with shape $L \times L$, the model first tries to extract features by applying an $M \times M$ filter (i.e. the convolutional kernel) $\mathbf{K}$ on the input 2D array. Here, the size of this convolutional kernel $\mathbf{K}$ should not be greater than the original array $\mathbf{X}$, i.e. $M \leq L$. The convolutional operation "masks" the filter $\mathbf{K}$ on every possible fully-covered position of $\mathbf{X}$ (see the orange and blue example frames in Figure 2.5), sums up the element-wise product of the filter $\mathbf{K}$ and the sub-matrix of $\mathbf{X}$ on each step, and generates an $N \times N$ feature map $\mathbf{F}$ by assigning value to specific positions (see red and blue arrows in Figure 2.5). The mathematical expression

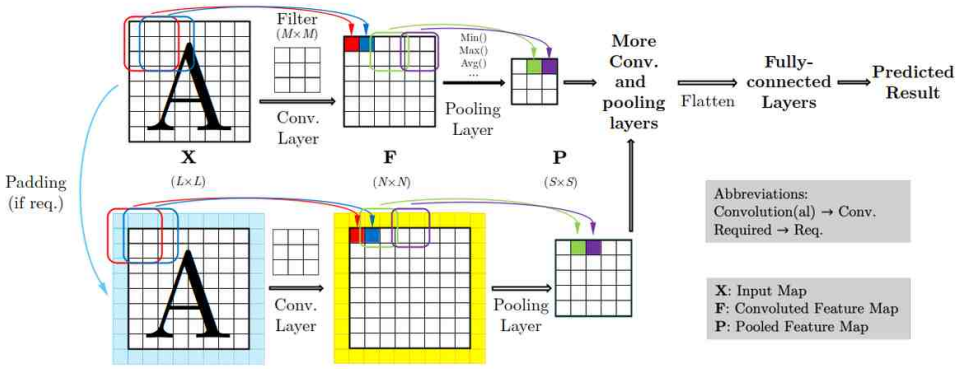


Figure 2.5: The general workflow of CNN models.

describing the convolutional layer is

$$\mathbf{F}(i,j)_{\substack{0\le i<N\\0\le j<N}} = \sigma\left(\sum_{\substack{0\le m<M\\0\le n<M}} \mathbf{K}(m,n)\cdot\mathbf{X}(i+m,j+n)+b\right) \tag{2.77a}$$

and

$$N = L + p - M + 1, \tag{2.77b}$$

where b is the bias, p is the size of the padding (see later this paragraph), i and j are the indexes of the feature map $\mathbf{F}$ satisfying $0 \le i < N$ and $0 \le j < N$, and m and n are indexes of the convolutional kernel $\mathbf{K}$ satisfying $0 \le m < M$ and $0 \le n < M$.[4] It should be noted that, the "masked" areas on the original matrix $\mathbf{X}$ can overlap each other, as shown in Figure 2.5. In addition, the output feature map A can be activated by a proper activation function σ, as Equation (2.77a) shows. In some applications the feature map $\mathbf{F}$ is required to have the same size as the original input $\mathbf{X}$. Under such circumstance the original input $\mathbf{X}$ is padded with extra elements (usually 0) of size p surrounding $\mathbf{X}$ (see cyan blocks in Figure 2.5). Then the process continues as usual.

The next step is the pooling operation. The pooling operation aims to reduce the size of the feature map $\mathbf{F}$, in order to reduce the number of trainable parameters and computational cost. The pooling operation defines a pooling windows size (or pooling kernel) k and a pooling stride s (usually $s = k$). The pooling window $(k \times k)$ defines a range in which the pooling operation takes place, and the stride s defines the distance the window shifts for the next iteration. This pooling window iterates over the entire input feature map $\mathbf{F}$ and generate the output feature map $\mathbf{P}$. In the example shown by Figure 2.5, the green

[4]For programming and expressing convenience, the indexing rule here (i, j, m and n in Equation (2.77a)) starts from 0. In later sections, the range of index will be provided if the equation follows this rule.

and purple frames denote pooling windows with size $k = 2$ and the pooling stride is given as $s = k = 2$. Unlike convolution, pooling uses a function instead of a learnable filter. Examples of functions are given as follow:

$$\mathbf{P}_{\max}(i,j)_{\substack{0\le i<S\\0\le j<S}} = \max_{\substack{0\le m<k\\0\le n<k}} \mathbf{F}(s\cdot i+m,\, s\cdot j+n) \qquad \text{for maximum pooling,} \tag{2.78a}$$

$$\mathbf{P}_{\min}(i,j)_{\substack{0\le i<S\\0\le j<S}} = \min_{\substack{0\le m<k\\0\le n<k}} \mathbf{F}(s\cdot i+m,\, s\cdot j+n) \qquad \text{for minimum pooling, and} \tag{2.78b}$$

$$\mathbf{P}_{\text{ave}}(i,j)_{\substack{0\le i<S\\0\le j<S}} = \frac{1}{k^2}\sum_{\substack{0\le m<k\\0\le n<k}} \mathbf{F}(s\cdot i+m,\, s\cdot j+n) \qquad \text{for average pooling.} \tag{2.78c}$$

Here, $\mathbf{P}$ is the output feature map after pooling, with a size of $S \times S$ and

$$S = \text{floor}\left(\frac{N+2p-k}{s}+1\right). \tag{2.79}$$

In Equations (2.78a) to (2.78c), i and j are indexes for the output feature map $\mathbf{P}$, and m and n are indexes for the pooling windows. In Equation (2.79), the function "floor" returns the greatest integer less than or equal to the input variable and p is the padding size (see yellow blocks in Figure 2.5).

In an application of a CNN model there could be more than one convolutional and pooling layer, followed by a flatten operation (convert the matrix into a 1D vector) before passing to a fully-connected structure. Although CNNs are designed for computer vision, CNNs are now widely used in other applications such as natural language processing [75] and audio recognition [76].

2.3.6 Graph Convolutional Neural Network (GCN)

The graph convolutional neural network (GCN) [77] performs a convolutional process on a graph space (non-Euclidean domain). However, unlike the CNN example given above, which is in Euclidean space, the convolution operation on graph space can be a challenge [78, 79]. The graph space G is defined as a pair consisting of a node set V and an edge set E, i.e.

$$G = (V, E), \tag{2.80a}$$

where

$$V = \{v_1, v_2, \ldots, v_n\} \tag{2.80b}$$

and

$$E = \{e_{ij} | (v_i, v_j) \in V \times V\} \tag{2.80c}$$

Here, the node set V contains n elements where $v_i \in V$ represents a node. Similarly, the edge set E has each element $e_{ij} = (v_i, v_j)$ denoting an edge connecting two nodes and pointing from v_i to v_j. For an undirected graph, $e_{ij} = \{v_i, v_j\}$, where the type of bracket

distinguishes between directed and undirected graphs.

Based on the graph definition above (Equations (2.80a) to (2.80c)), the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is defined to represent the connection status in the graph as follow

$$A_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{2.81}$$

For undirected graphs, $\mathbf{A}$ is symmetric, i.e. $A_{ij} = A_{ji}$. another matrix is defined, the degree matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, as

$$D_{ij} = \begin{cases} \sum_k A_{ik}, & i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.82}$$

Here, the degree matrix $\mathbf{D}$ is a diagonal matrix, and element D_{ii} denotes the number of nodes that node v_i is connected to (i.e. number of connections that originate from node v_i).

To carry out a convolution on the graph G, a normalised adjacency matrix $\hat{\mathbf{A}} \in \mathbb{R}^{n \times n}$ is introduced as

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{2.83a}$$

where

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I} \tag{2.83b}$$

and

$$\tilde{D}_{ii} = \sum_j A_{ij} = D_{ii} + 1. \tag{2.83c}$$

Here, $\mathbf{I}$ is an identity matrix. In Equation (2.83b), the identity matrix, called a self loop, is added to the adjacency matrix $\mathbf{A}$. This ensures the nodes keep the self feature during the convolutional process. According to the definition in Equation (2.83a), since the normalised adjacency matrix $\hat{\mathbf{A}}$ encodes the connectivity patterns of the graph and is transforms properly, the normalised adjacency matrix is a fundamental component of graph convolution that ensures a convolution operates only between connected nodes. The mathematical form of a convolutional layer can be written as

$$\mathbf{H}^{l+1} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^l\mathbf{W}^l + \mathbf{b}^l\right), \tag{2.84a}$$

where $\mathbf{H}^l \in \mathbb{R}^{n \times d_l}$ denotes the node feature matrix in network layer l, σ is the activation function, $\mathbf{W}^l \in \mathbb{R}^{d_l \times d_{l+1}}$ is the trainable weight matrix for layer l and $\mathbf{b}^l \in \mathbb{R}^{d_{l+1}}$ is the bias for layer l. Here, the node feature matrix $\mathbf{H}$ with feature dimension d can be defined

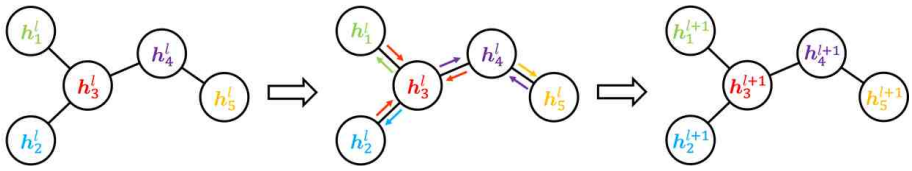


Figure 2.6: Illustration of the convolutional operation on an undirected graph. This figure shows that a node in a graph aggregates information from neighbouring nodes and updates the feature of the node itself. The arrows along the edges denotes the direction of the information aggregation. The colours of these arrows indicates the destination of the aggregation.

as

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1^\top \\ \mathbf{h}_2^\top \\ \vdots \\ \mathbf{h}_n^\top \end{bmatrix} \in \mathbb{R}^{n \times d}, \quad \text{where} \quad \mathbf{h}_i = \begin{bmatrix} H_{i1} \\ H_{i2} \\ \vdots \\ H_{id} \end{bmatrix} \in \mathbb{R}^d. \tag{2.84b}$$

Here, $\mathbf{h}_i$ is the feature vector for node i in dimension d in the current layer.

Figure 2.6 illustrates a convolutional operation on an undirected graph. During the convolutional process every node aggregates feature information form adjacent nodes and updates its own feature $\mathbf{h}_i^l$ to $\mathbf{h}_i^{l+1}$. Through multiple GCN layers, each node can encode information over the graph. These features can be extracted and passed to a later fully-connected layer for further prediction.

2.3.7 Message-passing Neural Network (MPNN)

Following the idea of GCN, J. Gimler proposed another graph-based algorithm called a message-passing neural network [80]. The MPNN is designed for solving quantum chemistry problems, especially molecular properties such as thermodynamic properties, electronic structure, vibrational properties and molecular polarity and responsiveness. In MPNN, there are three fundamental functions, the message function M_t, the update function U_t and the readout function R. Here, I provide a brief introduction to MPNN based on the same assumptions (Equations (2.80a) to (2.80c)) used in Section 2.3.5.

In a MPNN there are two main phase of operations, the message passing phase and the readout phase. In the message passing phase, first, the message function M is used to generate an aggregated message for each node from the graph structure information. The message generation process for node v_i can be written as

$$\mathbf{m}_i^{t+1} = \sum_{v_j \in \mathcal{N}(v_i)} M_t(\mathbf{h}_i^t, \mathbf{h}_j^t, e_{ij}), \tag{2.85a}$$

where

$$\mathcal{N}(v_i) = \{v_j | (v_i, v_j) \in E\}.$$

Here, the quantity $\mathbf{m}_i^{t+1}$ denotes the aggregated message for node v_i that is generated based

on the graph and features on time stage t, and $\mathcal{N}(v_i)$ represents the set of neighbouring nodes of v_i. Equation (2.85a) shows that the message functions will generate information based on the current node v_i, its neighbouring nodes in $\mathcal{N}(v_i)$ and the edge e_{ij} in between. This process generates an aggregated message $\mathbf{m}_i^{t+1}$

Next, the update function U_t is used to update the feature vector $\mathbf{h}_i^t$ for nodes in time stage t. The update process reads

$$\mathbf{h}_i^{t+1} = U_t(\mathbf{h}_i^t, \mathbf{m}_i^{t+1}). \tag{2.85b}$$

This equation shows that the update function U_t generates a new feature $\mathbf{h}_i^{t+1}$ for the next stage according to the current node feature $\mathbf{h}_i^t$ and the aggregated message $\mathbf{m}_i^t$ generated by the message function: see Equation (2.85a).

This message aggregation process (Equation (2.85a)) and node feature updating process (Equation (2.85b)) can be repeated several times, ensuring each node acquires sufficient information from the graph structure. After the message passing, the readout function R is applied to extract the result for each node from the latest feature. Assuming the graph has been updated m times, the mathematical form of this process is

$$\mathbf{y} = R\left(\{\mathbf{h}_i^m \mid v_i \in V\}\right), \tag{2.85c}$$

where the $\mathbf{y}$ is the output of the MPNN.

In a MPNN, the message function M, update function U and readout function R can contain trainable parameters, and the definitions of these functions may vary across different models. According to Reference [80], there are several existing MPNN-like models, such as convolutional network for learning molecular fingerprints [81], gated graph neural network (GG-NN) [82], interaction networks [83], etc. According to J. Gilmer, these existing models can be rewritten in MPNN-like form. In the following paragraphs, a short summary of GG-NN is presented in MPNN-like format as an example, based on Gilmer's publication in 2020 [84].

The message function in a GG-NN is designed as

$$M_t^{\text{GG-NN}}(\mathbf{h}_i^t, \mathbf{h}_j^t, e_{ij}) = A_{ij}\mathbf{h}_j^t, \tag{2.86a}$$

where A_{ij} in matrix $\mathbf{A}$ is the learnable parameter for edge e_{ij}. The update function in a GG-NN reads

$$U_t^{\text{GG-NN}} = \text{GRU}(\mathbf{h}_i^t, \mathbf{m}_i^{t+1}). \tag{2.86b}$$

Here, GRU denotes the gated recurrent unit (GRU) [73]. The GRU is a kind of improved recurrent neural network (introduced in Section 2.3.4) that mitigates the vanishing and exploding gradient problems in standard RNNs. Since a GG-NN is a weight-tying model (i.e. the weight matrix is shared amount update function), the update function applied at

each time step is the same. The readout function in a GG-NN is

$$R^{\text{GG-NN}} = \sum_{v_i \in V} \sigma\left(a(\mathbf{h}_i^m, \mathbf{h}_a^0)\right) \odot \left(b(\mathbf{h}_i^m)\right), \tag{2.86c}$$

where i is the index in node set V, and m is the last time stage (i.e. the update process runs m times). Here a and b denote two separate modules in a GG-NN, generating a feature based on the initial state of a node and the final state of a node respectively. The readout function R create an element-wise product (operator $\odot$) as the final output of the graph.

Chapter 3

Dataset

In this section I introduce the datasets that are used in this work. For a wider performance evaluation for my models, I obtain several datasets for training and testing purposes. Table 3.1 shows all datasets that have been obtained and their origins. In total 16 datasets are used in this work, originating from three different sources (GDB-13 [85], ChEBLE-35 [86, 87] and ZINC20 [88–90]). Eight datasets (named without a prefix) are from the GDB-13 database, four datasets (named with "LM" as a prefix) are built on selected molecules from the ChEMBLE-35 database and two datasets (named with "SE" as a prefix) are sourced from the ZINC20 database.

In Table 3.2 I present the uses of all the obtained datasets, showing which datasets each model is trained and tested on. In the atomic charge prediction project (see Chapter 4), I built five models that can predict different charges. Each model in the atomic charge prediction project has one training and at least one testing set. For model "Mulliken", model "Hirshfeld" and model "CM5", I applied another testing set (see Section 3.2) for each model, to verify the performance of my algorithm with datasets with larger molecules. In the structure prediction project I created a set of models based on the DFT-optimised structures (model "DFT"). Although another model was initially planned to be trained on semi-empirical structures from the ZINC database, the training process was terminated due to the large number of faulty structures in the dataset and persistent memory overflows on the testing machine. The model was trained with one dataset (see Section 3.1) and tested with a standard testing set (see Section 3.1) and a testing set with larger molecules (see Section 3.2).

In the following section I introduce all 16 datasets. They are catagorised based on their sources: GDB-13, ChEMBL-35 and ZINC20.

3.1 GDB-13-based Training and Testing Datasets

The molecular structures in the GDB-13 database are provided in the Simplified Molecular Input Line Entry System (SMILES) format [91, 92]. SMILES represents molecules as

Table 3.1: The origination of datasets

Datasets	Original Source	Datasets	Original Source
Mulliken_Train Mulliken_Test Hirshfeld_Train Hirshfeld_Test CM5_Train CM5_Test DDEC6_Train DDEC6_Test Struct_Train Struct_Test	GDB-13	LM_Mulliken_Test LM_Hirshfeld_Test LM_CM5_Test LM_Struct_Test	ChEMBLE-35
		SE_Charge_Train SE_Charge_Test	ZINC20

Table 3.2: Datasets used for each model in this work

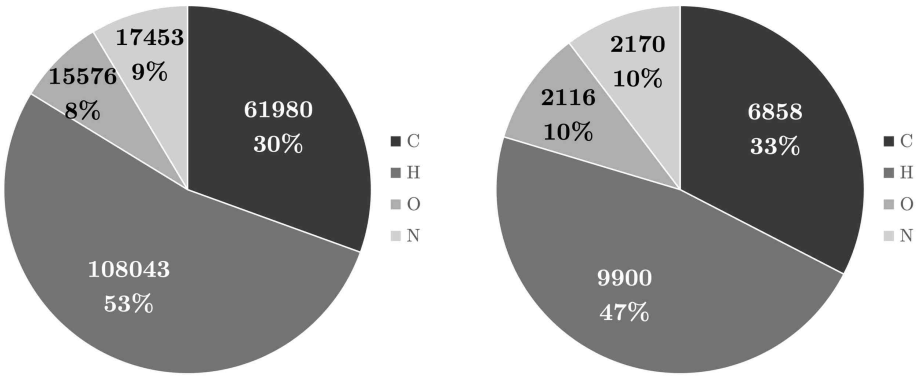
Project	Model	Training Set	Testing Set 1	Testing Set 2
Atomic Charge Prediction	Mulliken	Mulliken_Train	Mulliken_Test	LM_Mulliken_Test
	Hirshfeld	Hirshfeld_Train	Hirshfeld_Test	LM_Hirshfeld_Test
	SE	SE_Charge_Train	SE_Charge_Test	N/A
	CM5	CM5_Train	CM5_Test	LM_CM5_Test
	DDEC6	DDEC6_Train	DDEC6_Test	N/A
Structure Prediction	Struct	Struct_Train	Struct_Test	LM_Struct_Test

ASCII strings and it is widely used as input in many machine learning models for various molecular properties prediction [93–95], including this work. In this section I present the approach taken to go from the GDB-13 set of molecules in SMILES notation to the corresponding atomic charges and structural information datasets. I also provide an overall summary of the molecules in the datasets.

3.1.1 Data Selection

The GDB-13-based training and testing dataset used in this work is a part of the CNO subset (carbon, nitrogen and oxygen only) of the GDB-13 (Generated Database 13) database [85]. GDB-13 is a computer-generated exhaustive list of molecules, with each molecule containing no more than 13 non-hydrogen atoms. The generation of molecules follows the rules of valence and excludes unstable structure (such as tricyclic bridge-heads) [96]. The list of all selected molecules in both training and testing sets is provided in Appendix C.

I choose the GDB-13 database as it contains a sufficient number of small molecules to allow me to reach the key objective of this works which is to design a machine-learning algorithm that can be trained with small molecules and applied on larger molecules. To simplify and minimise the cost of DFT calculations used to generate the dataset, I choose to focus on only four elements – carbon, nitrogen, oxygen and hydrogen. This should be enough to verify the feasibility and performance of my algorithms.



(a) Training Datasets (b) Testing Datasets

Figure 3.1: The element distribution of GDB-13-based datasets

3.1.2 Summary of Selected Molecules

In total, 13,517 molecules are extracted from the GDB-13.CNO database. I used a train-test split based on a ratio of 9 : 1. Thus, the GDB-13-based training sets contain 12,124 molecules and testing sets contain 1,393 molecules[1]. The GDB-based training sets have a total of 203,052 atoms (each molecule contains up to 23 atoms), while the testing sets contain a total of 21,044 atoms (each molecule contains up to 21 atoms). The number and proportion of each element is presented in Figure 3.1. By comparing Figures 3.1(a) and 3.1(b) it can be seen that the proportion of each element is close between the training and testing sets, with about 50% hydrogen atoms, 30% carbon atoms, 10% oxygen and nitrogen atoms.

3.1.3 Generation Workflow

The generation workflow of all GDB-13-based datasets is given in Figure 3.2. Here I give a general introduction to this workflow, and detailed information for each step is given in the following sections. The input molecules obtained from the GDB-13 database are in SMILES format. Thus, I first use Open Babel [97] to covert the SMILES strings into the xyz format, with coordinate structure generation and MD-based geometry optimisation. After that the coordinates are passed to Gaussian16 [98] for a DFT calculation, from which are obtained the Mulliken charge, Hirshfeld charge, CM5 charge and DFT-optimised geometry. At the end the wave functions (wfx file) output by Gaussian are passed to Chargemol [38–41] to generate DDEC6 charges.

Open Babel Conversion and MD Optimisation

The first process in this workflow is to generate the MD-optimised xyz coordinates, which can then be passed to DFT calculations. The software I use here is Open Babel [97], a format conversion tool with build-in conformation and geometry optimisation. The bash command of this operation is given as follow:

[1]Here, the split is not exactly 1 : 9 due to unsuccessful operations during the generation workflow.

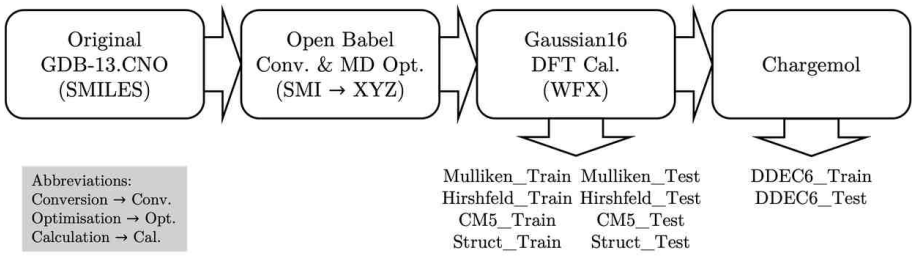


Figure 3.2: General generation workflow of the GDB-13-based datasets.

```
obabel -ismi ${smi_file} -oxyz ${output_file} -m -h --gen3D
    --best
```

-ismi denotes the input files are in SMILES format

-oxyz denotes the output format is xyz

-m means saving each molecule as an individual file

-h means adding hydrogen atoms (hydrogen atoms are omitted in the SMILES format) to the output

–gen3D –best I generated the 3D coordinates for every molecule, and ran a force-field based geometry optimisation with the highest preset accuracy. See below for more information

The conformational search and force-field based geometry optimisation used in Open Babel is based on the method introduced by N. Yoshikawa [99]. In the conformational optimisation process, the software first breaks down the molecular SMILES strings into several known fragments, and searches a library for a possible conformational structure with the lowest computational energy. Open Babel then applies a further 250-step steepest descent geometry optimisation with the MMFF94 forcefield [100–104], following by a Weighted Rotor conformational search and conjugate gradient geometry optimization [105]. It should be noted that the conformational search method described above yields only one possible conformation for a given SMILES string. However, the conformational dependence becomes more pronounced in extended molecular systems due to their increased structural complexity and accessible conformational space. This limitation may result in incomplete learning of conformational space by the machine learning model. Despite the limitation above, these optimised structures by Open Babel should be capable to validate the algorithms introduced in Chapters 4 and 5, and they are suitable inputs for the next-step DFT calculations.

Gaussian16 DFT Calculations

Gaussian DFT calculation are used to obtain more accurate geometries, as well as the Mulliken, Hirshfeld and CM5 charges which are available in Gaussian16. The Gaussian16

input file header is given as follow:

```
%NProcShared=16
#n B3LYP/6-31G(d) Opt pop=Hirshfeld scf=tight output=wfx

 molecule_name
 the MD-optimised coordinates are not list in this example
    header (replace ...)

0 1
...

molecule.wfx
```

From the above Gaussian settings it can be seen that the calculation did the following:

%NProcShared =16 16 cores were requested for the calculations. This is sufficient as each molecule contains no more than 23 atoms.

#n B3LYP/6-31G(d) I chose the B3LYP functional [54] and 6-31G(d) basis set [106, 107]. The B3LYP functional is found empirically to be suitable for small molecules that consist only of carbon, hydrogen, oxygen and nitrogen atoms. The 6-31G(d) basis set is a Gaussian-type double ζ basis set, with polarisation functions. A further discussion of basis set is given in later paragraph. This combination of functional and basis set can achieve an acceptable balance of efficiency and accuracy [108, 109]. Efficiency is important because calculations need to be performed for a total of 13517 small molecules from the GDB-based datasets. There are also some large molecules containing 50-100 heavy atoms for each molecule from the larger molecules databases (introduced in Section 3.2).

Opt During the DFT calculations I run a geometry optimisation.

pop=Hirshfeld The Hirshfeld population is computed.

scf=tight The self-consistency is set to "tight" to achieve better accuracy.

output=wfx The wavefunctions are written to a file (molecule.wfx).

0 1 I set the overall charge number to zero and the multiplicity to a singlet since there are no free radicals and all molecules are neutral and calculated in the ground state.

I now justify the choice of basic set. The purpose of this work is to investigate the performance of machine leraning models that predict atomic charges. For this we need realistic and consistent training and testing data. Having highly accurate data would be nice, but is not essential for this project. This gives me a bit of freedom to choose a basis set that is computationally efficient, which is helpful as calculations must be performed on a very large number of molecules: the GDB-based datasets contain 13,517 small molecules (224,096 atoms) and the ChEMBL-based datasets contain 139 larger molecules (19,151

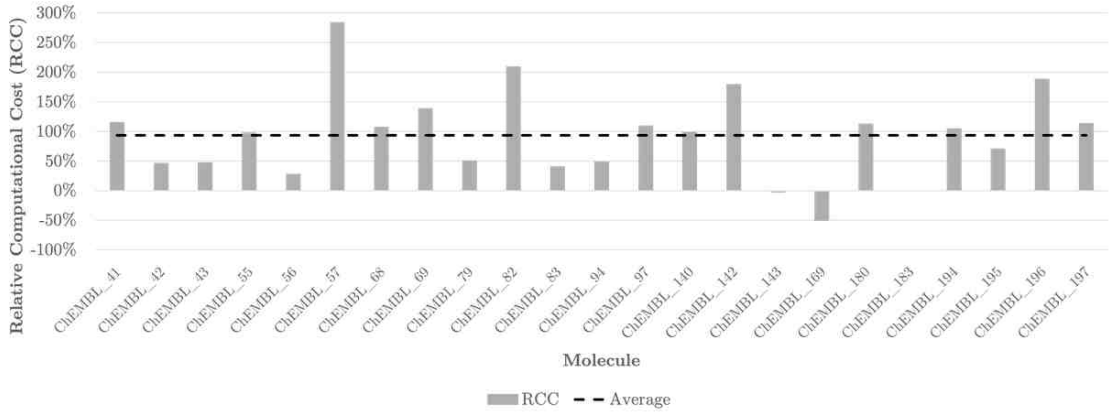


Figure 3.3: The relative computational cost (RCC) of B3LYP/Def2SVP over B3LYP/6-31G(d) on 23 larger molecules.

atoms, see Section 3.2 for more information). The increase in the amount of data matters more than high of accuracy of the data. A double ζ with polarization basis set is a natural choice for this work as it balances the accuracy and efficiency. By contrast, a single ζ basis set would be unacceptably inaccurate, and triple/quadruple ζ basis sets are unaffordable for such a large number of calculations.

According to Reference [109], two double ζ basis sets, 6-31G(d) and Def2SVPD, are competitive for this work. N. Mardirossian reported that the Def2SVPD, which has p-orbital polarisation for hydrogen atoms and extra diffuse functions (compared to 6-31G(d)), has significant advantage over 6-31G(d) in accuracy (see Figure 27 in Reference [109]). It should be noted that the targeted DFT results in this work are only for the ground-state of neutral small molecules[2]. Thus, the use of diffuse functions is not essential since it may introduce significant cost for the DFT calculations while providing limited improvement in accuracy.

I here run a test to compare the time efficiency of the two basis sets on 23 randomly chosen molecules from the ChEMBL-35 database. Figure 3.3 shows the time efficiency comparison of 6-31G(d) and Def2SVP using the same functional (B3LYP). I used the following comparison metric named the relative computational cost (RCC),

$$\mathrm{RCC} = \frac{T_{\mathrm{Def2SVP}} - T_{\mathrm{6\text{-}31G(d)}}}{T_{\mathrm{6\text{-}31G(d)}}} \times 100\%, \tag{3.1}$$

where T_{Def2SVP} and $T_{\mathrm{6\text{-}31G(d)}}$ denote the CPU time of a DFT calculation using the Def2SVP and 6-31G(d) basis sets respectively. Figure 3.3 indicates that for 20 out of 23 molecules Def2SVP costs more time than 6-31G(d). On average the Def2SVP basis set costs 93.3% more time than the 6-31G(d) basis set. The original data for this set is presented in Appendix F. Because of its greater efficiency, I use the 6-31G(d) basis set for all the following DFT calculations.

[2]The molecules (50-100 heavy atoms) in larger molecule datasets (introduced in Section 3.2) can be regarded as small molecules.

From the Gaussian16 output log file, the Mulliken charge, Hirshfeld charge, CM5 charge and optimised geometry in a vacuum environment ($\varepsilon = 1$) can be obtained. It is true that the use of a vacuum environment with $\varepsilon = 1$ in this context does not accurately reflect the actual conditions of molecular docking in drug development. This discrepancy arises because the original purpose of this dataset was not drug development, and time constraints prevented the regeneration of appropriate DFT data. While the use of vacuum environment data is slightly inappropriate here, we believe it remains sufficient for conducting fundamental testing and validation of the MPF and GLaCE algorithms. Thus, several training and testing datasets can be set up, including datasets "Mulliken_Train", "Mulliken_Test", "Hirshfeld_Train", "Hirshfeld_Test", "CM5_Train", "CM5_Test", "Struct_Train" and "Struct_Test" (see the Gaussian16 DFT calculation in Figure 3.2). During the Gaussian16 DFT calculations, a "wfx" wave function file for each molecule is generated, as the input for later DDEC6 charge calculation.

Chargemol DDEC6 Charge Calculation

To generate DDEC6 charges [38, 39] for both training and testing datasets, a piece of software named Chargemol [110] is used. An example input file for this program is given below:

```
    <atomic densities directory complete path>
    /path/to/atomic/densities
    </atomic densities directory complete path>
    <input filename>
    wfx wave funciton file (calculated by Gaussian16)
    </input filename>
    <charge type>
    DDEC6
    </charge type>
    <compute BOs>
    .false.
    </compute BOs>
```

The atomic densities are computed using a preset library provided by Chargemol. The script above shows that the target of the Chargemol calculation is the DDEC6 charges. The bond order calculations (which is not required here) are skipped. In summary, the datasets "DDEC6_Train" and "DDEC6_Test" can be set up.

3.2 ChEMBL-35-based Testing Datasets

Apart from the standard GDB-based testing set, I created another testing set that consist of larger molecules than the training set, namely the larger molecules testing datasets (LM testing sets). I chose the ChEMBL-35 dataset [86, 87] as the source of my larger molecules testing set. ChEMBL is a large database that contains bioactive molecules with drug-like properties. As the molecules in ChEMBL exist in real world, I believe creating a testing

set based on the ChEMBL database can partly show the performance of my model in practice. Moreover, the molecules I selected (see below) from the ChEMBL database are much larger than the molecules in my GDB-13 training sets. By testing the GDB-trained model on LM testing sets, I can show that my models can satisfy the key objectives of this project (as mentioned in the introduction), i.e. the models can be trained with small molecules but can be applied in larger molecules.

3.2.1 Data Selection

The criteria for selecting molecules from ChEMBL-35 are:

- Number of heavy atoms ranges from 50 to 100, inclusive.
- Elements limited to C, H, O and N only.
- Zero overall charge on a molecule.
- Each molecule is a single connected compound (no fragments).

I choose 139 molecules that each contain 50-100 heavy atoms (maximum 213 atoms per molecule), since these molecules are larger than the GDB-based training sets (no more than 8 heavy atoms in each molecules) but computationally acceptable (not too expensive). Due to the elemental limitation of my model that trained on GDB-based datasets, the selected molecules in the LM testing sets only contain C, H, O and N atoms. In addition, I also constraint the overall charge of molecules to be zero, as well as the molecules should be a single connected compound. This is due to the limitation of the GDB-13-based training sets: cases apart from these do not exist in the training sets and thus the trained model may not be accurate for other cases. The chosen molecules in the LM testing set are listed in Appendix D, and the Python script for filtering molecules is provided in Appendix G. Readers should note that, although I extract 200 molecules from the ChEMBL-35 database, only 139 molecules are successfully calculated through the workflow given in Section 3.2.3. In total, 61 calculations failed or terminated in either the MD optimisation or the DFT calculation due to errors or time exceeding the allocated limit (10 minutes for an MD optimisation and 24 hours for a DFT calculation).

3.2.2 Summary of Selected Molecules

The ChEMBL-based LM testing sets contain a total of 19,151 atoms. The element distribution of LM testing sets is provided in Figure 3.4. In the LM testing sets (including datasets "LM_Mulliken_Test", "LM_Hirshfeld_Test", "LM_CM5_Test" and "LM_Struct_Test"), 35% of those are carbon, 51% of those are hydrogen, 10% of atoms are oxygen and the remaining 4% of atoms are nitrogen. The number of each element in these datasets is also provided in Figure 3.4. Here I aim to generate a larger-molecule testing set with a similar size to the GDB-based testing sets (21,044 atoms). However, due to their fundamentally different generation methods (GDB is computer enumerated while ChEMBL is a collection of existing molecules), the GDB-based testing datasets and LM testing sets cannot achieve

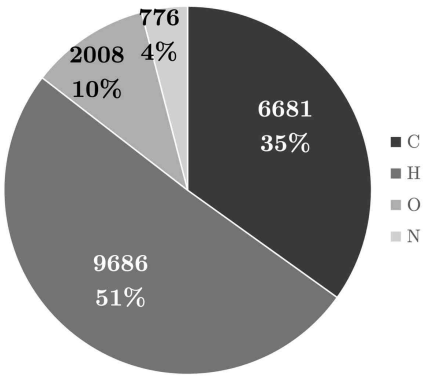


Figure 3.4: The element distribution of ChEMBL-35-based testing datasets

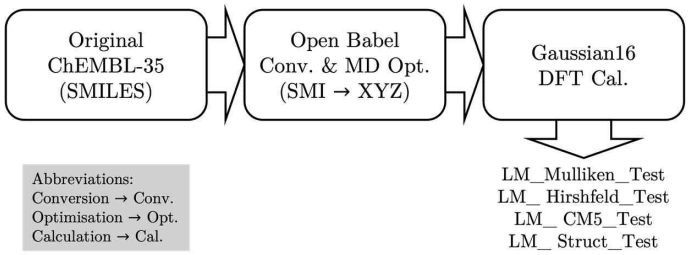


Figure 3.5: General generation workflow of the larger molecule testing datasets

identical element distribution. The key difference in the element distribution is that the nitrogen atoms in LM testing sets (4%) is much less than that in GDB-based testing sets (10%). Readers should note that this could reduced the reliability of the testing accuracy on the properties for nitrogen atoms on LM testing sets, because of the reduced total number of nitrogen samples.

3.2.3 General Workflow

The workflow for generating LM testing sets is given in Figure 3.5. Similar to the GDB-based datasets mentioned above, the ChEMBLE-35 database entries are also provided as SMILES strings. By doing the format conversion and geometry optimisation with Open Babel, the xyz coordinates for molecules can be obtained, and which can be later passed to Gaussian16 for DFT calculations. All processes are introduced in following paragraphs.

Open Babel Conversion and Geometry Optimisation

Here I follow exactly the same conversion and geometry optimisation procedure for these molecules from ChEMBLE-35 as for the small molecules: see Section 3.1.3.

Gaussian16 DFT Calculation

In order to keep the consistency of GDB-based datasets and LM testing sets, I use the same functional (B3LYP) and basis set (6-31G(d)) for the DFT calculation. Due to the larger size of molecules in ChEMBL database, the number of cores requested is increaed

to 32, and the memory requested is 128GB: see below. In addition, the wavefunction output is disabled since there is no following DDEC6 charge calculation due to the high cost of DDEC6 charge calculation in larger molecules. For all other information, please see Section 3.1.3.

```
%NProcShared=32
%Mem=128GB
#n B3LYP/6-31G(d) Opt pop=Hirshfeld scf=tight

...
```

From the DFT calculations the Mulliken charge, Hirshfeld charge, CM5 charge and DFT-optimised geometry can be obtained. Thus, the LM testing sets, i.e. datasets "LM_Mulliken_Test", "LM_Hirshfeld_Test", "LM_CM5_Test" and "LM_Struct_Test", can be built.

3.3 ZINC-based Training and Testing Datasets

In order to test my models with larger training and testing datasets and validate the performance in wider range of elements, I generated two datasets base on the ZINC20 database [90]. ZINC20 is an ultralarge-scale database for ligands. It provides millions of molecules in mol2 format with semi-empirical structures and charges generated by AMSOL [111]. The details of the selected ZINC database are provided below.

3.3.1 Data Selection

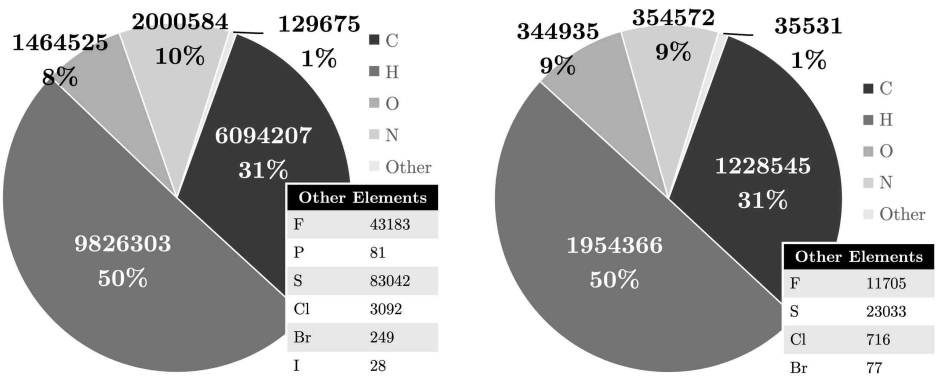
As the size of ZINC database exceeds the capacity of the available machine, I first randomly chose two sets of data for training and testing purpose respectively (9 : 1 ratio), given in Appendix E. Then, I applied a filtering process according to the following criteria that are embedded in the program "prj_acdm" (introduced in Chapter 6):

- Limited the elements to C, H, O, N, F, S, Cl and Br.
- Deleted the molecules without charge information.

Finally, 492,162 molecules (no more than 52 atoms per molecule) are selected for the training sets and 100,000 molecules (no more than 51 atoms per molecule) are selected for the testing sets for the atomic charge prediction project. A detailed summary of the ZINC-based datasets are provided below. The names start with "SE", standing for semi-empirical.

3.3.2 Summary of Selected Molecules

A brief summary of the element distribution is provided in Figure 3.6. In total, 19,515,294 atoms are included in training sets and 3,907,949 atoms are included in testing sets. For the distribution of elements, both training and testing sets have around 30% carbon, 50% hydrogen, about 8% oxygen, roughly 10% nitrogen and 1% other elements (i.e. F, S, Cl and Br, provided in Figure 3.6).



(a) Training Datasets

(b) Testing Datasets

Figure 3.6: The element distribution of ZINC20-based datasets

I would like to determine how many times a particular atom must appear in the dataset to provide robust training for ZINC-based datasets. The dataset features 8 elements that are included in both the training and testing sets, with various numbers of atoms for each element. This can not only show the performance of the models over a larger range of elements, but can also provide information on the minimum required amount of data for a reliable prediction in later models. Due to the limited number of Cl and Br samples in the training set, the predicted results may not be accurate. Detailed information is provided in later chapters.

Since the ZINC20 database contains empirical charges and structures, there is no further operation required. From above, the datasets "SE_Charge_Train" and "SE_Charge_Test" can be set up.

Chapter 4

Project: Atomic Charge Prediction

In this section, I introduce a coordinate-free and low order scaling ML method that can be applied to various kinds of atomic charges, which can be an important property for ligand docking. This work is named the Atomic Charge Prediction project (abbreviated as project ACP). This method is based on an original idea, namely a Message-Passing Featuriser, abbreviated as MPF. The MPF is based on the Message-Passing Neural Network (MPNN). From the introduction to MPNNs presented in Section 2.3.7, I observe several limitations in MPNNs for our target application of virtual screening. See below.

The first limitation is that, MPNN models are unable to be trained with small molecules and make predictions for larger molecules. This is because of limitations on the acceptable size of molecules when predicting since the input molecular graph size has been already fixed when creating the model. From Equation (2.86a) in the GG-NN example (Section 2.3.7), the size of the learnable matrix $\mathbf{A}$ depends on the number of edges in the input molecular graph. Thus, a trained GG-NN model can only predict properties for molecules whose size does not exceed that of the learnable matrix $\mathbf{A}$. If the size of input graphs varies then the model must either pad smaller graphs with zeros, or attempt to predict larger graphs using insufficient parameters, leading to degraded performance or even failure. If we can overcome this limitation, the model can be trained on small molecules and then predict properties for larger molecules. This feature can significantly reduce the cost of training and the trained model can be more flexible for the size of input molecules.

The second limitation is that some of the existing models [62, 112] use coordinates as model input that require molecular dynamics optimisation or even quantum chemistry calculations before predicting properties for molecules. This can be extremely costly for large molecules. In MPF, I eliminate any MD or QM calculation in the prediction process. However, as MPF-based models are based on supervised learning, MD and QM calculations are still required in the training process. With the improvement stated above (the first limitation), the cost of the training process has been already reduced. By removing the requirement of pre-calculations, MPF-based models can accept SMILES as input in the prediction process, and predict the properties directly.

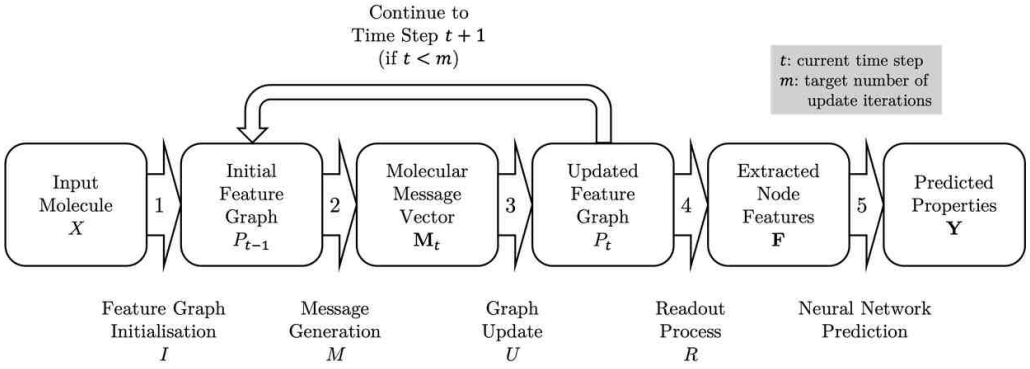


Figure 4.1: General process of message-passing featuriser.

The third problem is efficiency. Due to my limited access to high-performance GPUs[1], the original algorithm for MPNNs is unaffordable in this work. Since the trainable parameters are embedded in the message function, update function and readout function, the entire MPNN process is run through in each epoch during the training process of the model. This repetition significantly increases the cost of training in terms of both time and computational resources. By eliminating this repetition, the training cost of an MPF-based model can be further reduced.

In this work, I designed the MPF algorithm to address the problems mentioned above. The key objectives of MPF-based models are

- accepting any size of molecule after being trained,
- accepting SMILES as input without pre-calculations, and
- minimising message-passing repetition to reduce the training cost.

The methodology and result analysis are provided in later sections. This was published in 2024, see Reference [113].

4.1 Methodology

In this methodology section I provide the definition of the MPF and the implementation details for the atomic charge prediction project. Since the notation in the following sections is complex, a notation table is provided in Appendix A.

4.1.1 Message-passing Featuriser (MPF)

Compared to a classic MPNN, the major difference of the MPF is replacing the trainable parameters in the MPNN. Instead, the message-passing process is used as a featuriser, i.e. a structure that can extract features from the input variables, and the trainable parameters are moved to the later fully-connected layers. The general process of the MPF is given in Figure 4.1 as operations (arrows) 1-4. Further introduction is given below.

[1] short for graphics processing unit

Definitions of Input Molecule

The input for MPF is the molecule X. In my program "prj_acdm" (see Chapter 6), the only accepted format of molecules is mol2. For SMILES input, the program can automatically convert SMILES strings to mol2 format via Open Babel [97] without generating coordinates. Molecule X is a set including the atom set V, the bond set E, the adjacency matrix $\mathbf{A}$, the topology matrix $\mathbf{T}$ and the atomic constant properties set C. The mathematical form of the molecule set for a n-atom molecule is

$$X = \{V, E, \mathbf{A}, \mathbf{T}, C\}\,, \tag{4.1a}$$

where the atom set is

$$V = \{v_1, v_2, \ldots, v_n\}\,, \tag{4.1b}$$

the bond set E is

$$E = \{e_{ij} | \{v_i, v_j\} \in V \times V, i \neq j\}\,. \tag{4.1c}$$

and the element in adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ is

$$A_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{4.1d}$$

In the above definitions, I follow the conventional notation of GCNs (see Section 2.3.6), since molecules are represented as graphs in MPF. Thus, Equations (4.1b) to (4.1d) are the same as Equations (2.80b), (2.80c) and (2.81). I here define a topology matrix $\mathbf{T} \in (\mathbb{N}^*)^{n \times n}$ that includes the bond order information, i.e.

$$T_{ij} = \text{Bond}(v_i, v_j), \tag{4.1e}$$

where the function Bond represents the bond order using integers. Function Bond follows the definition of bond in the mol2 format, which is

$$\text{Bond}(v_i, v_j) = \begin{cases} 0, & \text{atoms } v_i \text{ and } v_j \text{ are not connected,} \\ 1, & \text{atoms } v_i \text{ and } v_j \text{ form a single bond,} \\ 2, & \text{atoms } v_i \text{ and } v_j \text{ form a double bond,} \\ 3, & \text{atoms } v_i \text{ and } v_j \text{ form a triple bond,} \\ 4, & \text{atoms } v_i \text{ and } v_j \text{ form an amide bond,} \\ 5, & \text{atoms } v_i \text{ and } v_j \text{ form an aromatic bond.} \end{cases} \tag{4.1f}$$

The atomic properties set C is a user-defined set (for the definition used in the project ACP, see Equation (4.6a)), which can include constant properties of atoms or elements, such as atomic mass, Pauling electronegativity, etc.

Feature Graph Initialisation

The MPF first converts the input molecule X into an initial feature graph with the initialisation function I, as operation 1 shown in Figure 4.1. The initial feature graph P_0 is the starting graph of the later update iterations. The initial graph is defined as

$$P_0 = \{X, \mathbf{H}_0\}, \tag{4.2a}$$

where $\mathbf{H}_0$ is the initial stage of molecular feature vector $\mathbf{H}_t \in \mathbb{R}^{n\times d}$. The molecular feature vector $\mathbf{H}_t$ contains the features for all atoms in the molecules for time step t, and each atom has d features. Thus, the molecular feature vector $\mathbf{H}_t$ can be written as

$$\mathbf{H}_t = \begin{bmatrix} \mathbf{h}_{t,1} \\ \mathbf{h}_{t,2} \\ \vdots \\ \mathbf{h}_{t,n} \end{bmatrix} = \begin{bmatrix} \mathrm{h}^1_{t,1} & \mathrm{h}^2_{t,1} & \cdots & \mathrm{h}^d_{t,1} \\ \mathrm{h}^1_{t,2} & \mathrm{h}^2_{t,2} & \cdots & \mathrm{h}^d_{t,2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{h}^d_{t,n} & \mathrm{h}^d_{t,n} & \cdots & \mathrm{h}^d_{t,n} \end{bmatrix}, \tag{4.2b}$$

where $\mathbf{h}_{t,i} \in \mathbb{R}^d$ denotes the atomic feature vector for atom i at time step t and $\mathrm{h}^j_{t,i}$ represents the feature j for atom i at time step t.

To obtain the initial molecular feature vector $\mathbf{H}_0$, the initialisation function I is applied to the input graph. This process can be stated in three ways as

$$\begin{aligned} \mathbf{H}_0 &= I(X) && \text{per molecule,} \\ \mathbf{h}_{0,i} &= I'(X,i) && \text{per atom,} \\ \mathrm{h}^j_{0,i} &= I'_j(X,i) && \text{per feature.} \end{aligned} \tag{4.2c}$$

Equation (4.2c) also indicates that the molecular initialisation function I can be decomposed into atomic initialisation function I' or further into a per-feature initialisation function I'_j. In addition, this initialisation function I (as well as I' and I'_j) is user-defined. For a detailed definition in project ACP, see Equation (4.7b).

Message Generation

The message generation process (operation 2 in Figure 4.1) is the first operation in the update loop. In this process, the molecular message vector $\mathbf{M}_t \in \mathbb{R}^{n\times k}$ (k denotes the number of messages in each atomic message vector $\mathbf{m}_{t,i} \in \mathbb{R}^k$) for time stage t is generated, which is defined as

$$\mathbf{M}_t = \begin{bmatrix} \mathbf{m}_{t,1} \\ \mathbf{m}_{t,2} \\ \vdots \\ \mathbf{m}_{t,n} \end{bmatrix} = \begin{bmatrix} \mathrm{m}^1_{t,1} & \mathrm{m}^2_{t,1} & \cdots & \mathrm{m}^k_{t,1} \\ \mathrm{m}^1_{t,2} & \mathrm{m}^2_{t,2} & \cdots & \mathrm{m}^k_{t,2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{m}^1_{t,n} & \mathrm{m}^2_{t,n} & \cdots & \mathrm{m}^k_{t,n} \end{bmatrix} \tag{4.3a}$$

Here, $\mathbf{m}_{t,i}$ represents the atomic message vector for atom i at time stage t and $\mathrm{m}^j_{t,i}$ denotes the specific message j.

The molecular message vector $\mathbf{M}_t$ is generated by the message generation function M_t (for time step t). This process can be written as

$$\begin{aligned} \mathbf{M}_t &= M_t(P_{t-1}) && \text{per molecule,} \\ \mathbf{m}_{t,i} &= M'_t(P_{t-1}, i) && \text{per atom,} \\ \mathrm{m}^j_{t,i} &= M'_{t,j}(P_{t-1}, i) && \text{per feature.} \end{aligned} \tag{4.3b}$$

Here, the molecular message generation function M_t can also decomposed into atomic message generation function M'_t and per-message generation function $M'_{t,j}$. Please note that the notation for message generation functions M_t, M'_t and $M'_{t,j}$ (typeset in italic) and the molecular message vector $\mathbf{M}_t$ (typeset in bold Roman) are distinct. In addition, the message generation functions can vary with different applications and at different time step. The precise form of the message functions used in the atomic charge prediction project is provided in Equations (4.8b) and (4.8c).

Graph Update

The operation 3 given in Figure 4.1 is the update process. The update operation generates a new graph

$$P_t = \{X, \mathbf{H_t}\}, \tag{4.4a}$$

which contains new molecular feature vector $\mathbf{H}_t$ (see Equation (4.2b) for definition). This new molecular feature vector $\mathbf{H}_t$ is calculated by the molecular update function U_t, according to the previous feature graph P_{t-1} and the molecular message vector $\mathbf{M}_t$ generated in current stage t (see Equation (4.3b)). This process can be written as

$$\begin{aligned} \mathbf{H}_t &= U_t(P_{t-1}, \mathbf{M}_t) && \text{per molecule,} \\ \mathbf{h}_{t,i} &= U'_t(P_{t-1}, \mathbf{M}_t, i) && \text{per atom,} \\ \mathrm{h}^j_{t,i} &= U'_{t,j}(P_{t-1}, \mathbf{M}_t, i) && \text{per feature.} \end{aligned} \tag{4.4b}$$

where U_t is the molecular update function for time step t, which is decomposable (into atom-level U'_t and feature-level $U'_{t,j}$) and customisable (see Equation (4.9) for definition used in this project). Here, the MPF iterates within the message generation and graph update process (operation 2 and 3 in Figure 4.1) for several update times. Therefore, the information of each atom can be passed to further atoms, i.e. more information can be aggregated into each atom. This can ensure the feature extracted by MPF contains sufficient information about the surrounding environment of the atom.

Feature Readout

After the update iterations, the MPF reads out the molecular final features $\mathbf{F} \in \mathbb{R}^{n \times s}$ (s denotes the number of features for each atom) for later NN prediction, given as operation 4 in Figure 4.1. This molecular final feature $\mathbf{F}$ can be decomposed into atom-level and

feature-level as

$$\mathbf{F} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_n \end{bmatrix} = \begin{bmatrix} \mathrm{f}_1^1 & \mathrm{f}_1^2 & \cdots & \mathrm{f}_1^s \\ \mathrm{f}_2^1 & \mathrm{f}_2^2 & \cdots & \mathrm{f}_2^s \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{f}_n^1 & \mathrm{f}_n^2 & \cdots & \mathrm{f}_n^s \end{bmatrix}, \tag{4.5a}$$

where $\mathbf{f}_i$ is the atomic final feature vector for atom i and f_i^j represents the final feature j of atom i.

The readout function R is applied to the final stage of feature graph P_m and molecular message vector $\mathbf{M}_m$ to obtain the final features. The mathematical forms of this process are

$$\begin{aligned} \mathbf{F} &= R(P_m, \mathbf{M}_m) && \text{per molecule,} \\ \mathbf{f}_i &= R'(P_m, \mathbf{M}_m, i) && \text{per atom,} \\ \mathrm{f}_i^j &= R'_j(P_m, \mathbf{M}_m, i) && \text{per feature.} \end{aligned} \tag{4.5b}$$

Similarly, the readout functions (R, R' and R'_j) are customisable. For the detailed definition of readout function and features in this project, see Equations (4.10b) to (4.10f).

Neural Network Prediction

Equation (4.5b) shows one of the most important features of MPF — the MPF-output feature vector $\mathbf{f}$ is for a single atom and is independent from other feature vectors. Thus, the neural network process can take one feature vector each time and predict the properties for each atom independently (see operation 5 in Figure 4.1). This can ensure that the MPF-based model adapts freely to any size of input molecule after being trained, as the neural network regards each atom in the molecule as an independent sample. In addition, the training cost of an MPF-based model can also be significantly decreased since the message-passing and updating processes only need to be done once before the neural network training process. The design of MPF should meet the criteria listed in the beginning of Chapter 4.

4.1.2 MPF Implementation for Atomic Charge Prediction

In this project, the MPF algorithm is used to predict several kinds of atomic charge. In this section I present the MPF implementation for atomic charge prediction. Please note that I follow the notation used in Section 4.1.1 in the following paragraphs.

Atomic Constant Properties C

Here I use one atomic constant property in project ACP, which is the Pauling-scaling electronegativity. Thus the atomic constant properties set C contains only one element:

$$C = \{\mathbf{c}_\chi\} \tag{4.6a}$$

where the Pauling electronegativity vector $\mathbf{c}_\chi$ reads

$$\mathbf{c}_\chi = [\chi_{\mathrm{H}}, \chi_{\mathrm{He}}, \ldots, \chi_{\mathrm{Uuo}}]\,. \tag{4.6b}$$

For notational convenience I use χ_i to represent the corresponding electronegativity for atom i, i.e.

$$\chi_i = \mathbf{c}_\chi[Z_i]. \tag{4.6c}$$

Here, Z_i denotes the atomic number (element) of atom i.

Initialisation Function I

In project ACP the atomic feature vector $\mathbf{h}_{t,i}$ (in any time step t) contains one feature ($d = 1$):

$$\mathbf{h}_{t,i} = \left[\mathrm{h}_{t,i}^{\mathrm{self}}\right], \tag{4.7a}$$

where i denotes the atom ID (i.e. $v_i \in V$) and the $\mathrm{h}_{t,i}^{\mathrm{self}}$ is named the self feature for atom i in time step t. The self feature is designed to store the aggregated information during update process, and represents the general environment that the atom located in. Here, self features $\mathrm{h}_{0,i}^{\mathrm{self}}$ for all atom are initialised to 1 for an equal starting point for all atoms with the initialisation function:

$$\mathrm{h}_{0,i}^{\mathrm{self}} = I'_{\mathrm{self}} = 1. \tag{4.7b}$$

Since the messages are generates as the product of self features (see Equations (4.8b) and (4.8c)), the initial value of self features cannot be 0, otherwise, the message generation function M_t cannot output a meaningful message.

Message Generation Function M

In project ACP, each atomic message vector $\mathbf{m}_{t,i}$ in molecular message vector $\mathbf{M}_t$ for any time t contains two messages (i.e. $k = 2$), $\mathrm{m}_{t,i}^{\mathrm{bond}}$ and $\mathrm{m}_{t,i}^{\mathrm{atom}}$, i.e.

$$\mathbf{m}_{t,i} = \left[\mathrm{m}_{t,i}^{\mathrm{bond}}, \mathrm{m}_{t,i}^{\mathrm{atom}}\right]. \tag{4.8a}$$

It should be noticed that, the message vector $\mathbf{m}_{t,i}$ (typeset in bold Roman), the message $\mathrm{m}_{t,i}^{\mathrm{bond}}$ and $\mathrm{m}_{t,i}^{\mathrm{atom}}$ (typeset in Roman font) and the maximum update steps m (typeset in italic) are distinct variables. Readers are advised to distinguish them carefully. $\mathrm{m}_{t,i}^{\mathrm{bond}}$ is the bond contribution message and $\mathrm{m}_{t,i}^{\mathrm{atom}}$ is the atom contribution message in time step t. Since project ACP uses the same message generation function for all time steps t, the notation for the message generation function M_t (for molecules) and M'_t (for atoms) below is simplified to M and M' respectively (not to be confused with the messages $\mathbf{M}_t$, $\mathbf{m}_{t,i}$, $\mathrm{m}_{t,i}^{\mathrm{bond}}$ and $\mathrm{m}_{t,i}^{\mathrm{atom}}$). As the message vector $\mathbf{m}_{t,i}$ contains two messages, the message generation function M' defined here consists of two parts: M'_{bond} and M'_{atom}, where they are

$$\mathrm{m}_{t+1,i}^{\mathrm{bond}} = M'_{\mathrm{bond}}(P_t, i) = \sum_j \mathrm{h}_{t,j}^{\mathrm{self}} \cdot T_{ij} \tag{4.8b}$$

and

$$\mathrm{m}_{t+1,i}^{\mathrm{atom}} = M'_{\mathrm{atom}}(P_t, i) = \sum_j \mathrm{h}_{t,j}^{\mathrm{self}} \cdot \chi_i \chi_j A_{ij}. \tag{4.8c}$$

As Equation (4.8b) shows, the bond contribution message $\mathrm{m}_{t,i}^{\mathrm{bond}}$ is generated with the bond order information (T_{ij}) and the self feature $\mathrm{h}_{t,j}^{\mathrm{self}}$ of neighbouring atoms. An atom that is not connected is skipped since $T_{ij} = 0$ if atoms v_i and v_j are not connected. In Equation (4.8c), it can be found that the atom contribution message $\mathrm{m}_{t,i}^{\mathrm{atom}}$ is built based on the Pauling electronegativity. The reasons I use electronegativity here is, first, the electronegativity is related to the atomic charge distribution (project objective); and second, the electronegativities for elements in our dataset (C, H, O, N, F, S, Cl and Br) are distinct. Thus, the electronegativity can represent the element of the atom in this project, acting as a proxy for the atomic number.

Graph Update Function U

The graph update function U_t used in project ACP aims to combine the messages in $\mathbf{m}_{t,i}$ and update the existing atom feature $\mathbf{h}_{t,i}$. Here, I use the same update function for all time steps t. Note that, as there is only one feature $\mathrm{h}_{t,i}^{\mathrm{self}}$ in the atomic feature vector $\mathbf{h}_{t,i}$, in the following paragraphs the graph update function U_t and atomic update function U'_t are simplified to U and U' respectively.

The atomic self feature update function U'_{self} has the form

$$\mathrm{h}_{t+1,i}^{\mathrm{self}} = U'_{\mathrm{self}}(P_t, \mathbf{M}_{t+1}, i) = \mathrm{h}_{t,i}^{\mathrm{self}} + \mathrm{m}_{t+1,i}^{\mathrm{bond}} + \mathrm{m}_{t+1,i}^{\mathrm{atom}}. \tag{4.9}$$

Here, the update function U'_{self} creates the new self feature $\mathrm{h}_{t+1,i}^{\mathrm{self}}$ by summing the messages and the current self feature. Since this update function may produce a result with a very large value, the extracted features are normalised before sending to the neural network. See Section 4.1.3 for further information. In total, 3 rounds of update ($m = 3$) are run to aggregate information.

Readout Function R

The final feature $\mathbf{F}$ in project ACP contains five features ($s = 5$) and the atomic final features can be extracted via the readout function as:

$$\mathbf{f}_i = R'(P_m, \mathbf{M}_m, i) = \left[\mathrm{f}_i^{\mathrm{bond}}, \mathrm{f}_i^{\mathrm{atom}}, \mathrm{f}_i^{\mathrm{self}}, \mathrm{f}_i^{\mathrm{MNE}}, \mathrm{f}_i^{\mathrm{SEN}}\right], \tag{4.10a}$$

where the bond contribution to the final feature is

$$\mathrm{f}_i^{\mathrm{bond}} = R'_{\mathrm{bond}}(\mathbf{M}_m, i) = \mathrm{m}_{m,i}^{\mathrm{bond}}, \tag{4.10b}$$

the atom contribution to the final feature is

$$\mathrm{f}_i^{\mathrm{atom}} = R'_{\mathrm{atom}}(\mathbf{M}_m, i) = \mathrm{m}_{m,i}^{\mathrm{atom}}, \tag{4.10c}$$

the self final feature is

$$\mathrm{f}_i^{\mathrm{self}} = R'_{\mathrm{self}}(P_m, i) = \mathrm{h}_{m,i}^{\mathrm{self}}, \tag{4.10d}$$

the mean neighbouring electronegativity (MNE) final feature is

$$\mathrm{f}_i^{\mathrm{MNE}} = R'_{\mathrm{MNE}}(P_m, i) = \frac{\sum_j \chi_j \cdot A_{ij}}{\sum_j A_{ij}} \tag{4.10e}$$

and the self electronegativity (SEN) final feature is

$$\mathrm{f}_i^{\mathrm{SEN}} = R'_{\mathrm{SEN}}(P_m, i) = \chi_i. \tag{4.10f}$$

The readout function generates these five features as an atomic final feature vector $\mathbf{f}_i$ for each atom in the molecule. These features are atomic ordering invariant, which mean the order of feature has no relation to the order of atoms, bonds or other objects. These atomic final feature vectors are passed to the later neural network successively for prediction.

4.1.3 Neural Network Implementation for Atomic Charge Prediction

The neural network process in this project contains three stage, the pre-processing (normalisation), fully-connected neural network prediction and the post-processing (denormalisation). In this section I present detailed information for these processes.

Pre-processing

As explained in the introduction above, the update function (Equation (4.9)) I used here can produce very large values following many update iterations. Thus, a normalisation process before passing the final feature vector to the neural network is essential. Here, I use Z-score normalisation method for pre-processing the final feature:

$$\mathrm{f'}_i^j = \frac{\mathrm{f}_i^j - \mu_j^{\mathrm{preset}}}{\sigma_j^{\mathrm{preset}}}, \quad j \in \{\mathrm{bond}, \mathrm{atom}, \mathrm{self}, \mathrm{MNE}, \mathrm{SEN}\}, \tag{4.11a}$$

Here, μ_j^{preset} and $\sigma_j^{\mathrm{preset}}$ are the average value and the standard deviation value of final feature j in the training dataset. Thus, μ_j^{preset} and $\sigma_j^{\mathrm{preset}}$ are marked as "preset" as they are calculated in the training process and are constants in the prediction process. In Equation (4.11a), $\mathrm{f'}_i^j$ is the normalised final feature. The normalised final feature notation is

$$\mathbf{F}' = \begin{bmatrix} \mathbf{f'}_1 \\ \mathbf{f'}_2 \\ \vdots \\ \mathbf{f'}_n \end{bmatrix} = \begin{bmatrix} \mathrm{f'}_1^{\mathrm{bond}} & \mathrm{f'}_1^{\mathrm{atom}} & \mathrm{f'}_1^{\mathrm{self}} & \mathrm{f'}_1^{\mathrm{MNE}} & \mathrm{f'}_1^{\mathrm{SEN}} \\ \mathrm{f'}_2^{\mathrm{bond}} & \mathrm{f'}_2^{\mathrm{atom}} & \mathrm{f'}_2^{\mathrm{self}} & \mathrm{f'}_2^{\mathrm{MNE}} & \mathrm{f'}_2^{\mathrm{SEN}} \\ & & \cdots & & \\ \mathrm{f'}_n^{\mathrm{bond}} & \mathrm{f'}_n^{\mathrm{atom}} & \mathrm{f'}_n^{\mathrm{self}} & \mathrm{f'}_n^{\mathrm{MNE}} & \mathrm{f'}_n^{\mathrm{SEN}} \end{bmatrix}. \tag{4.11b}$$

This normalised final feature vector can now be sent to the neural network for prediction.

Neural Network Design

The neural network used in project ACP is a 5×50 fully-connected neural network, i.e. five 50-neuron hidden layers in this neural network. The allowed values of the layer index l are

$$\begin{cases} l = 0, & \text{input layer,} \\ l \in \{1, 2, 3, 4, 5\}, & \text{hidden layers, and} \\ l = 6, & \text{output layer.} \end{cases} \tag{4.12a}$$

This neural network only accepts one atomic final feature vector $\mathbf{f}_i$ each time. Since each atomic final feature vector $\mathbf{f}_i$ consist of five final features ($s = 5$), there are five neurons in the input layer of this NN. In addition, the target prediction in project APC is the atomic charge. Therefore, only one output is required for each atom, i.e. one neuron in the output layers. Thus, the numbers of neurons q_l in layer l in this neural network is

$$\begin{cases} q_0 = 5, & \text{input layer,} \\ q_l = 50, l \in \{1, 2, 3, 4, 5\}, & \text{hidden layers, and} \\ q_6 = 1, & \text{output layer.} \end{cases} \tag{4.12b}$$

In this neural network, there is a total of 10,551 trainable parameters. The number of trainable parameters is constrained by the size of available training sets. In this project, the smallest training sets are the GDB-based datasets, which consist of 203,052 atoms (see Section 3.1 for more information). There is approximately one trainable parameter for every 20 available samples (parameter-sample ratio 1 : 19.24), ensuring each parameter can be trained with sufficient data.

The activation function used in this model is tanh, applying only on hidden layers. The activation function for output and input layers is linear. For more information about activation functions, please see the activation function paragraph in Section 2.3.1 and Figure 2.3. The mathematical form of the neural network can be summarised as follows

$$\begin{aligned} \mathbf{G}_i^0 &= \mathbf{f}_i' && \text{input layer,} \\ \mathbf{G}_i^l &= \tanh(\mathbf{W}^{l-1}\mathbf{G}_i^{l-1} + \mathbf{b}^{l-1}) && \text{hidden layers, } l \in \{1, 2, 3, 4, 5\}, \text{ and} \\ \mathbf{y}_i' &= \mathbf{G}_i^6 = \mathbf{W}^5\mathbf{G}_i^5 + \mathbf{b}^5 && \text{output layer.} \end{aligned} \tag{4.12c}$$

In the above equations for atom i, $\mathbf{G}_i^l$ denotes the feature in layer l, $\mathbf{W}^l \in \mathbb{R}^{q_l \times q_{l+1}}$ is the weight matrix between layers l and $l+1$, $\mathbf{b}^l$ is the bias in layer l and

$$\mathbf{y}_i' = \left[\mathrm{y}_i'\right] \tag{4.12d}$$

is the atomic predicted result vector (containing only one element y_i' in project ACP, i.e. $o = 1$) for atom i before denormalisation (which is described below). These predicted

result vectors can be combined into the normalised molecular predicted result vector

$$\mathbf{Y}' = \left[\mathbf{y}'_1, \mathbf{y}'_2, \ldots, \mathbf{y}'_n\right]. \tag{4.12e}$$

The optimiser used to train this model is the Adam optimiser (see Equation (2.74a)), with parameters $\eta = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$. In the training process every model is trained for 100 epochs with a batch size of 16 (for GDB-based training sets) or 768 (for ZINC-based training set). The loss function in training is mean absolute error (MAE).

Post-processing

A denormalisation process is required as post-processing in order to retrieve the predicted charges for atoms. This is because both the input final features of the NN and the true values in the training sets are normalised by Z-scale normalisation. According to Equation (4.11a), the denormalisation can be written as

$$\mathrm{y}_i = \sigma_{\mathrm{y}}^{\mathrm{preset}} \cdot \mathrm{y}'_i + \mu_{\mathrm{y}}^{\mathrm{preset}}, \tag{4.13a}$$

where $\sigma_{\mathrm{y}}^{\mathrm{preset}}$ and $\mu_{\mathrm{y}}^{\mathrm{preset}}$ are the standard deviation and average value obtained from the true values in the training set. In Equation (4.13a), y_i is the predicted atomic charge.

The last process in the post-processing is the rebuilding of the predicted result vector for the molecule as

$$\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n] = [\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_n]. \tag{4.13b}$$

The molecular predicted result vector $\mathbf{Y}$ consists of a atomic predicted result vectors $\mathbf{y}_i$ for all atoms. Thus, $\mathbf{Y}$ is the vector of the predicted charge values y_i in project ACP.

4.1.4 Uses of Software

In the above model, several programs are used for different purposes. These programs and thier applications are list below.

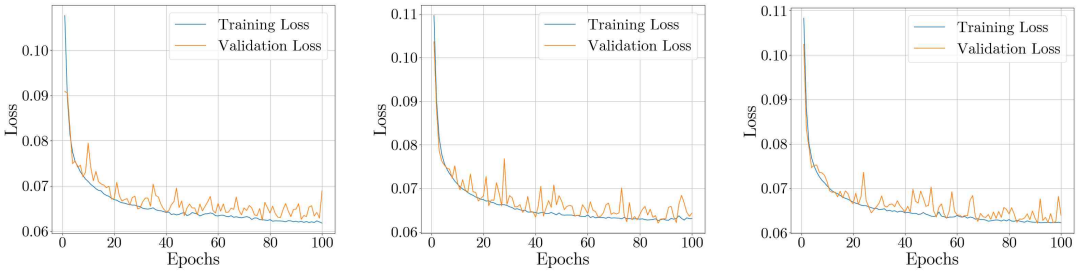
"prj_acdm" is the main program for MPF, which is written by myself. Module "predac" is the part that implements project ACP. Please see Chapter 6 for more information.

Open Babel [97] can be automatically called by "prj_acdm" for the conversion from SMILES to mol2 so that the SMILES-based molecule can be accepted by "prj_acdm".

TensorFlow [57] is used for the implementation of neural network model (Section 4.1.3).

4.2 Result and Discussion on Accuracy

In project ACP, a total of five sets of models is trained and tested. Table 3.2 presents the training sets and testing sets for each model in this project. For each training set, three independent models are trained to minimise the variability of analysis. The program used



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.2: The training loss and validation loss of model "Mulliken" during training epochs.

for training is "prj_acdm". The input files for training and testing Mulliken charge models are presented in Appendix H as examples. Here, the analysis of the training and testing results for each model is given in following sections.

4.2.1 Mulliken Charge

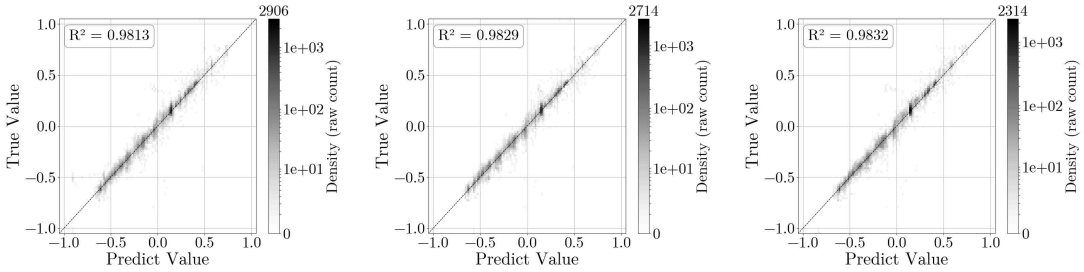
Training Analysis

For Mulliken charge, three independent models are trained based on the dataset "Mulliken_Train". Figure 4.2 gives the changes of training loss and validation loss during the training process (the loss function used is mean absolute error). From these three plots it can be observed that the decrease of training loss is reaching a plateau and the changes of validation loss have no continuous trend of increase. Thus, these models for Mulliken charge prediction are well trained, without serious over-fitting problems. The next stage of the analysis can be carried out.

Testing on "Mulliken_Test"

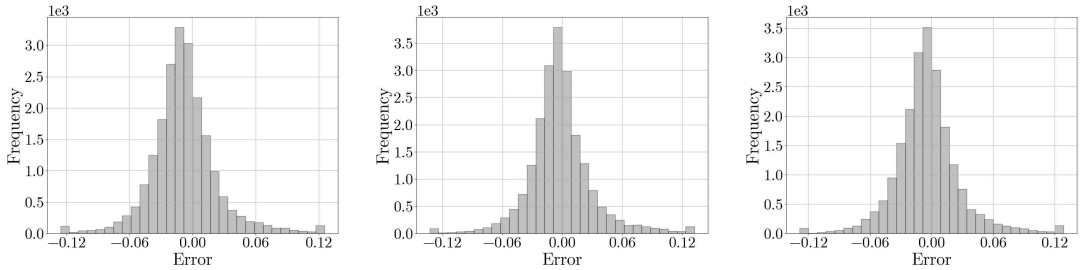
To evaluate the performance of trained models, the first testing set applied is dataset "Mulliken_Test". This testing set originated from the GDB-13 database, which is the same as the training set. The size of the molecules in this dataset is similar to the molecules in the training set. The aim of the test on this dataset "Mulliken_Test" is to provide a preliminarily evaluation of the accuracy of the models. In this section a general visualised accuracy evaluation is provided, followed by a numerical analysis of the performance of the model for each element.

The heatmaps provided in Figure 4.3 shows the distribution of the predicted results relative to the true values from DFT. For better visualization, the values at each sampling point are plotted on a logarithmic scale (log10). As Figure 4.3 shows, most data points (i.e. the predicted charge for each atom in the dataset) lie close to the diagonal dashed line $y = x$, where the data being closer to this line means better accuracy. The coefficient of determination for each model is also provided in Figure 4.3. By analysing the heatmaps and the R-squared score (> 0.98) of three models, the correlation between the model predictions and the true values can be clearly demonstrated.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.3: Heatmaps analysis for Mulliken charge models tested on dataset "Mulliken_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values is the elementary charge e.



(a) Model_1 (b) Model_2 (c) Model_3

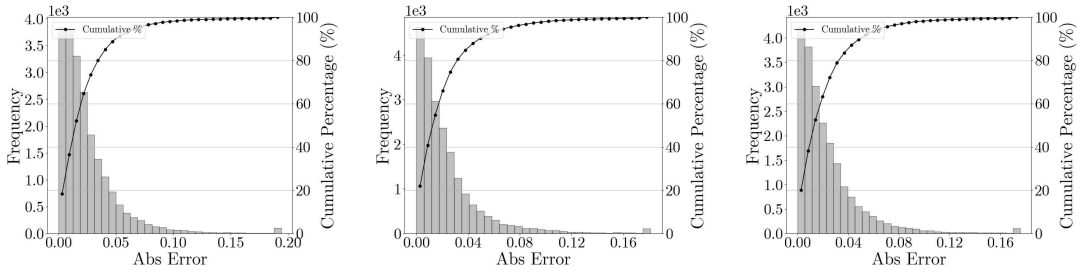
Figure 4.4: Error histograms for Mulliken charge models tested on dataset "Mulliken_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e.

In Figure 4.4, the histograms of the error between predicted value and true value for these three Mulliken charge models are provided. In these histograms overflow bins are used for both negative and positive errors, containing the most extreme 0.5% of the errors. From these histograms it can be found that the errors locate in a Gaussian-like distribution, with a slight offset (about $-0.01e$) of the peak toward negative values. This may cause a general shift of the predicted charge for atoms. By comparing to the values of the Mulliken charges (mostly located between $\pm 0.7e$, observed from Figure 4.3), this shift is one order of magnitude smaller than the data values.

The histograms for the distribution of absolute errors of the predicted result are presented in Figure 4.5. These absolute error histograms and cumulative percentage curves can give a clearer presentation of the models' accuracy distribution. From the histograms in Figure 4.5 it can be observed that 80% of the predicted results have less than $0.04e$ absolute error compared to the true value from DFT.

Apart from the visualised evaluation, a numerical analysis is also given. Here, I use seven metric to evaluate the performance of the model for each element. They are

- the mean absolute error (MAE),
- the median absolute error (MedAE),



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.5: Absolute error histograms for Mulliken charge models tested on dataset "Mulliken_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on absolute error.

- the top low 80% absolute error (TL80AE, introduced later),
- the median absolute percentage error (MedAPE),
- the top low 80% percentage error (TL80PE, introduced later),
- the root mean squared error (RMSE) and
- the coefficient of determination (R^2).

It is noticeable that, the definition of the percentage error of the result used in above metrics (including MedAPE & TL80PE) is

$$\text{Percentage Error} = \frac{\text{Predicted Value} - \text{True Value}}{\text{True Value}}. \tag{4.14}$$

The "TL80AE" and "TL80PE" mentioned above refer to the trimmed absolute error and trimmed absolute percentage error computed from the 80% of data points with the lowest individual errors. After sorting all absolute errors (or percentage errors) in ascending order, TL80AE and TL80PE are the maximum values within the lowest 80% subset. These two metrics can reflect the typical prediction performance of models, and provide a more robust measure of accuracy.

In Table 4.1, the value of all seven metrics for three Mulliken charge model on all elements are provided. The table shows that these three MPF-based models can achieve an overall MedAPE of 7.66% on average, with a TL80PE of less than 20% on average. This shows that these MPF-based models can predict Mulliken charge with reasonable accuracy. By comparing MedAPE and TL80PE element-wise, we see that these models predict more accurate Mulliken charges on nitrogen (2.52% MedAPE and 6.42% TL80PE) and oxygen atoms (4.75% MedAPE and 11.54% TL80PE), which are much better than for carbon atoms (13.58% MedAPE and 42.08% TL80PE). By checking the molecules and atoms with high error, I found that the surrounding and bonding environment of carbon atoms can be more complex than other elements. In particular the small ring structures (3-atom or 4-atom carbon rings) can often cause great errors. The second possible reason is that

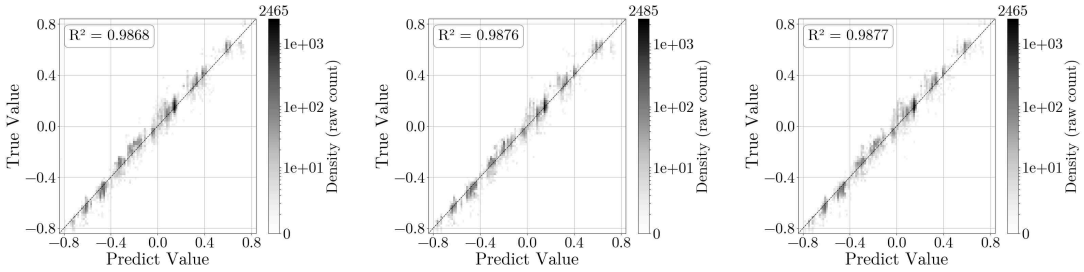
Table 4.1: The value of metrics on testing for three Mulliken charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error.

Metric	Model	H	C	N	O	Overall
MAE (e)	Average	0.0174	0.0355	0.0274	0.0185	0.0245
	Model_1	0.0201	0.0361	0.0279	0.0161	0.0257
	Model_2	0.0155	0.0349	0.0267	0.0224	0.0237
	Model_3	0.0167	0.0356	0.0275	0.0170	0.0240
MedAE (e)	Average	0.0138	0.0237	0.0179	0.0123	0.0163
	Model_1	0.0167	0.0242	0.0173	0.0110	0.0179
	Model_2	0.0117	0.0220	0.0190	0.0143	0.0153
	Model_3	0.0130	0.0247	0.0174	0.0117	0.0159
TL80AE (e)	Average	0.0277	0.0533	0.0381	0.0290	0.0357
	Model_1	0.0313	0.0529	0.0383	0.0245	0.0376
	Model_2	0.0252	0.0528	0.0391	0.0352	0.0344
	Model_3	0.0266	0.0541	0.0369	0.0271	0.0352
MedAPE	Average	7.85%	13.58%	4.75%	2.52%	7.66%
	Model_1	9.67%	13.70%	4.53%	2.25%	8.52%
	Model_2	6.67%	12.70%	5.07%	2.98%	7.09%
	Model_3	7.21%	14.34%	4.66%	2.32%	7.37%
TL80PE	Average	16.74%	42.08%	11.54%	6.43%	19.93%
	Model_1	18.73%	41.69%	11.35%	5.47%	20.75%
	Model_2	15.39%	42.92%	11.99%	7.85%	19.33%
	Model_3	16.11%	41.63%	11.29%	5.96%	19.69%
RMSE (e)	Average	0.0226	0.0576	0.0491	0.0261	0.0406
	Model_1	0.0254	0.0596	0.0523	0.0224	0.0424
	Model_2	0.0207	0.0580	0.0409	0.0323	0.0397
	Model_3	0.0218	0.0553	0.0542	0.0237	0.0397
R^2	Average	0.9333	0.9574	0.9209	0.9090	0.9820
	Model_1	0.9165	0.9544	0.9115	0.9351	0.9804
	Model_2	0.9449	0.9569	0.9460	0.8646	0.9828
	Model_3	0.9385	0.9609	0.9051	0.9274	0.9828

the distribution of the value of charge on carbon atoms is much wider than that of other elements (see Appendix I.1). This adds more complexity to the charge prediction on carbon atoms.

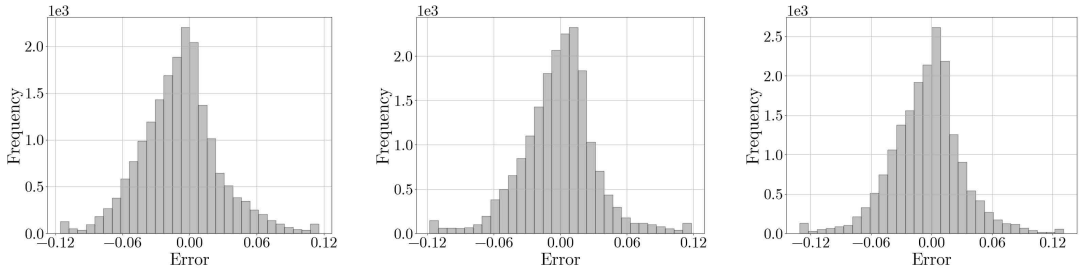
Testing on "LM_Mulliken_Test"

The model2 trained as described in Section 4.2.1 is also tested on dataset "LM_Mulliken_Test" to evaluate its performance on larger molecules. The molecules in dataset "LM_Mulliken_Test" are much larger (50-100 heavy atoms) than those in the training set of these models (dataset "Mulliken_Train" has 4-8 heavy atoms). Figure 4.6 presents the heatmaps comparing the predicted and true values. Here, the data points again align along the diagonal direction indicating that the model can still perform well in predicting charge for larger molecules.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.6: Heatmaps analysis for Mulliken charge models tested on dataset "LM_Mulliken_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e.



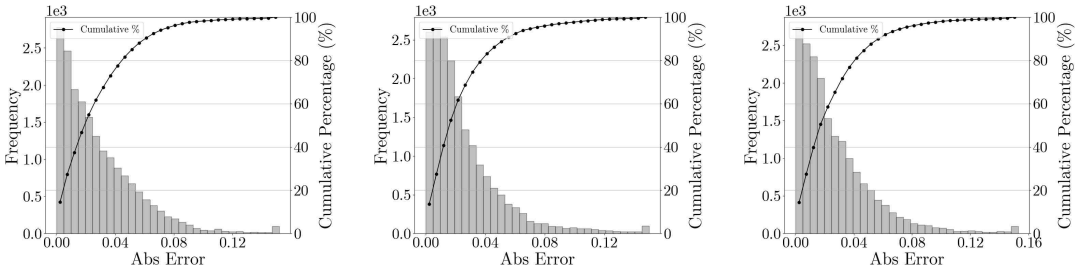
(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.7: Error histograms for Mulliken charge models tested on dataset "LM_Mulliken_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme of the data. The unit of the error is the elementary charge e.

The error histograms for each models are presented in Figure 4.7. Compared to Figure 4.4, three main differences can be observed. First, the peak of the error distribution locates close to zero, shifting slightly in a positive direction (especially Model_2). The second difference is that, the error distribution in larger molecule test displays an asymmetric shape, with a more gradual decline on the negative side than the positive side. This indicates that the models tend to predict value slightly smaller than the DFT true value in dataset "LM_Mulliken_Test". The third difference here is the broader distribution of the errors in the larger molecule test. In conclusion, these models trained on small molecules suffer loss in accuracy when making predictions for larger molecules. Quantitative analysis is presented in later paragraphs.

In the histograms of absolute error (presented in Figure 4.8), the increase of the absolute error is noticeable. By comparing to Figure 4.5, a more gentle decline can be observed in the larger molecule test results due to the loss in accuracy. In addition, less than 80% of the data has an error lower than $0.04e$, indicating the predicted result for "LM_Mullikne_Test" is less accurate.

Therefore, a numerical analysis is needed for quantitative evaluation of the loss in accuracy. Table 4.2 gives the value of metrics in the accuracy analysis. Compared to Table 4.1, most



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.8: Absolute error histograms for Mulliken charge models tested on dataset "LM_Mulliken_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on the absolute error.

of the metrics for the errors from the larger molecule test increase. Noticeably the average errors for oxygen of the three models significantly increases to 4.97% in MedAPE and 8.82% in TL80PE, while the values of these metrics for small molecules are 2.52% and 6.43% respectively. The errors slightly increase for carbon and hydrogen atoms. The reason for the decrease in accuracy for larger molecules might be the missing long-range interactions or complex conformational effect in the training dataset. Since the molecules in the training set have a maximum of 8 heavy atoms, the long-range interactions have an effect that is simple for the model to include. Further, the update round in the model is set to 3, i.e. the NN input features for each atom can contain information from atoms up to third nearest neighbour in the topology. Thus, when predicting charges for larger molecules with 50-100 heavy atoms, the model cannot respond correctly to complex longer ranged interactions.

However, the accuracy of nitrogen atoms in larger molecule tests is better than that in the GDB-based "Mulliken_Test" dataset, with an average TL80PE of 7.50% in the "LM_Mulliken_Test" and 11.54% in the "Mulliken_Test". This unexpected result might be caused by the differences of dataset. By comparing Figures 3.1(b) and 3.4, it is obvious that the number (as well as the proportion) of nitrogen atoms in "LM_Mulliken_Test" is significantly lower than that in "Mulliken_Test". Thus, "LM_Mulliken_Test" has less diverse nitrogen atomic environments. This might be the reason why the overall accuracy of predicted charges on nitrogen atoms in the larger molecule tests is better.

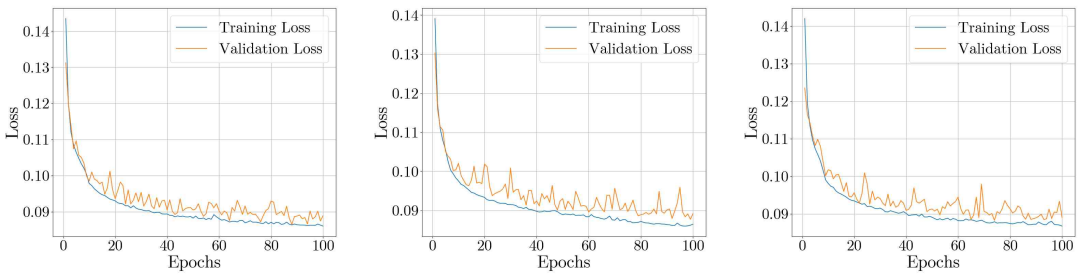
4.2.2 Hirshfeld Charge

Training Analysis

Similar to the process in Mulliken charge models, three models are trained to evaluate the performance of this MPF-based algorithm for Hirshfeld charge prediction. Figure 4.9 gives the results of the training process including training loss and validation loss. These curves show that these trained models are close to convergence and have no obvious over-fitting problem. Thus, these three models are proved eligible for later tests.

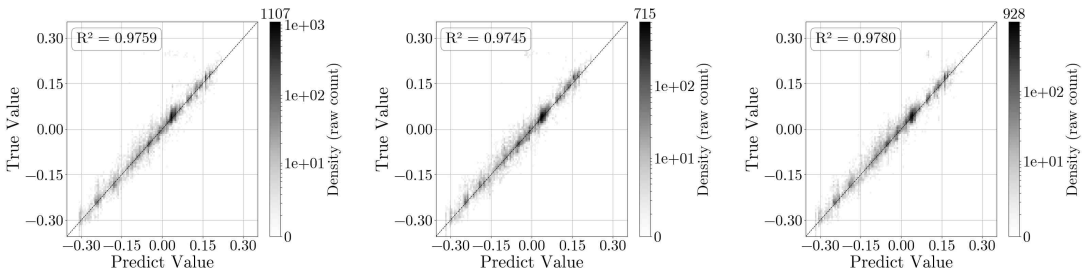
Table 4.2: The value of metrics on larger molecule testing for three Mulliken charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error.

Metric	Model	H	C	N	O	Overall
MAE (e)	Average	0.0187	0.0381	0.0320	0.0310	0.0273
	Model_1	0.0199	0.0389	0.0354	0.0349	0.0287
	Model_2	0.0178	0.0368	0.0307	0.0269	0.0259
	Model_3	0.0184	0.0385	0.0298	0.0310	0.0272
MedAE (e)	Average	0.0145	0.0301	0.0212	0.0273	0.0197
	Model_1	0.0148	0.0314	0.0239	0.0319	0.0214
	Model_2	0.0144	0.0277	0.0200	0.0218	0.0183
	Model_3	0.0143	0.0312	0.0197	0.0281	0.0195
TL80AE	Average	0.0304	0.0593	0.0463	0.0483	0.0432
	Model_1	0.0337	0.0615	0.0514	0.0543	0.0470
	Model_2	0.0278	0.0569	0.0434	0.0418	0.0397
	Model_3	0.0296	0.0597	0.0440	0.0488	0.0429
MedAPE	Average	8.61%	14.70%	3.47%	4.97%	8.76%
	Model_1	8.70%	15.98%	3.84%	5.81%	9.21%
	Model_2	8.68%	13.69%	3.26%	4.03%	8.41%
	Model_3	8.46%	14.43%	3.31%	5.06%	8.68%
TL80PE	Average	17.30%	43.40%	7.50%	8.82%	21.22%
	Model_1	18.54%	43.33%	8.36%	9.99%	22.56%
	Model_2	16.63%	42.02%	7.28%	7.64%	20.18%
	Model_3	16.74%	44.83%	6.86%	8.83%	20.91%
RMSE (e)	Average	0.0246	0.0505	0.0480	0.0385	0.0380
	Model_1	0.0266	0.0510	0.0511	0.0421	0.0395
	Model_2	0.0230	0.0496	0.0469	0.0356	0.0367
	Model_3	0.0242	0.0509	0.0461	0.0378	0.0379
R^2	Average	0.9138	0.9731	0.8193	0.6937	0.9860
	Model_1	0.8998	0.9726	0.7960	0.6348	0.9849
	Model_2	0.9251	0.9741	0.8281	0.7398	0.9870
	Model_3	0.9166	0.9728	0.8338	0.7065	0.9861



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.9: The training loss and validation loss of model "Hirshfeld" during training epochs.



(a) Model_1 (b) Model_2 (c) Model_3

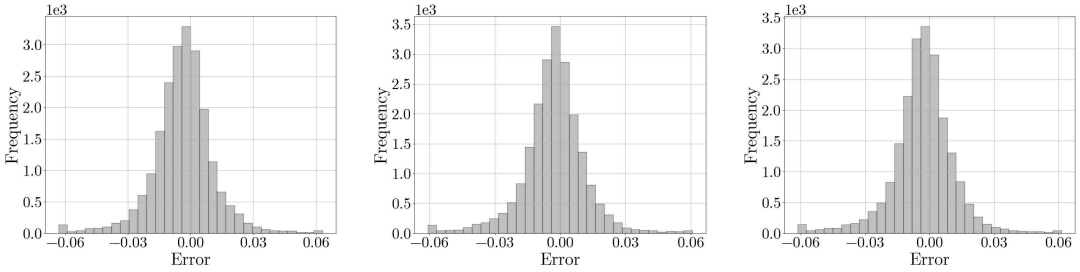
Figure 4.10: Heatmaps analysis for Hirshfeld charge models tested on dataset "Hirshfeld_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e.

Testing on "Hirshfeld_Test"

The first test run on Hirshfeld is based on dataset "Hirshfeld_Test". The heatmaps for this test are given in Figure 4.10, with x- and y-axes representing the predicted and true values respectively. Generally speaking, the data points are aligned along the diagonal direction, indicating a strong correlation between the predicted value and true value. However, there are two primary differences that can be found when comparing to the heatmaps for the Mulliken charge test (Figure 4.3). First, the scale of the Hirshfeld charge is significantly lower than that for the Mulliken charge (see tick labels on axis of heatmaps). Also, the R-squared values for the three models are slightly lower ($R^2 \approx 0.975$ in Hirshfeld charge prediction, compared to $R^2 \approx 0.982$ in Mulliken charge prediction), showing that the prediction accuracy for the Hirshfeld charges is lower. Further investigation is presented in the following paragraphs.

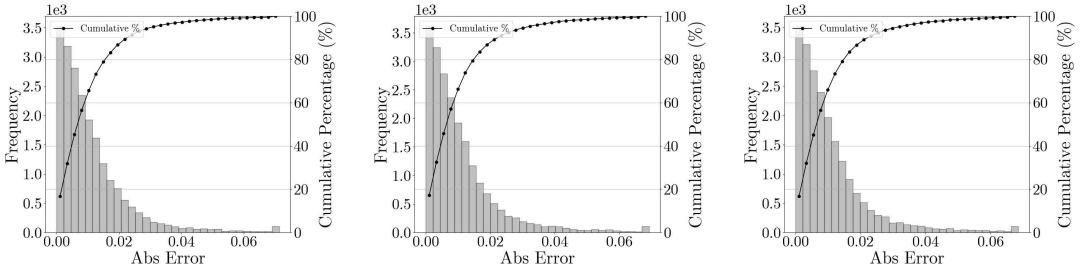
The error histograms for Hirshfeld charge models are presented in Figure 4.11. Here, similar distributional characteristics can be observed by comparing to the histograms for Mulliken charge Figure 4.4. Figure 4.11 shows that the distribution peak shifts mildly toward the negative direction. The average distance of the shift is less than $0.01e$. Figure 4.12 shows the histogram of the absolute error for Hirshfeld charge prediction. Here, 80% of the absolute error is less than $0.02e$. Although the scales of the errors are smaller than those for the test of Mulliken charge models, the absolute percentage errors of the Hirshfeld charges are expected to be larger, as the Hirshfeld charges are significantly smaller than the corresponding Mulliken charges. A quantitative comparison is given in the following paragraph.

Here, I follow the same metrics used in evaluating the Mulliken charge models. Table 4.3 gives the result of quantitative evaluation for three Hirshfeld charge models. The accuracy for each element also follows the regularity observed in the test on Mulliken charge models shown in Table 4.1, i.e. the prediction for oxygen atoms has the best accuracy while the prediction for carbon atoms has the worst accuracy. In addition, it can be observed that the metrics value of absolute errors (MAE, MedAE and TL80AE) are significantly smaller compared to the error metrics for Mulliken charge provided in Table 4.1. However, the



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.11: Error histograms for Hirshfeld charge models tested on dataset "Hirshfeld_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e.



(a) Model_1 (b) Model_2 (c) Model_3

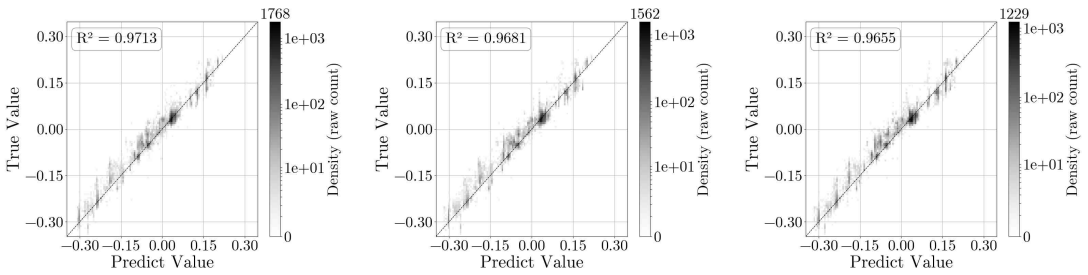
Figure 4.12: Absolute error histograms for Hirshfeld charge models tested on dataset "Hirshfeld_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on absolute error.

Table 4.3: The value of metrics from testing the three Hirshfeld charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error.

Metric	Model	H	C	N	O	Overall
MAE (e)	Average	0.0085	0.0116	0.0169	0.0144	0.0109
	Model_1	0.0091	0.0116	0.0165	0.0141	0.0112
	Model_2	0.0083	0.0113	0.0177	0.0145	0.0109
	Model_3	0.0080	0.0118	0.0165	0.0147	0.0108
MedAE (e)	Average	0.0070	0.0074	0.0105	0.0099	0.0076
	Model_1	0.0077	0.0071	0.0100	0.0099	0.0079
	Model_2	0.0068	0.0072	0.0107	0.0098	0.0075
	Model_3	0.0066	0.0078	0.0110	0.0099	0.0075
TL80AE (e)	Average	0.0133	0.0178	0.0245	0.0230	0.0161
	Model_1	0.0143	0.0178	0.0243	0.0227	0.0167
	Model_2	0.0130	0.0179	0.0251	0.0229	0.0159
	Model_3	0.0127	0.0177	0.0242	0.0236	0.0159
MedAPE	Average	12.99%	18.22%	8.94%	4.66%	12.20%
	Model_1	14.77%	18.35%	8.29%	4.66%	12.94%
	Model_2	12.24%	17.42%	9.05%	4.68%	11.80%
	Model_3	11.96%	18.90%	9.47%	4.65%	11.87%
TL80PE	Average	26.52%	64.93%	27.59%	11.04%	31.40%
	Model_1	28.61%	64.61%	26.65%	10.81%	32.39%
	Model_2	25.14%	64.57%	27.74%	10.99%	30.35%
	Model_3	25.82%	65.62%	28.37%	11.33%	31.47%
RMSE (e)	Average	0.0109	0.0174	0.0296	0.0202	0.0169
	Model_1	0.0115	0.0177	0.0294	0.0195	0.0171
	Model_2	0.0107	0.0170	0.0332	0.0204	0.0174
	Model_3	0.0103	0.0176	0.0263	0.0206	0.0163
R^2	Average	0.9378	0.9390	0.8556	0.8710	0.9750
	Model_1	0.9301	0.9373	0.8591	0.8798	0.9746
	Model_2	0.9395	0.9421	0.8203	0.8683	0.9736
	Model_3	0.9438	0.9377	0.8874	0.8650	0.9769

percentage error metrics (MedAPE & TL80PE) for Hirshfeld charge are greatly larger than those for Mulliken charge, with an average TL80PE value of 64.93% in carbon atoms.

One of the reasons of such an anomalous phenomenon is that the scale of Hirshfeld charge is relatively smaller than the scale of Mulliken charge (see Appendix I.2). An inspection of those results with extremely high percentage errors shows that most of these cases are due to the small scale of the true values (see definition in Equation (4.14)), ranging from $10^{-6}e$ to $10^{-3}e$. Thus, although the absolute error of is smaller, the relative error as a proportion of the Hirshfeld charge can be significantly large. Thus, this result shows that the MPF-based models can still be suitable for Hirshfeld charge prediction.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.13: Heatmaps analysis for Hirshfeld charge models tested on dataset "LM_Hirshfeld_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e.

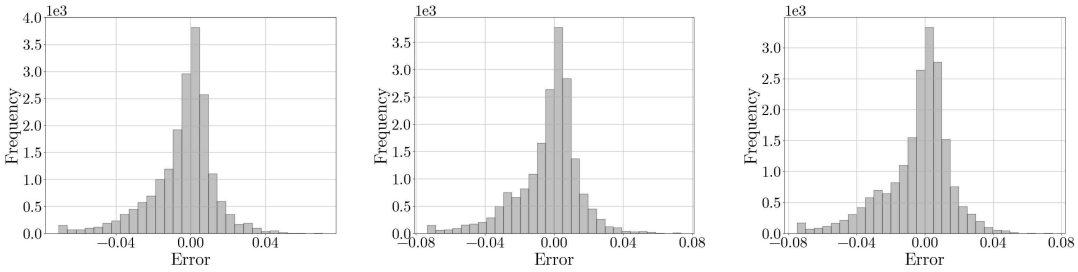
Testing on "LM_Hirshfeld_Test"

The Hirshfeld charge models are also tested on the dataset "LM_Hirshfeld_Test" to evaluate the performance of the models when predicting charge for larger molecules. The heatmaps, error histograms and absolute error histograms of the predicted results are listed in Figure 4.13, Figure 4.14 and Figure 4.15 respectively.

From the heatmaps in Figure 4.13, although the distribution still follows the diagonal direction, there is more dispersion compared to the results tested on the GDB-based small molecule dataset (Figure 4.10). In addition, the R-squared values on the larger molecule dataset for all three Hirshfeld charge models is lower, indicating that the correlation between the predicted and true value is weakened.

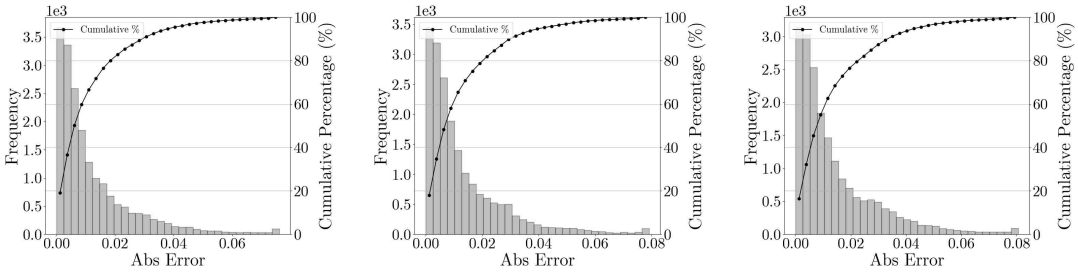
From the error histograms given in Figure 4.14, a similar phenomenon to that mentioned in LM test on Mulliken charge models can be also found. That is, the slope of the error distribution on the negative side is smaller than that in the positive direction, indicating that the model tends to predict a smaller charge in many cases. This can be also proved by inspecting the heatmaps in Figure 4.13, where more data points are located on the top-left side of the diagonal dashed line. In the absolute error distribution presented in Figure 4.15, 80% of the absolute errors are lower than $0.03e$. This corresponds to a great loss in accuracy by percentage. Further investigation is carried out in the quantitative analysis.

The evaluation metrics in Table 4.4 reveal several negative indicators regarding the accuracy of MPF-based models in predicting Hirshfeld charges. The overall Med reaches 15.54%, meaning that 50% of predictions have percentage errors less than 15.53%. Particularly, the absolute percentage error in Hirshfeld charge prediction for carbon atoms is unexpectedly high, with an average TL80PE of 89.82%. Thus the raw data with high percentage error are inspected manually. The inspection shows that the scale of the DFT true value of these carbon atoms is relatively small, with a range from 10^{-6} to 10^{-3} (see Appendix I.2). This shows a bad robustness of MPF-based models when predicting Hirshfeld charge for carbon atoms which are too close to neutral.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.14: Error histograms for Hirshfeld charge models tested on dataset "LM_Hirshfeld_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e.



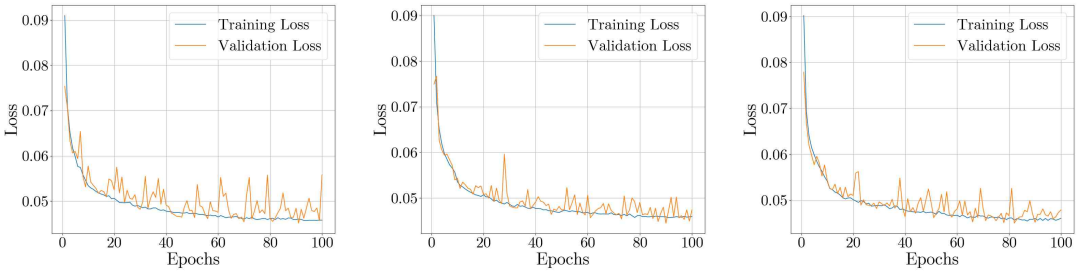
(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.15: Absolute error histograms for Hirshfeld charge models tested on dataset "LM_Hirshfeld_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data according to the absolute error.

Table 4.4: The value of metrics on larger molecule testing for three Hirshfeld charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error.

Metric	Model	H	C	N	O	Overall
MAE (e)	Average	0.0084	0.0164	0.0216	0.0231	0.0132
	Model_1	0.0077	0.0150	0.0203	0.0224	0.0123
	Model_2	0.0083	0.0160	0.0233	0.0229	0.0131
	Model_3	0.0092	0.0181	0.0212	0.0241	0.0143
MedAE (e)	Average	0.0061	0.0103	0.0179	0.0176	0.0081
	Model_1	0.0055	0.0093	0.0163	0.0168	0.0073
	Model_2	0.0061	0.0099	0.0183	0.0175	0.0080
	Model_3	0.0069	0.0116	0.0190	0.0186	0.0090
TL80AE (e)	Average	0.0130	0.0294	0.0316	0.0365	0.0216
	Model_1	0.0118	0.0273	0.0295	0.0352	0.0196
	Model_2	0.0129	0.0283	0.0343	0.0356	0.0214
	Model_3	0.0143	0.0327	0.0311	0.0388	0.0239
MedAPE	Average	15.53%	21.05%	15.82%	8.13%	15.54%
	Model_1	14.01%	19.86%	14.12%	7.81%	14.13%
	Model_2	15.17%	19.53%	17.86%	8.15%	15.22%
	Model_3	17.43%	23.75%	15.46%	8.42%	17.26%
TL80PE	Average	33.72%	89.82%	28.81%	21.02%	42.38%
	Model_1	29.62%	82.68%	27.09%	19.57%	36.89%
	Model_2	34.21%	90.99%	28.37%	20.35%	43.22%
	Model_3	37.31%	95.80%	30.95%	23.14%	47.04%
RMSE (e)	Average	0.0113	0.0230	0.0281	0.0307	0.0195
	Model_1	0.0106	0.0214	0.0268	0.0299	0.0184
	Model_2	0.0112	0.0226	0.0302	0.0303	0.0193
	Model_3	0.0123	0.0251	0.0273	0.0318	0.0208
R^2	Average	0.9058	0.9117	0.8558	0.7092	0.9616
	Model_1	0.9184	0.9243	0.8690	0.7234	0.9659
	Model_2	0.9086	0.9156	0.8340	0.7159	0.9624
	Model_3	0.8905	0.8953	0.8642	0.6883	0.9565

However, an unexpected phenomenon is found in Table 4.4. Although the MedAPE and TL80PE for Hirshfeld charge prediction for carbon atoms are significantly worse than those of oxygen atoms, the R-squared value of oxygen atoms is unexpectedly low at 0.70. One possible explanation for this discrepancy is the difference in the distribution of charge value in carbon and oxygen. An inspection of the DFT true values for dataset "LM_Hirshfeld_Test" is carried out, indicating that the Hirshfeld charge for carbon ranged from $-0.1583e$ to $0.2546e$, while oxygen atom has a more narrow range from $-0.3473e$ to $-0.0701e$.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.16: The training loss and validation loss of model "CM5" during training epochs.

4.2.3 CM5 Charge

Training Analysis

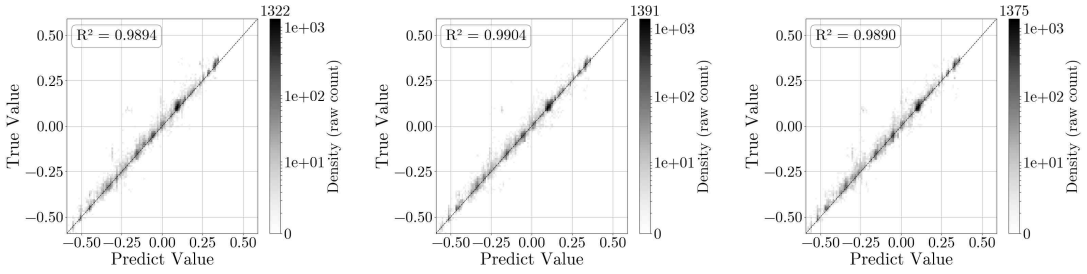
The training of CM5 charge models follows the same settings as the models for Mulliken and Hirshfeld charges. Figure 4.16 shows the trend of training and validation loss during the training process. It can be observed that the training and validation losses for "Model_2" and "Model_3" continuously decrease and approach convergence. However, an abnormal increase of the validation loss in "Model_1" is observed at the end of the training. Since the validation loss in "Model_1" continuously decreases, this abnormal rise of the validation loss might be a consequence of the noise in the training. This can be proved by comparing the curve for "Model_1" to "Model_2" and "Model_3", where only the validation loss in "Model_1" exhibits substantial noise and large fluctuations during training. It is likely a result of random initialization or batch variance, rather than a systematic issue. Despite its instability, "Model_1" is still included in later tests for evaluation on the robustness of MPF algorithm.

Testing on "CM5_Test"

The first test on CM5 charge models is based on the GDB-based dataset "CM5_Test". Figure 4.17 gives the heatmaps of these three models. It can be observed that the distribution of data points fits well the diagonal direction, showing that the overall accuracy of MPF-based models in CM5 prediction is satisfactory. In addition, the R-squared values of these three models all approach 0.99, indicating a strong correlation between the predicted values and the true values.

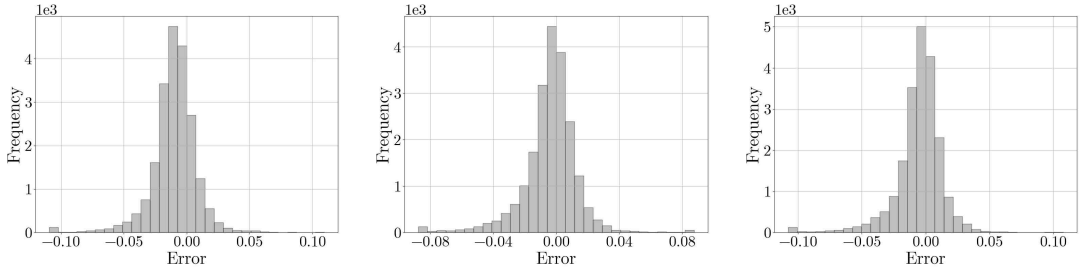
The error histograms are presented in Figure 4.18. These histograms show that all three models exhibit symmetrical distributions centred around zero, except for "Model_1" whose peak is slightly shifted towards negative values ($\sim -0.01e$). In the absolute error histograms given in Figure 4.19, 80% of the data has an error less than $0.02e$. However, the distribution of absolute errors of "Model_1" is slightly broader than for "Model_2" and "Model_3", with a slower decline of the distribution towards larger errors.

Table 4.5 presents the value of evaluation metrics for all three models for the individual elements. From Figure 4.16 we see "Model_1" exhibits some instability during training.



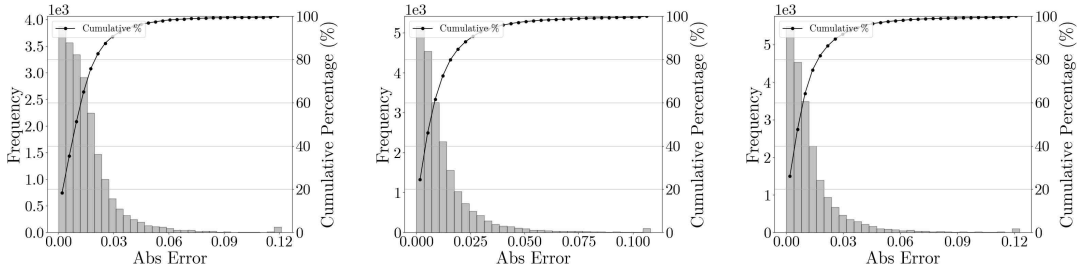
(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.17: Heatmaps analysis for CM5 charge models tested on dataset "CM5_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.18: Error histograms for CM5 charge models tested on dataset "CM5_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.19: Absolute error histograms for CM5 charge models tested on dataset "CM5_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on absolute error.

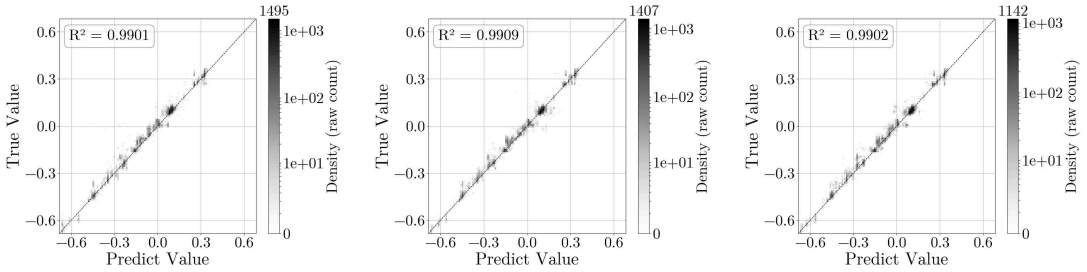
Table 4.5: The value of metrics from testing for three CM5 charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error.

Metric	Model	H	C	N	O	Overall
MAE (e)	Average	0.0103	0.0151	0.0200	0.0186	0.0137
	Model_1	0.0139	0.0158	0.0189	0.0175	0.0154
	Model_2	0.0086	0.0140	0.0194	0.0180	0.0124
	Model_3	0.0085	0.0153	0.0217	0.0204	0.0133
MedAE (e)	Average	0.0086	0.0094	0.0108	0.0113	0.0092
	Model_1	0.0125	0.0104	0.0101	0.0100	0.0114
	Model_2	0.0065	0.0084	0.0102	0.0113	0.0077
	Model_3	0.0067	0.0095	0.0122	0.0126	0.0084
TL80AE (e)	Average	0.0159	0.0222	0.0281	0.0258	0.0193
	Model_1	0.0208	0.0223	0.0253	0.0236	0.0217
	Model_2	0.0135	0.0211	0.0279	0.0250	0.0176
	Model_3	0.0133	0.0232	0.0310	0.0288	0.0187
MedAPE	Average	6.51%	11.28%	3.16%	3.46%	6.56%
	Model_1	9.46%	12.03%	2.93%	3.15%	8.21%
	Model_2	5.08%	9.94%	2.98%	3.47%	5.55%
	Model_3	4.97%	11.86%	3.57%	3.77%	5.92%
TL80PE	Average	13.42%	41.36%	9.17%	9.42%	16.82%
	Model_1	17.15%	46.65%	8.54%	8.84%	19.34%
	Model_2	11.60%	37.27%	8.95%	9.16%	15.14%
	Model_3	11.52%	40.18%	10.03%	10.27%	15.97%
RMSE (e)	Average	0.0132	0.0240	0.0395	0.0298	0.0229
	Model_1	0.0169	0.0251	0.0384	0.0301	0.0242
	Model_2	0.0114	0.0226	0.0381	0.0274	0.0213
	Model_3	0.0113	0.0243	0.0420	0.0320	0.0232
R^2	Average	0.9778	0.9629	0.8851	0.8978	0.9882
	Model_1	0.9649	0.9595	0.8914	0.8965	0.9869
	Model_2	0.9840	0.9673	0.8933	0.9143	0.9898
	Model_3	0.9845	0.9620	0.8705	0.8826	0.9879

This is reflected in the significantly larger absolute errors and absolute percentage errors: for example the overall MedAE for "Model_1" is $0.0114e$ while those for "Model_2" and "Model_3" are $0.0077e$ and $0.0084e$. Apart from that, the characteristics of the metrics of the CM5 model tested in dataset "CM5_Test" is similar to those of the Mulliken charge models (by comparing Appendices I.1 and I.3). To avoid repetition, readers are advised to check the description given for Mulliken charge in Section 4.2.1.

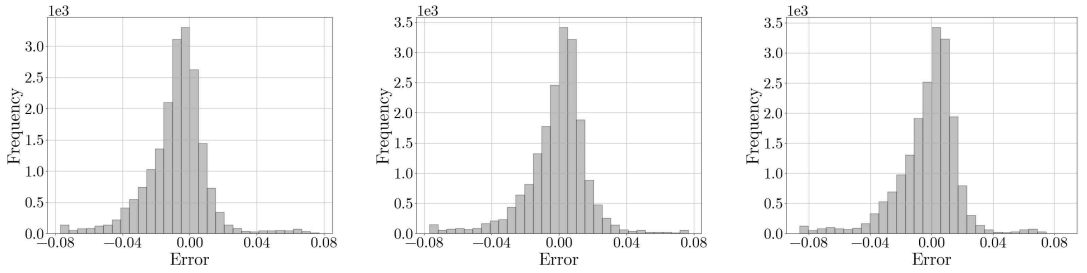
Testing on "LM_CM5_Test"

The CM5 models are also tested on the larger molecule dataset "LM_CM5_Test". Figure 4.20 presents the heapmaps of the predicted results against the true values. These three heatmaps indicate that all three models perform well in the larger molecule test since most



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.20: Heatmaps analysis for CM5 charge models tested on dataset "LM_CM5_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e.



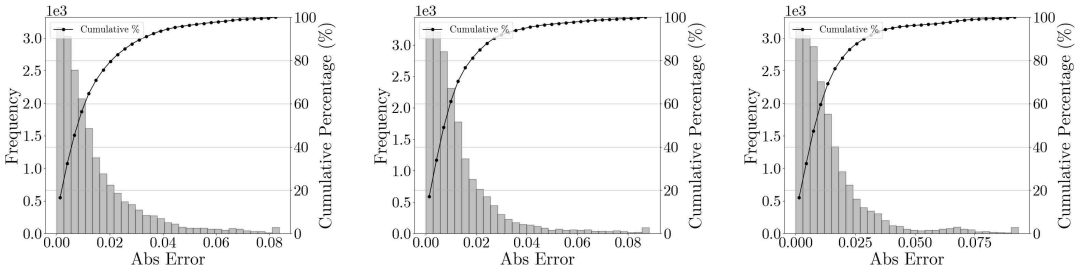
(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.21: Error histograms for CM5 charge models tested on dataset "LM_CM5_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e.

data points lie close to the diagonal direction. In addition, the R-squared values for the LM tests on the three models are larger than 0.99, indicating strong correlation between the predicted values and true values.

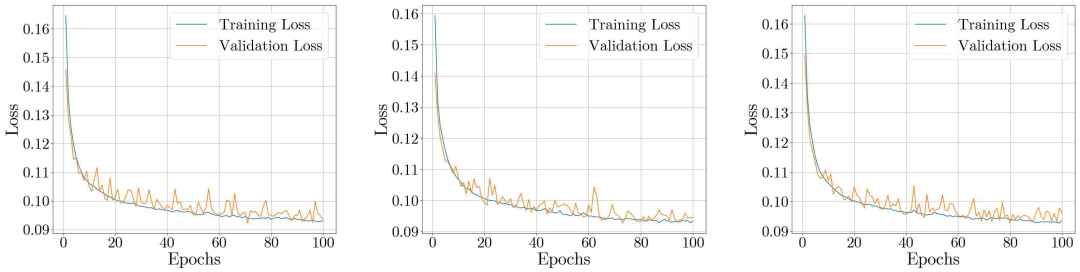
Figure 4.21 gives the error histograms for CM5 charge models. Here, the asymmetric error distribution between positive and negative values can also be observed, and is similar to the pattern found in Mulliken and Hirshfeld charge models tested on the larger molecule dataset. Figure 4.22 shows the distribution of absolute error in histograms and cumulative percentage curves. From these diagrams it can be found that about 80% of the absolute error locates within $0.025e$. Additionally, the great error of "Model_1" mentioned in the previous GDB-based test cannot be observe obviously in the absolute error in histograms in larger molecule tests. The reason for such behaviour might be the differences between datasets "CM5_Test" and "LM_CM5_Test", where dataset "CM5_Test" contains a larger variety of structures (such as 3-atom and 4-atom rings, which are rare in the ChEMBLE-based 'LM_CM5_Test" datasets).

From the evaluation metrics given in Table 4.6, the overall performance of the MPF-based models in predicting CM5 charge is clearly reflected. The overall MedAPE for all elements in CM5 prediction is only 7.15%, while the value of TL80PE is 20.29%. This indicates that the MPF-based models can have good accuracy for CM5 charge prediction. However,



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.22: Absolute error histograms for CM5 charge models tested on dataset "LM_CM5_Test". The unit of absolute error (x-axis) is elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data in absolute error.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.23: The training loss and validation loss of model "DDEC" during training epochs.

it can be also observed that for carbon atoms, although the MedAPE value is relatively low (13.92%, lower than that of Mulliken charge models), the average TL80PE value for carbon atoms is still as high as 80.22%. After an inspection of the raw data the reason for this phenomenon is similar to that for the Hirshfeld charge model, i.e. the DFT true values of these high percentage data points are relatively small in scale (from 10^{-6} to 10^{-3}).

4.2.4 DDEC Charge

Training Analysis

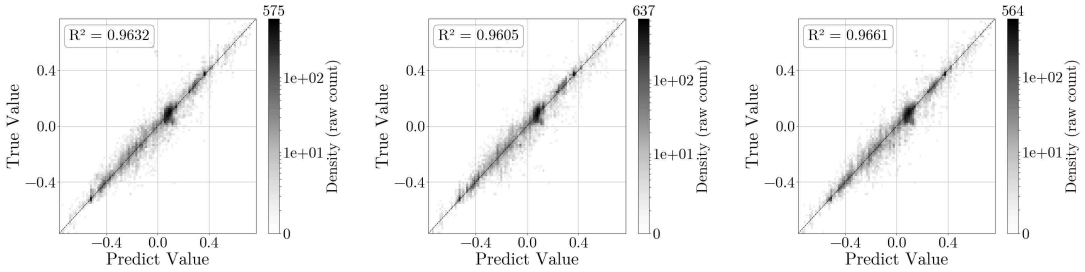
The training and validation loss of each epoch during the training process are plotted in Figure 4.23. The curves in these three plots show that the models are well-trained and have no obvious over-fitting behaviours. Therefore, these three trained model are valid for later tests.

Testing on "DDEC_Test"

The heatmaps of of test result are provided in Figure 4.24. The first characteristic observed is a broader distribution of the data points, similar to the distribution of Hirshfeld models. In addition, the R-squared values in the test on DDEC6 models range between 0.96 and 0.97, indicating the correlations between the predicted and true values are weaker, compared to Mulliken and CM5 charge models.

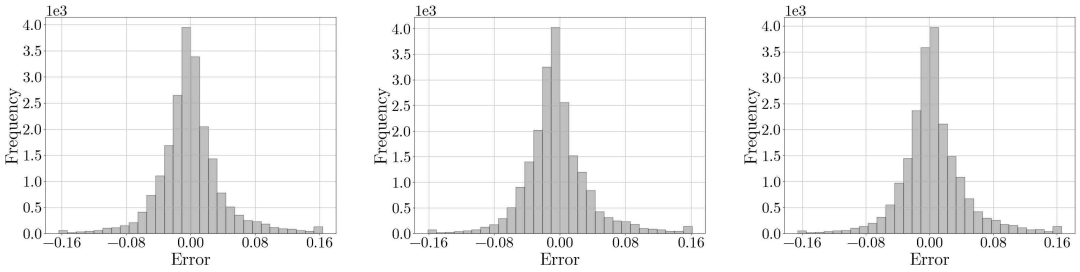
Table 4.6: The value of metrics for larger molecule testing for three CM5 charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error.

Metric	Model	H	C	N	O	Overall
MAE (e)	Average	0.0093	0.0180	0.0171	0.0207	0.0138
	Model_1	0.0094	0.0190	0.0195	0.0196	0.0142
	Model_2	0.0090	0.0169	0.0144	0.0205	0.0132
	Model_3	0.0094	0.0180	0.0174	0.0221	0.0141
MedAE (e)	Average	0.0076	0.0115	0.0122	0.0134	0.0092
	Model_1	0.0074	0.0124	0.0133	0.0123	0.0092
	Model_2	0.0074	0.0107	0.0101	0.0128	0.0088
	Model_3	0.0081	0.0114	0.0131	0.0151	0.0096
TL80AE (e)	Average	0.0146	0.0295	0.0278	0.0298	0.0205
	Model_1	0.0149	0.0324	0.0359	0.0274	0.0220
	Model_2	0.0142	0.0267	0.0206	0.0303	0.0192
	Model_3	0.0148	0.0293	0.0268	0.0317	0.0204
MedAPE	Average	6.88%	13.92%	2.61%	4.08%	7.15%
	Model_1	6.74%	14.91%	2.93%	3.72%	6.94%
	Model_2	6.65%	13.43%	2.14%	3.82%	6.99%
	Model_3	7.25%	13.42%	2.77%	4.69%	7.51%
TL80PE	Average	13.80%	80.22%	5.84%	10.46%	20.29%
	Model_1	13.80%	92.59%	6.94%	9.59%	20.63%
	Model_2	13.03%	60.15%	4.75%	10.49%	19.54%
	Model_3	14.58%	87.92%	5.83%	11.28%	20.70%
RMSE (e)	Average	0.0118	0.0265	0.0242	0.0302	0.0209
	Model_1	0.0122	0.0282	0.0271	0.0289	0.0217
	Model_2	0.0114	0.0250	0.0210	0.0302	0.0199
	Model_3	0.0117	0.0264	0.0246	0.0314	0.0210
R^2	Average	0.9791	0.9629	0.9536	0.8733	0.9893
	Model_1	0.9775	0.9584	0.9428	0.8838	0.9884
	Model_2	0.9804	0.9670	0.9654	0.8735	0.9902
	Model_3	0.9794	0.9634	0.9527	0.8626	0.9892



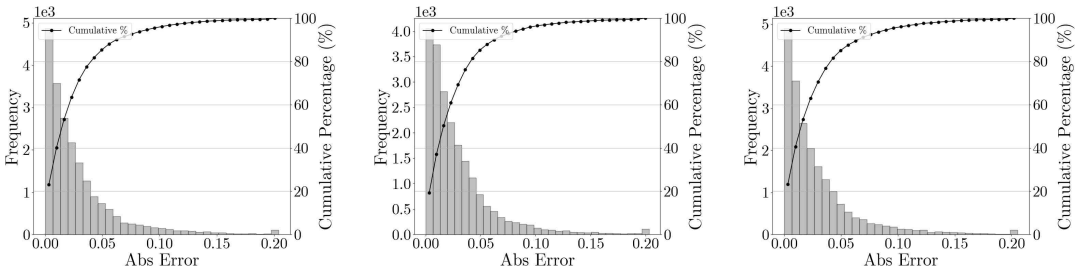
(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.24: Heatmaps analysis for DDEC6 charge models tested on dataset "DDEC_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.25: Error histograms for DDEC6 charge models tested on dataset "DDEC_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.26: Absolute error histograms for DDEC6 charge models tested on dataset "DDEC_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on absolute error.

The error histograms and the absolute error histograms of the test on DDEC6 models are given in Figure 4.25 and Figure 4.26, respectively. From Figure 4.25, it can be observed that symmetrical distributions of error centre at the zero point, with a relatively wide distribution towards both positive and negative directions. Figure 4.26 shows that, about 80% of the absolute error are less than $0.05e$, which is relatively large in proportion to the range of DDEC6 charge values (see Appendix I.4 for the range of true values). Further quantitative analysis can provide more information.

The evaluation metrics for DDEC6 models in this test are listed in Table 4.7. From the absolute-error metrics (MAE, MedAE, TL80AE & RMSE) in the table, it can be observed that the value of absolute error are relatively larger than those in previous models for other charges. More specifically, the overall TL80AE value for DDEC6 charge is $0.0437e$, while the TL80AE values are $0.0357e$ for Mulliken charge models, $0.0161e$ for Hirshfeld charge models and $0.0193e$ for CM5 charge. For the proportional-error metrics, the DDEC6 models have similar performance when compared to the Hirshfeld charge, with an overall TL80PE value of 43.57% for DDEC6 charge models and 43.35% for Hirshfeld charge models.

For element-wise comparison, the patterns indicated by these metrics are similar to previous

Table 4.7: The value of metrics for tests for three DDEC6 charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error.

Metric	Model	H	C	N	O	Overall
MAE (e)	Average	0.0213	0.0424	0.0338	0.0197	0.0293
	Model_1	0.0204	0.0417	0.0347	0.0209	0.0289
	Model_2	0.0224	0.0419	0.0361	0.0197	0.0299
	Model_3	0.0210	0.0436	0.0305	0.0185	0.0291
MedAE (e)	Average	0.0155	0.0286	0.0186	0.0113	0.0185
	Model_1	0.0144	0.0283	0.0183	0.0118	0.0180
	Model_2	0.0171	0.0283	0.0185	0.0113	0.0194
	Model_3	0.0148	0.0291	0.0190	0.0108	0.0180
TL80AE (e)	Average	0.0343	0.0671	0.0453	0.0291	0.0437
	Model_1	0.0324	0.0645	0.0463	0.0311	0.0432
	Model_2	0.0362	0.0675	0.0466	0.0295	0.0440
	Model_3	0.0343	0.0694	0.0430	0.0266	0.0439
MedAPE	Average	14.70%	22.39%	6.61%	2.81%	13.09%
	Model_1	13.79%	21.51%	6.37%	3.02%	12.82%
	Model_2	16.21%	22.52%	6.78%	2.76%	13.85%
	Model_3	14.11%	23.14%	6.68%	2.65%	12.60%
TL80PE	Average	38.81%	79.02%	20.98%	8.48%	43.57%
	Model_1	38.21%	78.92%	22.58%	9.16%	42.13%
	Model_2	38.71%	77.45%	21.28%	8.61%	43.40%
	Model_3	39.52%	80.70%	19.07%	7.66%	45.17%
RMSE (e)	Average	0.0293	0.0610	0.0692	0.0316	0.0472
	Model_1	0.0290	0.0600	0.0730	0.0331	0.0472
	Model_2	0.0294	0.0602	0.0823	0.0321	0.0489
	Model_3	0.0296	0.0629	0.0525	0.0296	0.0456
R2	Average	0.9168	0.9261	0.8517	0.9180	0.9627
	Model_1	0.9187	0.9287	0.8405	0.9100	0.9628
	Model_2	0.9166	0.9281	0.7971	0.9156	0.9601
	Model_3	0.9150	0.9215	0.9176	0.9283	0.9653

models tested on the GDB-based datasets. According to the test results for DDEC6, MPF-based models perform well on predictions for oxygen and nitrogen atoms, but perform less accurately on carbon atoms.

Since DDEC6 charges are widely adopted as target values for machine learning models, a model-wise comparison to existing models on accuracy can be carried out. Table 4.8 summarises the accuracy of MPF-based model alongside the Bleiziffer's random forest model [42] and the DeepAtomicCharge's GNN model [10].

It should be noticed that the data in Table 4.8 are obtained from the respective publications. The operating environment (solvent vs. vacuum), the training set and the testing set for each model are different. Thus, it is impossible to carry out a fair quantitative comparison. Nonetheless, the table provides a qualitative reference for evaluating the relative

Table 4.8: Accuracy comparison with existing DDEC6-based charge models. The present work is evaluated in vacuum ($\varepsilon = 1$), whereas Bleiziffer's model and DeepAtomicCharge were tested in specific solvent environments.

Model	Testing Set	Environment	RMSE
MPF-based (this work)	GDB-based	$\varepsilon = 1$	0.0472
Bleiziffer's Model [42]	ZINC-based	$\varepsilon = 4$	0.029
	ChEMBL-based		0.0162
DeepAtomicCharge [10]	Mix of ZINC and ChEMBL	$\varepsilon = 4$	0.0107
		$\varepsilon = 78$	0.0129

performance of different models under their respective conditions.

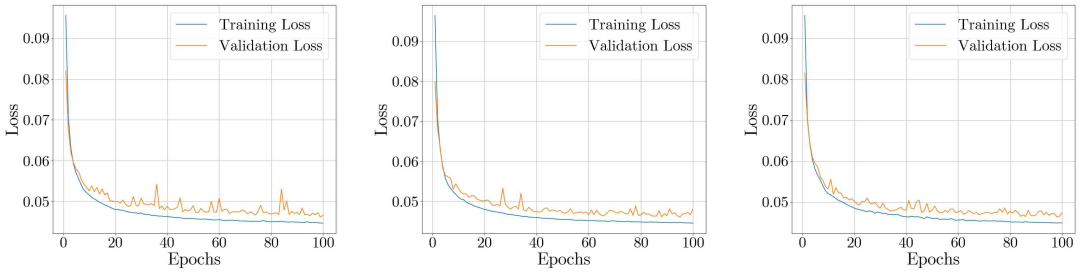
As shown in Table 4.8, the MPF-based model (this work) has the highest RMSE error in the specific testing set and environment. However, it is notable that the solvent environment may tend to smooth the electrostatic interaction and may lead to lower fitting errors for those models. Under vacuum conditions the MPF-based model suffers from stronger electrostatic fluctuations. The current result can illustrate the robustness of our model to some extent. Moreover, this result highlights a potential for the MPF-based model to achieve competitive accuracy in the same environment. Furthermore, the MPF-based model can accept molecules of arbitrary size regardless of the size of molecule present in the training set. This versatility gives the MPF-based model an advantage over other models that may require fixed-size inputs.

4.2.5 SE Charge

The last test charge for MPF algorithm is the semi-empirical charge (abbreviated as SE) generated by AMSOL (see Section 3.3). The training and testing datasets used for this charge model is based on the ZINC20 database which contains 8 elements (C, H, O, N, F, S, Cl and Br). In addition, the ZINC20 based training and testing datasets are more than 100 times larger than the previously used GDB-based training and testing sets. Here, I would like to use this dataset set to figure out the performance of the MPF-based algorithm for larger datasets with more elements.

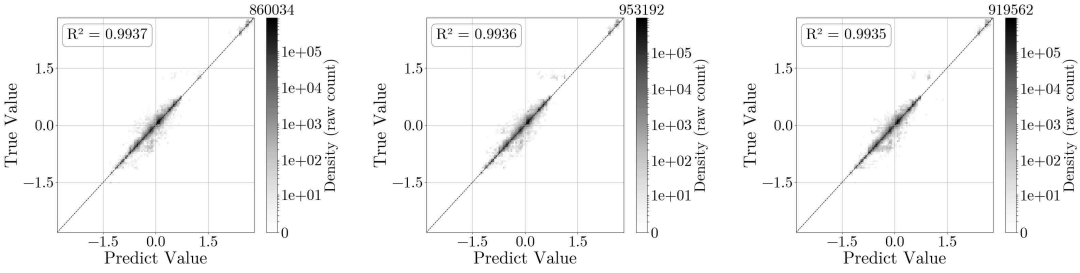
Training Analysis

The batch size is set to 768 when training SE models, since the training dataset for SE charge is significantly larger than the ones for other charges. Other parameters remain the same for consistency. The curves of losses during the training process for SE charge models are presented in Figure 4.27. Form these plots, both the training and validation losses exhibit a downward trend, with no clear signs of over-fitting. Thus, these three models are suitable for later testing.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.27: The training loss and validation loss of model "SE" during training epochs.



(a) Model_1 (b) Model_2 (c) Model_3

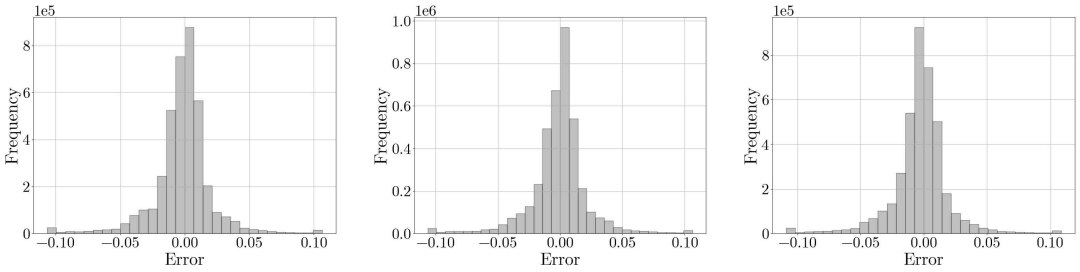
Figure 4.28: Heatmaps analysis for semi-empirical charge models tested on dataset "SE_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are the elementary charge e.

Testing on "SE_Charge_Test"

The heatmaps of the predicted value against true value are provided in Figure 4.28. The data points in these heatmaps are well aligned along the diagonal direction indicating satisfactory overall accuracy. In addition, the R-squared value for the three models all exceed 0.99, showing a strong correlation between the predicted and true values.

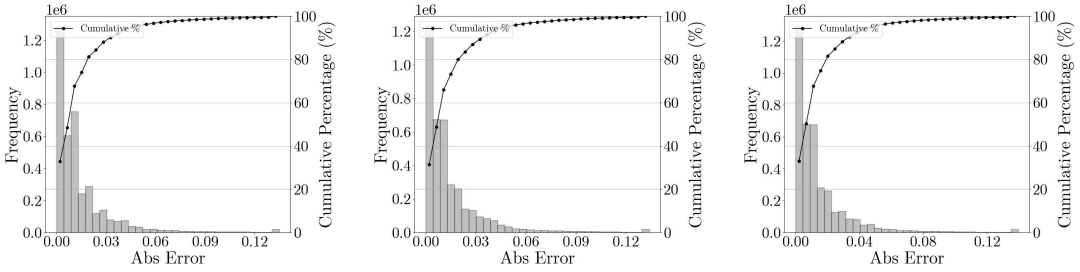
The histograms of the predicted error distribution of three models are presented in Figure 4.29. These histograms show that the errors are symmetrically distributed, centring precisely around zero. The absolute error histograms provided in Figure 4.30 show that about 80% of errors are less than $0.02e$, indicating desirable accuracy when predicting SE charges. However, in "Model_1" an increasing frequency can be spotted around $0.01e$ absolute error, while flat planes can be found in "Model_2" and "Model_3" around $0.01e$. Although these absolute errors are relatively small compared to the true value (most data locate within $-1.5e \sim 1.0e$, observed from Figure 4.28), this phenomenon can reflect a potential weakness for MPF-based models in predicting specific structures.

The evaluation metrics for each element are listed in Table 4.9 for further investigation. In these SE charge models, a total of eight elements is included in both training and testing datasets. It is found that the SE charge prediction for N, O, F and S atoms are satisfactorily accurate, with the values of TL80PE being lower than 10%. In addition, the accuracy of hydrogen and carbon atoms are similar to those for Mulliken and CM5 charges.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.29: Error histograms for semi-empirical charge models tested on dataset "SE_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error is the elementary charge e.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 4.30: Absolute error histograms for semi-empirical charge models tested on dataset "SE_Test". The unit of absolute error (x-axis) is the elementary charge e. The unit of frequency is provided on the top of the y-axis. The overflow box contains the extreme 0.5% of data based on the absolute error.

However, the predictions for Cl and Br atoms are unreliable in this test: the TL80PE values for Cl and Br prediction are around 300% and 1000% respectively. Further, the average R-squared values for these two elements are 0.41 and -19.57, indicating that the models cannot predict charge for Cl and Br atoms. One reason for this result might be the limited number of these atoms in the training dataset. From Figure 3.6(a) it can be seen that the proportions of Cl and Br atoms in the training set are only 0.014% and 0.00097%. Therefore, the models may not have enough information about the properties of these elements. Additionally, the value scale of the SE charge of Cl and Br atoms are relatively small (see Appendix I.5), this can also increase the predict error in Cl and Br atoms.

4.3 Result and Discussion on Time Efficiency

Apart from the accuracy of prediction, the time efficiency of the MPF-based models is also important. In this section, the costs of training and predicting are presented on a typical personal desktop computer. All training and testing processes mentioned in this section are carried out on the same machine. The machine specifications are listed below:

System: Apple Mac mini (2024)

CPU: Apple M4, 10-core (4 performance + 6 efficiency)

GPU: Integrated 10-core GPU

NPU: 16-core Neural Engine

Unified Memory: 16GB (120 GB/s bandwidth)

OS: macOS Sequoia (Version 15.4.1)

Compiler: Apple clang 17.0.0

4.3.1 Training Time Efficiency

The time costs of training for all models used in Section 4.2 are listed in Table 4.10. For the GBD-based training set (models for Mulliken, Hirshfeld, CM5 & DDEC6 charge) with 203,052 atoms (data), the average cost of time is around 350 seconds (less than 6 minutes). For the larger ZINC-based training set (models for semi-empirical charge), the MPF-based models can be trained within 31 minutes. This result shows that the cost of training process is affordable for a customer-level machine.

4.3.2 Predicting Time Efficiency

The cost of prediction for the MPF-based models is also important. In this project two prediction scenarios are tested. The first scenario is the MPF-based model prediction for a single large molecule, simulating predicting charge for a large protein or polymer molecule.

Table 4.9: The value of metrics from testing three semi-empirical charge models on all elements. TL80AE is short for the top low 80% absolute error and TL80PE is short for the top low 80% percentage error.

Metric	Model	H	C	N	O	F	S	Cl	Br	Overall
MAE (e)	Average	0.0121	0.0204	0.0160	0.0109	0.0079	0.0358	0.0205	0.1143	0.0151
	Model_1	0.0120	0.0199	0.0155	0.0109	0.0074	0.0314	0.0192	0.1110	0.0148
	Model_2	0.0119	0.0209	0.0169	0.0111	0.0086	0.0364	0.0212	0.1597	0.0152
	Model_3	0.0123	0.0203	0.0156	0.0107	0.0077	0.0395	0.0211	0.0722	0.0151
MedAE (e)	Average	0.0088	0.0103	0.0084	0.0070	0.0061	0.0115	0.0146	0.0345	0.0090
	Model_1	0.0092	0.0099	0.0082	0.0070	0.0056	0.0114	0.0129	0.0314	0.0091
	Model_2	0.0086	0.0109	0.0082	0.0069	0.0072	0.0103	0.0168	0.0372	0.0090
	Model_3	0.0087	0.0101	0.0089	0.0072	0.0054	0.0129	0.0140	0.0350	0.0089
TL80AE (e)	Average	0.0191	0.0290	0.0218	0.0158	0.0126	0.0436	0.0304	0.1605	0.0212
	Model_1	0.0195	0.0282	0.0207	0.0159	0.0114	0.0403	0.0264	0.1155	0.0207
	Model_2	0.0186	0.0303	0.0236	0.0160	0.0141	0.0475	0.0323	0.3080	0.0217
	Model_3	0.0192	0.0287	0.0212	0.0156	0.0123	0.0432	0.0325	0.0579	0.0213
MedAPE	Average	9.07%	7.82%	1.48%	1.34%	3.56%	0.61%	77.24%	93.37%	5.88%
	Model_1	9.32%	7.49%	1.40%	1.32%	3.47%	0.65%	69.26%	82.99%	5.91%
	Model_2	8.69%	8.07%	1.59%	1.32%	4.18%	0.52%	80.37%	107.42%	5.89%
	Model_3	9.21%	7.91%	1.44%	1.38%	3.03%	0.66%	82.08%	89.69%	5.84%
TL80PE	Average	20.32%	27.25%	4.85%	3.31%	7.74%	4.30%	297.56%	1028.38%	19.40%
	Model_1	20.13%	26.18%	4.66%	3.34%	7.30%	3.97%	237.35%	577.52%	19.28%
	Model_2	20.13%	28.89%	5.10%	3.37%	8.09%	4.30%	391.03%	2105.87%	19.53%
	Model_3	20.69%	26.69%	4.77%	3.22%	7.82%	4.65%	264.28%	401.74%	19.38%
RMSE (e)	Average	0.0183	0.0388	0.0290	0.0206	0.0114	0.0796	0.0323	0.2109	0.0281
	Model_1	0.0183	0.0386	0.0291	0.0200	0.0106	0.0694	0.0362	0.2204	0.0279
	Model_2	0.0180	0.0385	0.0302	0.0209	0.0119	0.0830	0.0310	0.2889	0.0281
	Model_3	0.0186	0.0392	0.0276	0.0210	0.0117	0.0863	0.0297	0.1234	0.0284
R^2	Average	0.9672	0.9757	0.9819	0.9863	0.8562	0.9947	0.4108	-19.5737	0.9936
	Model_1	0.9672	0.9759	0.9817	0.9871	0.8763	0.9960	0.2661	-19.3515	0.9937
	Model_2	0.9685	0.9761	0.9803	0.9859	0.8437	0.9943	0.4618	-33.9911	0.9936
	Model_3	0.9660	0.9752	0.9836	0.9858	0.8486	0.9938	0.5044	-5.3783	0.9935

Table 4.10: The training time consumption for each charge and model. Mulliken charge, Hirshfeld charge, CM5 charge and DDEC6 charge models are trained with batch size of 16. SE charge models are trained with batch size of 768. All trainings run for 100 epochs. Due to the current implementation of program "prj_acdm", the cost of time can only be measured with a precision of seconds.

Charge Type	**Number of Atoms in Training Set**	**Model_1 (s)**	**Model_2 (s)**	**Model_3 (s)**	**Average (s)**
Mulliken	203,052	351	349	350	350
Hirshfeld		348	347	347	348
CM5		347	351	348	349
DDEC6		351	350	350	350
SE	35,532,841	1,858	1,861	1,852	1,857

The second scenario is making a prediction on a large set of data, representing the case when using an MPF-based model to filter a database for ideal molecules.

Prediction Time for a Single Large Molecule

Since the MPF-based models predict data atom by atom with no relation with the atom type or conformation, here I generate predictions for several alkanes for this single large molecule test. The number of carbon atoms in these alkane molecules range from 1 to 20,000, while the total number of atoms ranges from 5 to 60,002. All operations here are limited to single thread (multi-threading is disable).

Table 4.11 gives the time of an MPF-based model (here, I use the Mulliken charge model "Model_1") used to make predictions for different sizes of normal alkane molecules, including the time used in each step of the model. The read-in time denotes the time that the program ("prj_acdm") needs to read-in these molecules. The process stage refers to the MPF-algorithm operations. The predict stage denotes the cost of time of neural network prediction.

By carrying out numerical analysis and polynomial fitting, it can be found that the time complexity of the read-in stage and the process stage are $O(n^2)$ (where n denotes the number of atoms), while the time complexity of the neural network predict stage is linear ($O(n)$). Therefore, the overall process has a time complexity of $O(n^2)$. The fitted equations are provided below (n as number of atoms in the molecules and t is the cost of time in seconds). The read-in stage ($R^2 = 1$):

$$t_{\text{read-in}} = 5 \times 10^{-8} n^2 - 9 \times 10^{-5} n + 0.2008. \tag{4.15a}$$

The process stage ($R^2 = 1$):

$$t_{\text{process}} = 6 \times 10^{-7} n^2 - 3 \times 10^{-5} n + 2.2952. \tag{4.15b}$$

Table 4.11: The time consumption of Mulliken charge model "Model_1" tested on different size of normal alkane. The predicting batch size is 1. Due to the current implementation of program "prj_acdm", the cost of time can only be measured with a precision of seconds.

C Atom Number	H Atom Number	Total Atom Number	Read-in Time (s)	Process Time (s)	Predict Time (s)	Total Time (s)
1	4	5	<1	3	<1	3
500	1,002	1,502	<1	3	<1	3
1,000	2,002	3,002	1	7	1	9
5,000	10,002	15,002	10	127	4	141
10,000	20,002	30,002	44	498	8	550
20,000	40,002	60,002	180	1,987	17	2,184

The neural network predict stage ($R^2 = 0.9999$):

$$t_{\text{predict}} = 3 \times 10^{-4} n + 0.0032. \tag{4.15c}$$

The total cost of time ($R^2 = 1$):

$$t_{\text{total}} = 6 \times 10^{-7} n^2 + 0.0002 n + 2.5463. \tag{4.15d}$$

The Python script for this test is provided in Appendix J. The above analysis is run on a single thread for single molecules, thus giving the maximum possible cost of time. From Equation (4.15d), it can be observed that both of the second order and the first order terms have a small prefactor value of 6×10^{-7} and 2×10^{-4} respectively. Therefore, the effect of these terms can be ignored in small molecules (< 1000 atoms, which covers most cases in practice) and the cost of time can be regarded as constant (2.5463 seconds for each molecule in each thread). In addition, for real calculations the input data can consist of many molecules and multithreading can then be enable. Thus, the time consumption in practice is expected to be significantly lower. See below.

Predicting Time for Large Datasets

An example prediction corresponding to real practice is carried out in this section to evaluate the time cost. In this test multithreading is enabled and the maximum number of threads is set to 10. In addition, this prediction test uses the Mulliken model "Model_1" on a relatively large dataset "SE_Train", which consists of 492,162 molecules (19,515,294 atoms, according to Section 3.3). The prediction batch size is set to 768. The "prj_acdm" input file for this test is provided in Appendix K.

From the above fitted equation of the cost of time (Equation (4.15d)), the expected time can be calculated as 7.24 years, which is not useful. However, with the smaller molecules and the enablement of multithreading, the total time cost is only 460 seconds, as listed in Table 4.12. Therefore, the above tests shows that the MPF-based model are highly time efficient in both training and predicting.

Table 4.12: The time consumption of the Mulliken charge model "Model_1" predicting charge on dataset "SE_Train". The predicting batch size is 768. Due to the current implementation of program "prj_acdm", the cost of time can only be measured with a precision of seconds.

Molecule Number	Total Atom Number	Read-in Time (s)	Process Time (s)	Predict Time (s)	Total Time (s)
492,162	19,515,294	196	251	13	460

4.4 Summary of Project ACP

In this chapter I introduced a new algorithm, namely the message-passing featuriser (MPF), for generating features from molecular topology. The fundamental methodology ensures MPF can generate ML-ready features for each atom only based on the molecular topology. In addition, the fundamental design enables the MPF to generate features for molecules of arbitrary size, regardless of the limitations of the training dataset.

Furthermore, the tests of the MPF-based models on various charge models reflects the robustness of MPF algorithm. From the above results we see that the MPF-based models can predict various kinds of atomic charges for many elements, often with with good accuracy. However, as discussed in Section 3.1.3 (see "Open Babel Conversion and MD Optimisation"), we observe modest conformational dependence in the predicted charges (as accuracy decreases slightly when testing in larger molecule datasets). This suggests that atomic charge properties are primarily governed by short-range interactions, with minor influence from long-range effects.

Also, the time cost for training and prediction with MPF-based models is affordable, and the time efficiency of the MPF algorithm is proved to be $O(n^2)$. Notably, MPF makes it possible and affordable to predict atomic charge for extremely large molecules (with more than 10^4 atoms) or for large datasets (over 10^5 molecules) on a personal computer within a short time.

Overall, the MPF algorithm fulfils the objectives outlined in this chapter, and these results show that MPF algorithm has great potential for further applications. To investigate its broader applicability, additional studies are carried out, see Chapter 5.

Chapter 5

Project: Molecular Structure Prediction

Based on the previous introduction to the MPF algorithm (see Chapter 4), I expect to find further applications of the MPF algorithm. According to Section 4.1.1, the MPF directly operates on the graph of the molecule, where the atoms are regarded as nodes and bonds are regarded as edges. However, due to the limited information presented directly by the single molecule graph (topology matrix $\mathbf{T}$ defined in Equation (4.1e)), the messages can only be passed through bonds and saved on atoms. This greatly limits the application of the MPF algorithm.

To explore the potential of MPF, a new algorithm is required for obtaining more information from the original molecule graph. In this section, the graph latent connectivity enhancement (GLaCE, pronounced as the French word "glace", /glas/) algorithm through graph restructuring is proposed. The GLaCE algorithm is an original idea and is especially designed for molecular applications and works by transforming the molecular graph into different levels that the MPF algorithm can operate on independently.

In this section I create an example application for the prediction of molecular structure — another key property in ligand docking. This molecular structure prediction project (abbreviated as project MSP) uses and validates both the GLaCE and MPF algorithms. I also update the MPF algorithm with new definitions of the functions, in order to better adapt to project MSP. The definitions of the GLaCE algorithm and improved MPF algorithm are introduced below.

5.1 Methodology

In this section, the general GLaCE algorithm is introduced in Section 5.1.1 and the model structure design for project MSP is provided in Section 5.1.2. The MPF and NN implementations are also provided in Section 5.1.2.

It is notable that, in this section, I mostly follow the notation used in Section 4.1.1.

However, since the GLaCE algorithm generates several levels of graph and the MPF implementation for each graph is different, some of the notation will be optimised for better presentation. These changes are clearly stated in the text below. Readers are advised to distinguish them carefully. For better understanding of the algorithms introduced below, a notation table is provided in Appendix B.

5.1.1 Graph Latent Connectivity Enhancement (GLaCE) Algorithm

Mathematical Form of GLaCE

The GLaCE algorithm operates on graphs. In this section, the notation of a graph can be defined as

$$X_\lambda = \{V_\lambda, E_\lambda\}, \tag{5.1a}$$

where λ denotes the level of graph. V_λ and E_λ are the node set and edge set for graph X_λ respectively, which read

$$V_\lambda = \left\{v_1^\lambda, v_2^\lambda, \ldots, v_{n^\lambda}^\lambda\right\} \tag{5.1b}$$

and

$$E_\lambda = \left\{e_{ij}^\lambda | (v_i^\lambda, v_j^\lambda) \in V_\lambda \times V_\lambda, i \neq j\right\}. \tag{5.1c}$$

According to the notation defined above, the general process of the GLaCE algorithm can be describe as

$$X_{\lambda+1} = \Gamma(X_\lambda, \Phi_{\lambda+1}, \Xi_{\lambda+1}), \tag{5.2}$$

where Γ denotes the GLaCE operation and $\Xi_{\lambda+1}$ is the interaction function for level $\lambda+1$ which defines the connectivity feature (edge feature) for the next-level graph. $\Phi_{\lambda+1}$ is the implementations of node formation of graph in level $\lambda+1$.

The node formation function $\Phi_{\lambda+1}$ clusters the edge e_{ij}^λ and its connected nodes v_i^λ and v_j^λ in the current level of graph X_λ into a new node in next level $\lambda+1$. It is defined as

$$V_{\lambda+1} = \left\{\Phi_{\lambda+1}(v_i^\lambda, v_j^\lambda, e_{ij}^\lambda) | T_{ij}^\lambda \neq 0\right\}, \tag{5.3}$$

where T_{ij}^λ is an element of the topology matrix of graph X_λ. This topology matrix $\mathbf{T}_\lambda$ is defined according to the interaction function Ξ_λ. An element in the topology matrix is given by

$$T_{ij}^\lambda = \Xi_\lambda(v_i^\lambda, v_j^\lambda), \quad v_i^\lambda, v_j^\lambda \in V_\lambda. \tag{5.4}$$

Here, the interaction function Ξ_λ defines the connectivity of graph X_λ and type of connections between nodes. The bond function defined in project ACP (see Equation (4.1f)) is an example of the interaction function in the graph of molecules. The elements of the adjacency matrix $\mathbf{A}_\lambda$ for level-λ graph is

$$A_{ij}^\lambda = \begin{cases} 1, & \text{if } T_{ij}^\lambda \neq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{5.5}$$

The second step is to build the new edges at level $\lambda+1$, namely the edge formation process. In the edge formation process, the edge set $E_{\lambda+1}$ is set up for the new graph $X_{\lambda+1}$ as

$$E_{\lambda+1} = \left\{ (v_i^{\lambda+1}, v_j^{\lambda+1}) | T_{ij}^{\lambda+1} \neq 0 \right\}. \tag{5.6}$$

Thus, the new graph for level $\lambda + 1$ can be setup as

$$X_{\lambda+1} = \{V_{\lambda+1}, E_{\lambda+1}\}. \tag{5.7}$$

The formal definition of GLaCE is now complete. Here, in the new graph $X_{\lambda+1}$, each node $v_\alpha^{\lambda+1}$ represents the cluster of node v_i^λ and v_j^λ and the edge connecting these two nodes e_{ij}^λ in the previous graph X_λ. Thus, the new graph $X_{\lambda+1}$ gives the connectivity relation between these clusters, presenting further topological information.

In the GLaCE algorithm the node formation function Φ_λ and the interaction function Ξ_λ are user-defined and can be vary between different levels. In addition, the node formation function Φ_λ can contain a filter, returning a null value for inputs that do not meet the required conditions (see below for an example for ethylene). This filtering feature can provide GLaCE with more flexibility in handling complex problem. For further usage in project MSP, see Section 5.1.2.

To avoid potential misunderstanding when using Arabic numerals in the notation, such as v_i^λ and $t_{ij}^{\lambda+1}$, where the level of graph might be mistaken as an exponent, Roman numerals are adopted to label the level of graph (for labelling purposes only) in the following sections. To provide a better understanding of the significance of the GLaCE algorithm, and as an introduction to the project MSP, a detailed definition of the terms in GLaCE for a molecular application is provided in the following section.

GLaCE Algorithm for a Molecular Application

For better understanding of the application of GLaCE to molecules, a representative example of ethylene is given in this section, along with the definition of each level of the graph. Figure 5.1 shows an example of GLaCE operations on an ethylene molecule, including the visualised graphs and topology matrices. Additionally, the lists of nodes for all four levels of graphs are also provided, where the relations between nodes and atoms in a molecule are given.

For a molecular application, the original molecule graph is defined as level-I ($\lambda = 1$, labelled as I). In the level-I graph each node represents an atom and each edge represents a bond, following the definitions used in project ACP. This level-I graph indicates the topological information of the molecules. By applying MPF algorithm to the level-I graph, the atomic feature can be acquired.

The level-II ($\lambda = 2$, labelled as II) graph in a molecular application is named the bond graph. Each node in a bond graph contains the information for a bond and two connected atoms. The bond graph describes the connectivity between bonds — whether they share

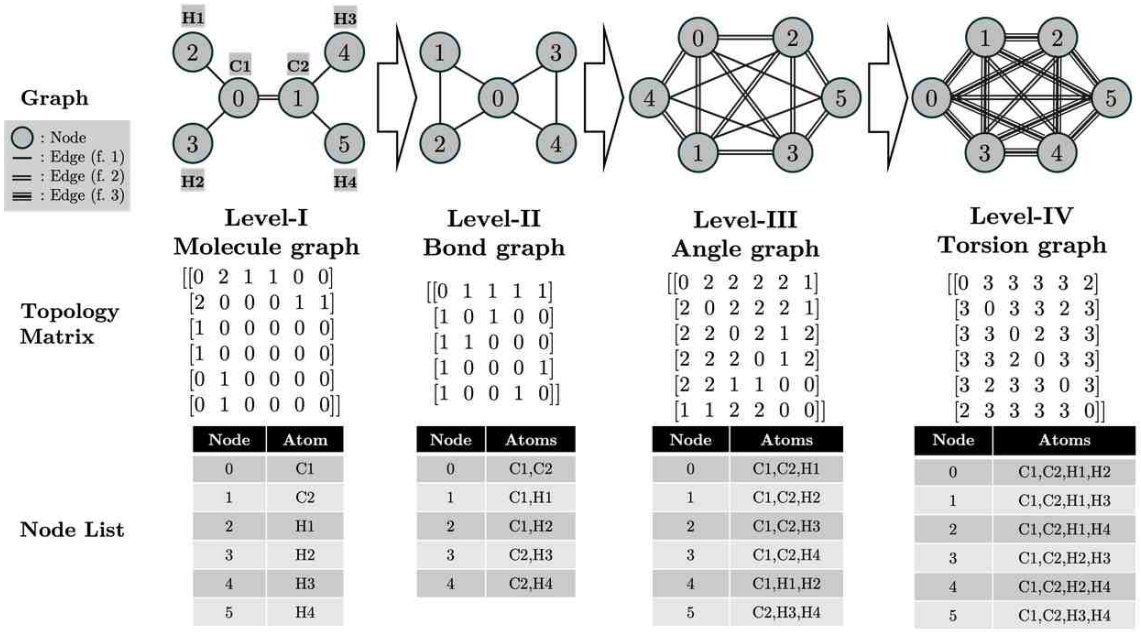


Figure 5.1: GLaCE operations on ethylene molecule: level-I to level-IV.

an atom (if so, the edge feature set to 1). From the level-II bond graph the MPF can generate a feature for each bond and predict the bond features in a later neural network.

The angle graph is defined as the level-III ($\lambda = 3$, labelled as III) graph, which can be obtained by applying GLaCE to the level-II bond graph. In the angle graph each node consists of three atoms and two bonds, i.e. a bond angle, in the original molecule graph (level-I). The edge feature defined in Figure 5.1 is the number of shared atoms between nodes (please note that this definition of an edge feature is only for this example and has no relation with project MSP). By aggregating information through the angle graph via MPF, the properties for the bond angle can be obtained.

The level-IV ($\lambda = 4$, labelled as IV) graph is named the torsion graph, and is generated from the angle graph. In the torsion graph each node denotes a torsion (four connected atoms, three connected bonds or two connected bond angles) in the molecule. While generating nodes for the level-IV torsion graph, a filter is enabled in the node formation function Φ_{IV}. Thus, from the level-IV graph the torsion properties can be predicted.

The descriptions above fully introduces the application of GLaCE to molecules. In project MSP I follow the GLaCE algorithm with the above definitions for the graph levels. The Python code for this ethylene example is provided in Appendix L

5.1.2 General Design of Project MSP

To evaluate the performance of the GLaCE algorithm I create a model for molecular structure prediction. However, unlike predicting atomic charge in project ACP, the direct prediction of molecular structure can be challenging for the following reasons:

1. I am producing a molecular structure based on only connectivity information (SMILES),

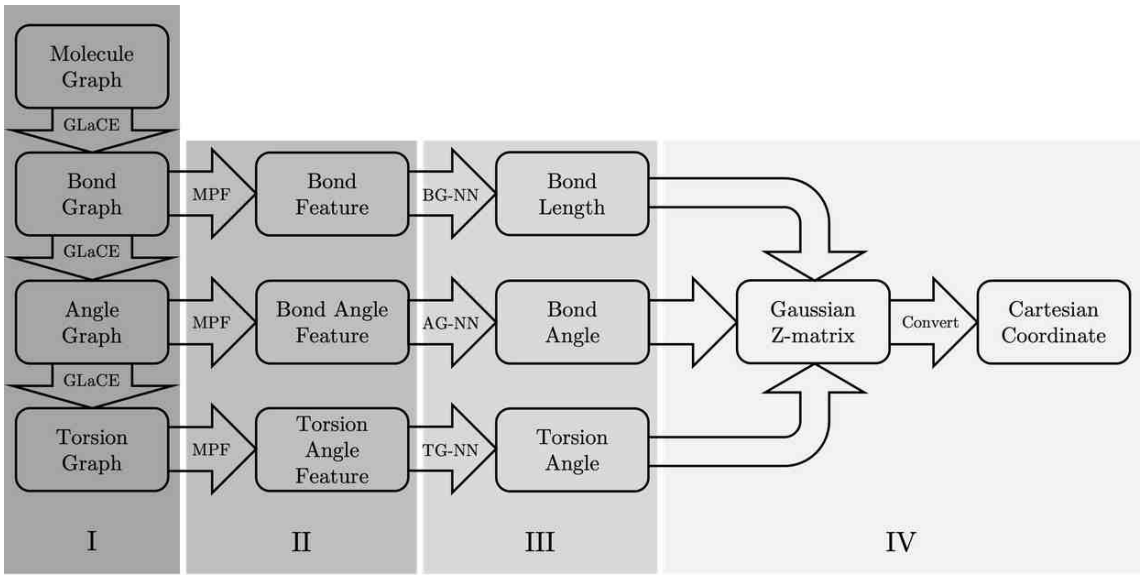


Figure 5.2: The general design of project MSP, including GLaCE operations (stage I), MPF process (stage II), NN prediction (stage III) and post-processing (stage IV).

2. the properties must be rotationally invariant, and

3. I am directly predicting the values that define the structure without any input values.

To overcome the difficulties stated above the MSP model is carefully designed. Here, I choose the Gaussian Z-matrix format as the target output of the model. The Gaussian Z-matrix format stores the molecular structure as bond lengths, bond angles and dihedral angles (i.e. torsion angles[1]). The general workflow of project MSP is given by Figure 5.2 where the entire process consists of four parts, the GLaCE operations (Stage I), MPF process (Stage II), neural network prediction (Stage III) and the post-processing (Stage IV).

In Stage I, the model accepts the input molecule and creates the molecule graph (level-I). By applying GLaCE operations sequentially the level-II bond graph, level-III angle graph and level-IV torsion graph are obtained (see Section 5.1.1 for further information). For further implementation of the GLaCE functions in this process, please see Section 5.1.3.

In Stage II, differently defined MPF operations are applied on the level-II, level-III and level-IV graphs respectively. The product of the MPF stages are the final features for each bond, each bond angle and each torsion angle. For details of the MPF process, readers are referred to Section 5.1.4.

Stage III performs the neural network prediction for the features generated in Stage II, operating on bond, bond angle and torsion angle respectively. The detailed design of the neural network in project MSP is also presented in Section 5.1.4. The last process, Stage IV in Figure 5.2, is to combine the predicted result (bond lengths, bond angles and torsion

[1]The term "torsion angle" is commonly used in the context of biological molecules, specifically referring to the dihedral angles found in proteins. As this work is inspired by a biological scenario, I adopt the term "torsion angle" to maintain consistency with the conventional terminology.

angles) from Stage III into a Gaussian Z-matrix file which can fully define the structure of the molecule.

From the above design the input of the model can be only a SMILES string: the SMILES string is then converted into the mol2 format (same approach as that in project ACP, without coordinate generation when predicting only). Here, the first challenge of SMILES input is satisfied. For the second challenge, this model directly predicts bond lengths, bond angles and torsion angles, which ensures the results are rotationally invariant. Moreover, through the GLaCE and MPF algorithm, the structure of the molecule is directly predicted via bond lengths, bond angles and torsion angles, indicating that the third challenge can be also solved.

5.1.3 Implementation of GLaCE Algorithm

In this section, the implementation of the GLaCE algorithm in project MSP is presented. Here, the input molecular graph (level-I) is assumed to be fully-defined. The definition of the node formation function and the interaction function for level-II, level-III and level-IV graph generations are different, for optimisation for various scenarios. See below.

Obtaining Level-II Bond Graph

By applying GLaCE on the input molecule graph (level-I), the bond graph can be obtained. The node formation function in level-II reads

$$v_\alpha^{\mathrm{II}} = \Phi_{\mathrm{II}}(v_i^{\mathrm{I}}, v_j^{\mathrm{I}}, e_{ij}^{\mathrm{I}}) = \left\{\nu_\alpha^{\mathrm{II}}, \varepsilon_\alpha^{\mathrm{II}}, \varphi_{\mathrm{atom},\alpha}^{\mathrm{II}}, \varphi_{\mathrm{bond},\alpha}^{\mathrm{II}}\right\}, \quad \text{if } T_{ij}^{\mathrm{I}} \neq 0, \tag{5.8a}$$

where

$$\nu_\alpha^{\mathrm{II}} = \left\{v_i^{\mathrm{I}}, v_j^{\mathrm{I}}\right\} \quad \text{and} \quad \varepsilon_\alpha^{\mathrm{II}} = \left\{e_{ij}^{\mathrm{I}}\right\}.$$

Here, the node v_α^{II} consist of two subsets, the atom list subset ν_α^{II} and the bond list subset $\varepsilon_\alpha^{\mathrm{II}}$. In this level-II bond graph, the subset ν_α^{II} has two atoms, while the subset $\varepsilon_\alpha^{\mathrm{II}}$ has one bond. In Equation (5.8a) $\varphi_{\mathrm{atom},\alpha}^{\mathrm{II}} \in \mathbb{R}$ and $\varphi_{\mathrm{bond},\alpha}^{\mathrm{II}} \in \mathbb{R}$ are the atom-related constant features and bond-related constant features for node α in level-II bond graph respectively. They are defined as

$$\varphi_{\mathrm{atom},\alpha}^{\mathrm{II}} = \sum_{i,j} \chi_i \cdot T_{ij}^{\mathrm{I}} \cdot \chi_j, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_\alpha^{\mathrm{II}} \tag{5.8b}$$

and

$$\varphi_{\mathrm{bond},\alpha}^{\mathrm{II}} = \sum_{i,j} T_{ij}^{\mathrm{I}}, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_\alpha^{\mathrm{II}}. \tag{5.8c}$$

where here and below the notation $e_{ij}^{\mathrm{I}} \in \varepsilon_\alpha^{\mathrm{II}}$ means the values of i and j are restricted to the values that satisfy this condition. These two constant features are used to represent the inherent properties of the node. By using the sum of the values the ordering problem of the atom and bond can be eliminated (also applied to later levels of the graphs), compared to direct listing of the values.

The interaction function Ξ_{II} for the bond graph is defined as

$$T_{ij}^{\mathrm{II}} = \Xi_{\mathrm{II}}(v_i^{\mathrm{II}}, v_j^{\mathrm{II}}) = \begin{cases} 1, & \text{if } \nu_i^{\mathrm{II}} \cap v_j^{\mathrm{II}} \neq \emptyset \text{ and } i \neq j, \\ 0, & \text{otherwise.} \end{cases} \tag{5.8d}$$

Here, the interaction (i.e. the edge feature or corresponding element in the topology matrix $\mathbf{T}_{\mathrm{II}}$) is defined as either 0 or 1, depending on whether these bonds have a shared atom or not. Therefore, the edge formation is defined as

$$e_{ij}^{\mathrm{II}} = \{v_i^{\mathrm{II}}, v_j^{\mathrm{II}}\}, \quad \text{if } T_{ij}^{\mathrm{II}} \neq 0, \tag{5.8e}$$

where the edge e_{ij}^{II} is formed only if the interaction T_{ij}^{II} between node i and node j is not zero. Thus the level-II bond graph can be defined as

$$X_{\mathrm{II}} = \{V_{\mathrm{II}}, E_{\mathrm{II}}\}, \tag{5.8f}$$

where V_{II} is the node set for the level-II bond graph and E_{II} is the edge set for the level-II bond graph.

Obtaining the Level-III Angle Graph

The level-III angle graph can be obtained by applying GLaCE to the level-II bond graph. Here the node formation function is defined as

$$v_\alpha^{\mathrm{III}} = \Phi_{\mathrm{II}}(v_i^{\mathrm{II}}, v_j^{\mathrm{II}}, e_{ij}^{\mathrm{II}}) = \left\{\nu_\alpha^{\mathrm{III}}, \varepsilon_\alpha^{\mathrm{III}}, \varphi_{\mathrm{atom},\alpha}^{\mathrm{III}}, \varphi_{\mathrm{bond},\alpha}^{\mathrm{II}}\right\}, \quad \text{if } T_{ij}^{\mathrm{II}} \neq 0, \tag{5.9a}$$

where

$$\nu_\alpha^{\mathrm{III}} = \nu_i^{\mathrm{II}} \cup \nu_j^{\mathrm{II}} \quad \text{and} \quad \varepsilon_\alpha^{\mathrm{III}} = \varepsilon_i^{\mathrm{II}} \cup \varepsilon_j^{\mathrm{II}}.$$

Since each node in the level-II graph contains two atoms and nodes v_i^{II} and v_j^{II} share one atom (because $T_{ij}^{\mathrm{II}} \neq 0$), the atom list set $\nu_\alpha^{\mathrm{III}}$ of node α in the level-III angle graph has three atoms. Similarly, each node in the level-II bond graph has only one bond and there is no sharing bond between nodes in level-II. Thus the bond list set $\varepsilon_\alpha^{\mathrm{III}}$ consist of two bonds. In summary, each node in the level-III angle graph consists of three atoms and two bonds. Here, the atom-related constant feature and bond-related constant feature are defined similarly to Equations (5.8b) and (5.8c) as

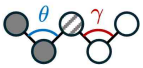
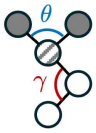
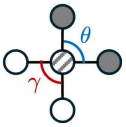
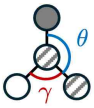
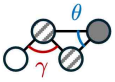
$$\varphi_{\mathrm{atom},\alpha}^{\mathrm{III}} = \sum_{i,j} \chi_i \cdot T_{ij}^{\mathrm{I}} \cdot \chi_j, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_\alpha^{\mathrm{III}} \tag{5.9b}$$

and

$$\varphi_{\mathrm{bond},\alpha}^{\mathrm{III}} = \sum_{i,j} T_{ij}^{\mathrm{I}}, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_\alpha^{\mathrm{III}}. \tag{5.9c}$$

The definition of the interaction function in the level-III angle graph is given according to the connection case given in Table 5.1. Here, a total of eight cases is defined, including all possible interactions between two bond angles in the molecule graph (level-I). Thus, the

Table 5.1: Interaction definition of level-III angle graph in different cases. θ (in blue colour) and γ (in red colour) denote two corresponding bond angles in the level-I molecule graph. The solid circles in the graph denote the atoms belonging to bond angle θ, the hollow circles denote the atoms belonging to bond angle γ and the diagonally hatched circles denote the shared atoms.

Number of Shared Atoms	**Cases**	**Schematic in Level-I Molecule Graph**	**Assigned Feature Value**
1	Case 1		1
	Case 2		2
	Case 3		3
2	Case 4		4
	Case 5		5
	Case 6		6
	Case 7		7
3	Case 8		8

interaction function reads

$$T_{ij}^{\mathrm{III}} = \Xi_{\mathrm{III}}(v_i^{\mathrm{III}}, v_j^{\mathrm{III}}) = \begin{cases} 1, & \text{if Case 1 applies,} \\ 2, & \text{if Case 2 applies,} \\ 3, & \text{if Case 3 applies,} \\ 4, & \text{if Case 4 applies,} \\ 5, & \text{if Case 5 applies,} \\ 6, & \text{if Case 6 applies,} \\ 7, & \text{if Case 7 applies,} \\ 8, & \text{if Case 8 applies,} \\ 0, & \text{if } i = j \text{ or bond angles are not connected.} \end{cases} \tag{5.9d}$$

I here use integer numbers to represent the interaction type of bond angles (see Table 5.1 for more information), where numbers 1 to 8 denote different interaction cases and 0 denotes no connection between two bond angles. The edge formation function now reads

$$e_{ij}^{\mathrm{III}} = \left\{ v_i^{\mathrm{III}}, v_j^{\mathrm{III}} \right\}, \quad \text{if } T_{ij}^{\mathrm{III}} \neq 0, \tag{5.9e}$$

and the level-III angle graph can be defined as

$$X_{\mathrm{III}} = \{V_{\mathrm{III}}, E_{\mathrm{III}}\} \tag{5.9f}$$

The level-III angle graph has now been generated.

Obtaining Level-IV Torsion Graph

To generate the level-IV torsion graph, the GLaCE algorithm is applied to the level-III angle graph to generate nodes representing torsion angle in molecules and edges that define the interaction between torsion angles. The nodes in the level-IV torsion graph can be defined as

$$v_\alpha^{\mathrm{IV}} = \Phi_{\mathrm{III}}(v_i^{\mathrm{III}}, v_j^{\mathrm{III}}, e_{ij}^{\mathrm{III}}) = \left\{ \nu_\alpha^{\mathrm{IV}}, \varepsilon_\alpha^{\mathrm{IV}}, \varphi_{\mathrm{atom},\alpha}^{\mathrm{IV}}, \varphi_{\mathrm{bond},\alpha}^{\mathrm{IV}} \right\}, \quad \text{if } T_{ij}^{\mathrm{III}} = 4, \tag{5.10a}$$

where

$$\nu_\alpha^{\mathrm{IV}} = \nu_i^{\mathrm{III}} \cup \nu_j^{\mathrm{III}} \quad \text{and} \quad \varepsilon_\alpha^{\mathrm{IV}} = \varepsilon_i^{\mathrm{III}} \cup \varepsilon_j^{\mathrm{III}}.$$

The definitions of two constant features are

$$\varphi_{\mathrm{atom},\alpha}^{\mathrm{IV}} = \sum_{i,j} \chi_i \cdot T_{ij}^{\mathrm{I}} \cdot \chi_j, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_\alpha^{\mathrm{IV}} \tag{5.10b}$$

and

$$\varphi_{\mathrm{bond},\alpha}^{\mathrm{IV}} = \sum_{i,j} T_{ij}^{\mathrm{I}}, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_\alpha^{\mathrm{IV}}. \tag{5.10c}$$

Here, a constraint $T_{ij}^{\mathrm{III}} = 4$ is applied to this node formation process. According to the introduction in the previous section, Table 5.1 indicates eight cases of bond angle interaction. These cases can be classified into three groups according to the number of shared atoms between two bond angles. The bond angles in Cases 1 to 3 (inclusive) share one atom and bond angles in Case 4 to 7 (inclusive) share two atoms, while the bond angles in Case 8 share three atoms. Since a node in the level-IV torsion graph should consist of exactly four atoms, only Cases 4, 5, 6 & 7 can form four-atom fragments, which can be converted to nodes in the level-IV torsion graph. In addition, in Cases 5, 6 & 7 the torsion angles have been already defined, since there is no remaining degree of freedom if all bond angles are defined according to level-III angle graph. Therefore, the only valid case in the torsion graph is Case 4.

In the level-IV torsion graph the interaction function is defined as the number of shared

atoms (nodes in the level-I graph) between two torsions, which is

$$T_{ij}^{\mathrm{IV}} = \Xi(v_i^{\mathrm{IV}}, v_j^{\mathrm{IV}}) = \begin{cases} 1, & \text{if } \left|\nu_i^{\mathrm{IV}} \cap \nu_j^{\mathrm{IV}}\right| = 1 \text{ and } i \neq j, \\ 2, & \text{if } \left|\nu_i^{\mathrm{IV}} \cap \nu_j^{\mathrm{IV}}\right| = 2 \text{ and } i \neq j, \\ 3, & \text{if } \left|\nu_i^{\mathrm{IV}} \cap \nu_j^{\mathrm{IV}}\right| = 3 \text{ and } i \neq j, \\ 4, & \text{if } \left|\nu_i^{\mathrm{IV}} \cap \nu_j^{\mathrm{IV}}\right| = 4 \text{ and } i \neq j, \\ 0, & \text{otherwise.} \end{cases} \tag{5.10d}$$

Since the specific cases of interactions between torsions are highly diverse I here use a simple approach, i.e. the number of shared atoms, to define the connection properties in the level-IV torsion graph. This avoids introducing significant complexity to the graph and ultimately adding complexity to the neural network. Therefore, the edge formation is

$$e_{ij}^{\mathrm{IV}} = \{v_i^{\mathrm{IV}}, v_j^{\mathrm{IV}}\}, \quad \text{if } T_{ij}^{\mathrm{IV}} \neq 0, \tag{5.10e}$$

and the level-IV topology graph can be defined as

$$X_{\mathrm{IV}} = \{V_{\mathrm{IV}}, E_{\mathrm{IV}}\}. \tag{5.10f}$$

5.1.4 Implementation of MPF Algorithm and Neural Network

In this section, the implementation of MPF algorithm in project MSP is presented for the level-II bond graph, the level-III angle graph and the level-IV torsion graph. Please note that in this section the notation follows the usage in Section 4.1.1, with additional graph level subscripts or superscripts. To avoid redundancy, only customisable functions in the MPF are provided below and all notations already defined in Section 4.1.1 are not listed here. Readers are advised to check the original definitions given in Section 4.1.1 and the notation table provided in Appendix B for the meaning of MPF-related notations.

Bond Graph (Level-II)

Atomic Constant Properties The atomic constant properties used in the level-II bond graph are

$$C_{\mathrm{II}} = \{\mathbf{c}_\chi\}, \tag{5.11}$$

where $\mathbf{c}_\chi$ is the Pauling-scale electronegativity vector (Equation (4.6b)). Here, I also adopt the notation χ_i as the electronegativity for atom i the same as Equation (4.6c).

Node Feature Definition In the level-II bond graph, each node has two features ($d^{\mathrm{II}} = 2$), which are

$$\mathbf{h}_{t,\alpha}^{\mathrm{II}} = \left[\mathrm{h}_{t,\alpha}^{\mathrm{II,atom}}, \mathrm{h}_{t,\alpha}^{\mathrm{II,bond}}\right]. \tag{5.12}$$

Here, $\mathrm{h}_{t,\alpha}^{\mathrm{II,atom}}$ denotes the atom-related feature and $\mathrm{h}_{t,\alpha}^{\mathrm{II,bond}}$ denotes the bond-related feature of node α.

Initialisation Function These node features are initialised to their corresponding node constant feature as

$$\mathrm{h}_{0,\alpha}^{\mathrm{II,atom}} = {I'}_{\mathrm{atom}}^{\mathrm{II}}(v_\alpha^{\mathrm{II}}) = \varphi_{\mathrm{atom},\alpha}^{\mathrm{II}} = \sum_{i,j} \chi_i T_{ij}^{\mathrm{I}} \chi_j, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_\alpha^{\mathrm{II}} \tag{5.13a}$$

and

$$\mathrm{h}_{0,\alpha}^{\mathrm{II,bond}} = {I'}_{\mathrm{bond}}^{\mathrm{II}}(v_\alpha^{\mathrm{II}}) = \varphi_{\mathrm{bond},\alpha}^{\mathrm{II}} = \sum_{i,j} T_{ij}^{\mathrm{I}}, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_\alpha^{\mathrm{II}}. \tag{5.13b}$$

The atom-related feature $\mathrm{h}_{t,\alpha}^{\mathrm{II,atom}}$ is designed to emphasise the electronegativity feature of the atoms and their relations in the node α, while the bond-related feature $\mathrm{h}_{t,\alpha}^{\mathrm{II,bond}}$ is designed to highlight the bond order feature in the node α.

Message Generation The message vector in the MPF process of the level-II bond graph contains two messages ($k^{\mathrm{II}} = 2$): a node contribution message $\mathbf{m}_{t+1,\alpha}^{\mathrm{II}}$ and an edge contribution message $\mathrm{m}_{t+1,\alpha}^{\mathrm{II,node}}$. The mathematical form can be written as

$$\mathbf{m}_{t+1,\alpha}^{\mathrm{II}} = \left[\mathrm{m}_{t+1,\alpha}^{\mathrm{II,node}}, \mathrm{m}_{t+1,\alpha}^{\mathrm{II,edge}}\right], \tag{5.14a}$$

where

$$\mathrm{m}_{t+1,\alpha}^{\mathrm{II,node}} = M'_{\mathrm{node}}(P_t^{\mathrm{II}}, \alpha) = \sum_i \mathrm{h}_{t,i}^{\mathrm{II,atom}} \cdot A_{i\alpha}^{\mathrm{II}} \tag{5.14b}$$

and

$$\mathrm{m}_{t+1,\alpha}^{\mathrm{II,edge}} = M'_{\mathrm{edge}}(P_t^{\mathrm{II}}, \alpha) = \sum_i \mathrm{h}_{t,i}^{\mathrm{II,bond}} \cdot T_{i\alpha}^{\mathrm{II}}. \tag{5.14c}$$

Here, the node contribution message $\mathrm{m}_{t+1,\alpha}^{\mathrm{II,node}}$ is the sum of the atom-related features $\mathrm{h}_{t,\alpha}^{\mathrm{II,atom}}$ of all neighbouring nodes. The edge contribution message $\mathrm{m}_{t+1,\alpha}^{\mathrm{II,edge}}$ is a sum of the product of bond-related features $\mathrm{h}_{t,\alpha}^{\mathrm{II,edge}}$ and the connectivity feature T_{ij}^{II} (which is the same as the adjacency matrix in the level-II bond graph). The primary reason for using the summed values is to eliminate the problem of the order of features in the message vector, compared to the direct listing of the features.

Graph Update The update functions used at level-II are defined for atom-related features $\mathrm{h}_{t+1,\alpha}^{\mathrm{II,atom}}$ and bond-related features $\mathrm{h}_{t+1,\alpha}^{\mathrm{II,bond}}$ respectively as

$$\mathrm{h}_{t+1,\alpha}^{\mathrm{II,atom}} = {U'}_{\mathrm{atom}}^{\mathrm{II}}(\mathbf{M}_{t+1}^{\mathrm{II}}, \alpha) = \mathrm{m}_{t+1,\alpha}^{\mathrm{II,node}} \tag{5.15a}$$

and

$$\mathrm{h}_{t+1,\alpha}^{\mathrm{II,bond}} = {U'}_{\mathrm{bond}}^{\mathrm{II}}(\mathbf{M}_{t+1}^{\mathrm{II}}, \alpha) = \mathrm{m}_{t+1,\alpha}^{\mathrm{II,edge}}. \tag{5.15b}$$

Here in the level-II bond graph, 3 rounds ($m^{\mathrm{II}} = 3$) of the update are run. It is notable that the updated node features here are completely replaced by the corresponding messages that are generated according to surrounding nodes. This is because an improved algorithm is used in the readout function which re-introduces the features of local nodes. See below.

Feature Readout Here, an accumulated node feature matrix $\mathbf{H}_{\mathrm{acc}}^{\mathrm{II}}$ for the bond graph is

defined before the readout process as

$$\begin{aligned}\mathbf{H}^{\mathrm{II}}_{\mathrm{acc}} &= \left[\mathbf{h}^{\mathrm{II}}_{\mathrm{acc},1}, \mathbf{h}^{\mathrm{II}}_{\mathrm{acc},2}, \ldots, \mathbf{h}^{\mathrm{II}}_{\mathrm{acc},n^{\mathrm{II}}}\right] = \sum_{t=0}^{m^{\mathrm{II}}} \frac{1}{t!}\mathbf{H}^{\mathrm{II}}_{t}, \\ \mathbf{h}^{\mathrm{II}}_{\mathrm{acc},i} &= \left[\mathrm{h}^{\mathrm{II,atom}}_{\mathrm{acc},i}, \mathrm{h}^{\mathrm{II,bond}}_{\mathrm{acc},i}\right].\end{aligned} \tag{5.16}$$

This accumulated node feature adds up all node feature matrices for each time step with a factorial-weighted factor, which can re-introduce the lost local feature in the update process Equations (5.15a) and (5.15b) and add proper decay for those nodes that are far from the local node. The readout function in the level-II bond graph reads

$$\mathbf{f}^{\mathrm{II}}_{\alpha} = R'^{\mathrm{II}}(P^{\mathrm{II}}_{m}, \mathbf{H}^{\mathrm{II}}_{\mathrm{acc}}, \mathbf{M}^{\mathrm{II}}_{m}, \alpha) = \left[\mathrm{f}^{\mathrm{II,atom}}_{\alpha}, \mathrm{f}^{\mathrm{II,bond}}_{\alpha}, \mathrm{f}^{\mathrm{II,catom}}_{\alpha}, \mathrm{f}^{\mathrm{II,cbond}}_{\alpha}\right]. \tag{5.17a}$$

In the level-II bond graph each node has four final features ($s^{\mathrm{II}} = 4$) in the feature vector. They are the atom-related final feature

$$\mathrm{f}^{\mathrm{II,atom}}_{\alpha} = \mathrm{h}^{\mathrm{II,atom}}_{\mathrm{acc},\alpha}, \tag{5.17b}$$

the bond-related final feature

$$\mathrm{f}^{\mathrm{II,bond}}_{\alpha} = \mathrm{h}^{\mathrm{II,bond}}_{\mathrm{acc},\alpha}, \tag{5.17c}$$

the atom-related constant feature

$$\mathrm{f}^{\mathrm{II,catom}}_{\alpha} = \varphi^{\mathrm{II}}_{\mathrm{atom},\alpha} \tag{5.17d}$$

and the bond-related constant feature

$$\mathrm{f}^{\mathrm{II,cbond}}_{\alpha} = \varphi^{\mathrm{II}}_{\mathrm{bond},\alpha}. \tag{5.17e}$$

NN Preprocessing In the preprocessing before the NN prediction, the fetaures are normalised, similarly to Equation (4.11a) as

$$\begin{aligned}\mathbf{F}'^{\mathrm{II}} &= \left[\mathbf{f}'^{\mathrm{II}}_{1}, \mathbf{f}'^{\mathrm{II}}_{2}, \ldots, \mathbf{f}'^{\mathrm{II}}_{n^{\mathrm{II}}}\right] \\ \mathbf{f}'^{\mathrm{II}}_{\alpha} &= \left[\mathrm{f}'^{\mathrm{II,atom}}_{\alpha}, \mathrm{f}'^{\mathrm{II,bond}}_{\alpha}, \mathrm{f}'^{\mathrm{II,catom}}_{\alpha}, \mathrm{f}'^{\mathrm{II,cbond}}_{\alpha}\right] \\ \mathrm{f}'^{\mathrm{II},j}_{\alpha} &= \frac{\mathrm{f}^{\mathrm{II},j}_{\alpha} - \mu^{\mathrm{II,preset}}_{j}}{\sigma^{\mathrm{II,preset}}_{j}}, \quad j \in \{\mathrm{atom}, \mathrm{bond}, \mathrm{catom}, \mathrm{cbond}\}.\end{aligned} \tag{5.18}$$

NN Implementation The neural network used for level-II bond length prediction is a 5-layer-100-node fully-connect neural network. Thus, the layer index l has the same definition

as in Equation (4.12a) given in project ACP. The number of neurons in each layer l is

$$\begin{cases} q_0^{\mathrm{II}} = 4, & \text{input layer,} \\ q_l^{\mathrm{II}} = 100, l \in \{1,2,3,4,5\}, & \text{hidden layers,} \\ q_6^{\mathrm{II}} = 1, & \text{output layer.} \end{cases} \tag{5.19}$$

In total there are 41,001 trainable parameters in the network. Here, I use tanh as the activation function for the hidden layers. So the entire neural network reads

$$\begin{aligned} \mathbf{G}_\alpha^{\mathrm{II},0} &= \mathbf{f}'^{\mathrm{II}}_\alpha && \text{input layer,} \\ \mathbf{G}_\alpha^{\mathrm{II},l} &= \tanh(\mathbf{W}^{\mathrm{II},l-1}\mathbf{G}_\alpha^{\mathrm{II},l-1} + \mathbf{b}^{\mathrm{II},l-1}) && \text{hidden layers, } l \in \{1,2,3,4,5\}, \text{ and} \\ \mathbf{y}'^{\mathrm{II}}_\alpha &= [\mathrm{y}'^{\mathrm{II}}_\alpha] = \sigma_{\mathrm{II}}\left(\mathbf{W}^{\mathrm{II},5}\mathbf{G}_\alpha^{\mathrm{II},5} + \mathbf{b}^{\mathrm{II},5}\right) && \text{output layer.} \end{aligned} \tag{5.20}$$

The σ_{II} is an extra activation function for output layer in order to constraint the value to avoid invalid output (such as negative bond length). It is defined as

$$\sigma_{\mathrm{II}}(x) = \begin{cases} \mathrm{norm}_{\mathrm{II}}(0), & \text{if } x < \mathrm{norm}_{\mathrm{II}}(0), \\ x, & \text{otherwise.} \end{cases} \tag{5.21}$$

Here, the normalisation process (Equation (5.18)) is represented by $\mathrm{norm}_{\mathrm{II}}$. The normalised output vector $\mathbf{y}'^{\mathrm{II}}_\alpha$ contains only one element $\mathrm{y}'^{\mathrm{II}}_\alpha$, i.e. the bond length. The model is trained for 100 epochs with a batch size of 256. The optimiser is Adam ($\eta = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$).

NN Post-processing To obtain the predicted bond length, a denormalisation process is applied as

$$\begin{aligned} \mathbf{Y}_{\mathrm{II}} &= [\mathbf{y}_1^{\mathrm{II}}, \mathbf{y}_2^{\mathrm{II}}, \ldots, \mathbf{y}_{n^{\mathrm{II}}}^{\mathrm{II}}], \\ \mathbf{y}_\alpha^{\mathrm{II}} &= [\mathrm{y}_\alpha^{\mathrm{II}}], \\ \mathrm{y}_\alpha^{\mathrm{II}} &= \sigma_{\mathrm{y}}^{\mathrm{II,preset}} \cdot \mathrm{y}'^{\mathrm{II}}_\alpha + \mu_{\mathrm{y}}^{\mathrm{II,preset}}. \end{aligned} \tag{5.22}$$

By the implementation given above, the bond length information can be predicted.

Angle Graph (Level-III)

Atomic Constant Properties The atomic constant property defined in the level-III angle graph has two features, defined as

$$C^{\mathrm{III}} = \{\mathbf{c}_\chi, \mathbf{c}_\zeta\}. \tag{5.23a}$$

Here, $\mathbf{c}_\zeta$ denotes the vector of the number of valence electrons of atoms, where I define

$$\zeta_i = \mathbf{c}_\zeta[Z_i], \tag{5.23b}$$

as the number of valence electrons of atom i, where Z_i denotes the atomic number of atom i. In addition, χ_i is defined as the Pauling-scaling electronegativity of atom i (the same as

Equation (4.6c)).

Node Feature Definition There are also two features ($d^{\mathrm{III}} = 2$) defined in each node of the level-III angle graph, which are the atom-related feature $\mathrm{h}_{t,\alpha}^{\mathrm{III,atom}}$ and the bond-related feature $\mathrm{h}_{t,\alpha}^{\mathrm{III,bond}}$. Thus, the node feature vector for atom α is defined as

$$\mathbf{h}_{t,\alpha}^{\mathrm{III}} = \left[\mathrm{h}_{t,\alpha}^{\mathrm{III,atom}}, \mathrm{h}_{t,\alpha}^{\mathrm{III,bond}}\right]. \tag{5.24}$$

Initialisation Function The node features in the level-III angle graph are also initialised to the corresponding node constant feature. This process can be written as

$$\mathrm{h}_{0,\alpha}^{\mathrm{III,atom}} = {I'}_{\mathrm{atom}}^{\mathrm{III}}(v_{\alpha}^{\mathrm{III}}) = \varphi_{\mathrm{atom},\alpha}^{\mathrm{III}} = \sum_{i,j} \chi_i T_{ij}^{\mathrm{I}} \chi_j, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_{\alpha}^{\mathrm{III}} \tag{5.25a}$$

and

$$\mathrm{h}_{0,\alpha}^{\mathrm{III,bond}} = {I'}_{\mathrm{bond}}^{\mathrm{III}}(v_{\alpha}^{\mathrm{III}}) = \varphi_{\mathrm{bond},\alpha}^{\mathrm{III}} = \sum_{i,j} T_{ij}^{\mathrm{I}}, \quad e_{ij}^{\mathrm{I}} \in \varepsilon_{\alpha}^{\mathrm{III}}. \tag{5.25b}$$

Message Generation The Message Generation process in the level-III angle graph is similar to that in the level-II bond graph. The message vector for each node is

$$\mathbf{m}_{t+1,\alpha}^{\mathrm{III}} = \left[\mathrm{m}_{t+1,\alpha}^{\mathrm{III,node}}, \mathrm{m}_{t+1,\alpha}^{\mathrm{III,edge}}\right], \tag{5.26a}$$

where the two messages ($k^{\mathrm{III}} = 2$) are the node contribution message

$$\mathrm{m}_{t+1,\alpha}^{\mathrm{III,node}} = M'_{\mathrm{node}}(P_t^{\mathrm{III}}, \alpha) = \sum_i \mathrm{h}_{t,i}^{\mathrm{III,atom}} \cdot A_{i\alpha}^{\mathrm{III}} \tag{5.26b}$$

and the edge contribution message

$$\mathrm{m}_{t+1,\alpha}^{\mathrm{III,edge}} = M'_{\mathrm{edge}}(P_t^{\mathrm{III}}, \alpha) = \sum_i \mathrm{h}_{t,i}^{\mathrm{III,bond}} \cdot T_{i\alpha}^{\mathrm{III}}. \tag{5.26c}$$

Graph Update In the MPF process for the level-III angle graph the update functions are the same as that used in the level-II bond graph, which are

$$\mathrm{h}_{t+1,\alpha}^{\mathrm{III,atom}} = {U'}_{\mathrm{atom}}^{\mathrm{III}}(\mathbf{M}_{t+1}^{\mathrm{III}}, \alpha) = \mathrm{m}_{t+1,\alpha}^{\mathrm{III,node}} \tag{5.27a}$$

and

$$\mathrm{h}_{t+1,\alpha}^{\mathrm{III,bond}} = {U'}_{\mathrm{bond}}^{\mathrm{III}}(\mathbf{M}_{t+1}^{\mathrm{III}}, \alpha) = \mathrm{m}_{t+1,\alpha}^{\mathrm{III,edge}}. \tag{5.27b}$$

The level-III angle graph is updated for 3 rounds ($m^{\mathrm{III}} = 3$).

Feature Readout The accumulated node feature is also used in the level-III angle graph.

The accumulated node feature matrix reads

$$\begin{aligned}\mathbf{H}_{\text{acc}}^{\text{III}} &= \left[\mathbf{h}_{\text{acc},1}^{\text{III}}, \mathbf{h}_{\text{acc},2}^{\text{III}}, \ldots, \mathbf{h}_{\text{acc},n^{\text{III}}}^{\text{III}}\right] = \sum_{t=0}^{m^{\text{III}}} \frac{1}{t!}\mathbf{H}_t^{\text{III}}, \\ \mathbf{h}_{\text{acc},i}^{\text{III}} &= \left[\mathrm{h}_{\text{acc},i}^{\text{III,atom}}, \mathrm{h}_{\text{acc},i}^{\text{III,bond}}\right]\end{aligned} \tag{5.28}$$

with 7 features ($s^{\text{III}} = 7$) in the final feature vector for each node in the level-III angle graph. The final feature vector for node α reads

$$\begin{aligned}\mathbf{f}_\alpha^{\text{III}} &= R'^{\text{III}}(P_m^{\text{III}}, \mathbf{H}_{\text{acc}}^{\text{III}}, \mathbf{M}_m^{\text{III}}, \alpha) \\ &= \left[\mathrm{f}_\alpha^{\text{III,atom}}, \mathrm{f}_\alpha^{\text{III,bond}}, \mathrm{f}_\alpha^{\text{III,catom}}, \mathrm{f}_\alpha^{\text{III,cbond}}, \mathrm{f}_\alpha^{\text{III,valance}}, \mathrm{f}_\alpha^{\text{III,nbond}}, \mathrm{f}_\alpha^{\text{III,sring}}\right]\end{aligned} \tag{5.29a}$$

where the atom-related final feature is

$$\mathrm{f}_\alpha^{\text{III,atom}} = \mathrm{h}_{\text{acc},\alpha}^{\text{III,atom}}, \tag{5.29b}$$

the bond-related final feature is

$$\mathrm{f}_\alpha^{\text{III,bond}} = \mathrm{h}_{\text{acc},\alpha}^{\text{III,bond}}, \tag{5.29c}$$

the atom-related constant feature is

$$\mathrm{f}_\alpha^{\text{III,catom}} = \varphi_{\text{atom},\alpha}^{\text{III}}, \tag{5.29d}$$

the bond-related constant feature is

$$\mathrm{f}_\alpha^{\text{III,cbond}} = \varphi_{\text{bond},\alpha}^{\text{III}}, \tag{5.29e}$$

the valence electron number of the central atom is

$$\mathrm{f}_\alpha^{\text{III,valance}} = \zeta_i, \quad \text{where atom } i \text{ is the central atom of bond angle } \alpha, \tag{5.29f}$$

the number of bond connected to the central atom is

$$\mathrm{f}_\alpha^{\text{III,nbond}} = \sum_j A_{ij}^{\text{I}}, \quad \text{where atom } i \text{ is the central atom of bond } \alpha \tag{5.29g}$$

and the minimum ring size that the bond angle is in

$$\mathrm{f}_\alpha^{\text{III,sring}} = \begin{cases} r_\alpha^{\text{III}}, & \text{if bond angle } \alpha \text{ locates in rings,} \\ 0, & \text{otherwise,} \end{cases} \tag{5.29h}$$

In Equation (5.29h), r_α denotes the minimum size of rings, since a bond angle can be present within several rings at the same time.

NN Preprocessing The normalisation process in the level-III angle graph is the same as

that in the level-II bond graph, which is

$$\begin{aligned}
\mathbf{F}'^{\mathrm{III}} &= \left[\mathbf{f}'^{\mathrm{III}}_1, \mathbf{f}'^{\mathrm{III}}_2, \ldots, \mathbf{f}'^{\mathrm{III}}_{n^{\mathrm{III}}}\right] \\
\mathbf{f}'^{\mathrm{III}}_\alpha &= \left[\mathrm{f}'^{\mathrm{III,atom}}_\alpha, \mathrm{f}'^{\mathrm{III,bond}}_\alpha, \mathrm{f}'^{\mathrm{III,catom}}_\alpha,\right. \\
&\left.\mathrm{f}'^{\mathrm{III,cbond}}_\alpha, \mathrm{f}'^{\mathrm{III,valence}}_\alpha, \mathrm{f}'^{\mathrm{III,nbond}}_\alpha, \mathrm{f}'^{\mathrm{III,sring}}_\alpha\right] \\
\mathrm{f}'^{\mathrm{III},j}_\alpha &= \frac{\mathrm{f}^{\mathrm{III},j}_\alpha - \mu^{\mathrm{III,preset}}_j}{\sigma^{\mathrm{III,preset}}_j}, \\
j &\in \{\mathrm{atom}, \mathrm{bond}, \mathrm{catom}, \mathrm{cbond}, \mathrm{valence}, \mathrm{nbond}, \mathrm{sring}\}.
\end{aligned} \tag{5.30}$$

NN Implementation In the level-III angle graph the neural network is designed as a 5×100 fully-connected neural network. Here, the layer indexing l follows the same definition presented in Equation (4.12a). The number of neurons in each layer can be written as

$$\begin{cases}
q^{\mathrm{III}}_0 = 7, & \text{input layer,} \\
q^{\mathrm{III}}_l = 100, l \in \{1,2,3,4,5\}, & \text{hidden layers,} \\
q^{\mathrm{III}}_6 = 1, & \text{output layer.}
\end{cases} \tag{5.31}$$

There are a total of 41,301 trainable parameters in the network, and the activation function used in this neural network is also tanh. Thus, the mathematical form of the neural network reads

$$\begin{aligned}
\mathbf{G}^{\mathrm{III},0}_\alpha &= \mathbf{f}'^{\mathrm{III}}_\alpha && \text{input layer,} \\
\mathbf{G}^{\mathrm{III},l}_\alpha &= \tanh(\mathbf{W}^{\mathrm{III},l-1}\mathbf{G}^{\mathrm{III},l-1}_\alpha + \mathbf{b}^{\mathrm{III},l-1}) && \text{hidden layers, } l \in \{1,2,3,4,5\}, \text{ and} \\
\mathbf{y}'^{\mathrm{III}}_\alpha &= [\mathrm{y}'^{\mathrm{III}}_\alpha] = \sigma_{\mathrm{III}}\left(\mathbf{W}^{\mathrm{III},5}\mathbf{G}^{\mathrm{III},5}_\alpha + \mathbf{b}^{\mathrm{III},5}\right) && \text{output layer.}
\end{aligned} \tag{5.32}$$

Here, the σ_{III} is the extra activation function to clip the output angle in order to avoid invalid output, which is defined as

$$\sigma_{\mathrm{III}}(x) = \begin{cases}
\mathrm{norm}_{\mathrm{III}}(0), & \text{if } x < \mathrm{norm}_{\mathrm{III}}(0), \\
\mathrm{norm}_{\mathrm{III}}(180), & \text{if } x > \mathrm{norm}_{\mathrm{III}}(180), \\
x, & \text{otherwise,}
\end{cases} \tag{5.33}$$

where $\mathrm{norm}_{\mathrm{III}}$ denotes the normalisation process in level-III model (see Equation (5.30)). The batch size of the training process is 256 and the number of epochs is 100. Here, the optimiser is also Adam (with the same parameters set as NN in level-II).

NN Post-processing The denormalisation process for the level-III angle graph result is

$$\begin{aligned}
\mathbf{Y}_{\mathrm{III}} &= [\mathbf{y}^{\mathrm{III}}_1, \mathbf{y}^{\mathrm{III}}_2, \ldots, \mathbf{y}^{\mathrm{III}}_{n^{\mathrm{III}}}], \\
\mathbf{y}^{\mathrm{III}}_\alpha &= [\mathrm{y}^{\mathrm{III}}_\alpha], \\
\mathrm{y}^{\mathrm{III}}_\alpha &= \sigma^{\mathrm{III,preset}}_{\mathrm{y}} \cdot \mathrm{y}'^{\mathrm{III}}_\alpha + \mu^{\mathrm{III,preset}}_{\mathrm{y}}.
\end{aligned} \tag{5.34}$$

Torsion Graph (Level-IV)

In the MPF implementation of the level-IV torsion graph a filtering process is applied to ensure the model training and predicting are performed on constrained torsions only. A torsion is defined as constrained if it satisfies either of the following criteria:

- the central bond (i.e. the sharing bond of two planes) is not a single bond, or
- the central bond is part of ring.

For all non-constrained torsions (such as the torsions in ethane), the degree of torsion is set to 0 by default.

Atomic Constant Properties The atomic constant property set in the torsion graph is the same as that in the level-II bond graph (Equation (5.11)), which is

$$C^{\mathrm{IV}} = \{\mathbf{c}_\chi\}. \tag{5.35}$$

Node Feature Definition The definition of a feature in each node in the level-IV torsion graph is the same as for the previous graph levels (Equations (5.12) and (5.24)). The node feature vector in the level-IV torsion graph reads

$$\mathbf{h}_{t,\alpha}^{\mathrm{IV}} = \left[\mathrm{h}_{t,\alpha}^{\mathrm{IV,atom}}, \mathrm{h}_{t,\alpha}^{\mathrm{IV,bond}}\right]. \tag{5.36}$$

Initialisation Function The initial node features in the level-IV torsion graph is also the same as for the level-II and level-III graphs. The mathematical form of this initialisation process reads

$$\mathrm{h}_{0,\alpha}^{\mathrm{IV,atom}} = I'^{\mathrm{IV}}_{\mathrm{atom}}(v_\alpha^{\mathrm{IV}}) = \varphi^{\mathrm{IV}}_{\mathrm{atom},\alpha} = \sum_{i,j} \chi_i T^{\mathrm{I}}_{ij} \chi_j, \quad e^{\mathrm{I}}_{ij} \in \varepsilon^{\mathrm{IV}}_\alpha \tag{5.37a}$$

and

$$\mathrm{h}_{0,\alpha}^{\mathrm{IV,bond}} = I'^{\mathrm{IV}}_{\mathrm{bond}}(v_\alpha^{\mathrm{IV}}) = \varphi^{\mathrm{IV}}_{\mathrm{bond},\alpha} = \sum_{i,j} T^{\mathrm{I}}_{ij}, \quad e^{\mathrm{I}}_{ij} \in \varepsilon^{\mathrm{IV}}_\alpha. \tag{5.37b}$$

Message Generation In the level-IV torsion graph the message generation is similar to the process for previous levels. Two messages ($k^{\mathrm{IV}} = 2$) are generated as follow:

$$\mathbf{m}_{t+1,\alpha}^{\mathrm{IV}} = \left[\mathrm{m}_{t+1,\alpha}^{\mathrm{IV,node}}, \mathrm{m}_{t+1,\alpha}^{\mathrm{IV,edge}}\right], \tag{5.38a}$$

where the node contribution message is

$$\mathrm{m}_{t+1,\alpha}^{\mathrm{IV,node}} = M'_{\mathrm{node}}(P_t^{\mathrm{IV}}, \alpha) = \sum_i \mathrm{h}_{t,i}^{\mathrm{IV,atom}} \cdot A^{\mathrm{IV}}_{i\alpha} \tag{5.38b}$$

and the edge contribution message is

$$\mathrm{m}_{t+1,\alpha}^{\mathrm{IV,edge}} = M'_{\mathrm{edge}}(P_t^{\mathrm{IV}}, \alpha) = \sum_i \mathrm{h}_{t,i}^{\mathrm{IV,bond}} \cdot T_{i\alpha}^{\mathrm{IV}}. \tag{5.38c}$$

Graph Update The update process in the level-IV torsion graph is also similar to those in the previous levels. It reads

$$\mathrm{h}_{t+1,\alpha}^{\mathrm{IV,atom}} = U'^{\mathrm{IV}}_{\mathrm{atom}}(\mathbf{M}_{t+1}^{\mathrm{IV}}, \alpha) = \mathrm{m}_{t+1,\alpha}^{\mathrm{IV,node}} \tag{5.39a}$$

and

$$\mathrm{h}_{t+1,\alpha}^{\mathrm{IV,bond}} = U'^{\mathrm{IV}}_{\mathrm{bond}}(\mathbf{M}_{t+1}^{\mathrm{IV}}, \alpha) = \mathrm{m}_{t+1,\alpha}^{\mathrm{IV,edge}}. \tag{5.39b}$$

The torsion graph is again updated for 3 rounds ($m^{\mathrm{IV}} = 3$).

Feature Readout In the level-IV torsion graph, the accumulated node feature is also used as the source of the final feature, noted as

$$\begin{aligned} \mathbf{H}_{\mathrm{acc}}^{\mathrm{IV}} &= \left[\mathbf{h}_{\mathrm{acc},1}^{\mathrm{IV}}, \mathbf{h}_{\mathrm{acc},2}^{\mathrm{IV}}, \ldots, \mathbf{h}_{\mathrm{acc},n^{\mathrm{IV}}}^{\mathrm{IV}}\right] = \sum_{t=0}^{m^{\mathrm{IV}}} \frac{1}{t!} \mathbf{H}_t^{\mathrm{IV}}, \\ \mathbf{h}_{\mathrm{acc},i}^{\mathrm{IV}} &= \left[\mathrm{h}_{\mathrm{acc},i}^{\mathrm{IV,atom}}, \mathrm{h}_{\mathrm{acc},i}^{\mathrm{IV,bond}}\right]. \end{aligned} \tag{5.40}$$

There are 14 features ($s^{\mathrm{IV}} = 14$) read out for each node in the torsion graph. They are

$$\begin{aligned} \mathbf{f}_\alpha^{\mathrm{IV}} &= R'^{\mathrm{IV}}(P_m^{\mathrm{IV}}, \mathbf{H}_{\mathrm{acc}}^{\mathrm{IV}}, \mathbf{M}_m^{\mathrm{IV}}, \alpha) \\ &= \Big[\mathrm{f}_\alpha^{\mathrm{IV,atom}}, \mathrm{f}_\alpha^{\mathrm{IV,bond}}, \mathrm{f}_\alpha^{\mathrm{IV,catom}}, \mathrm{f}_\alpha^{\mathrm{IV,cbond}}, \\ &\quad \mathrm{f}_\alpha^{\mathrm{IV,corder}}, \mathrm{f}_\alpha^{\mathrm{IV,sring}}, \mathrm{f}_\alpha^{\mathrm{IV,cnum}}, \mathrm{f}_\alpha^{\mathrm{IV,maxctype}}, \\ &\quad \mathrm{f}_\alpha^{\mathrm{IV,minctype}}, \mathrm{f}_\alpha^{\mathrm{IV,maxbangle}}, \mathrm{f}_\alpha^{\mathrm{IV,minbangle}}, \\ &\quad \mathrm{f}_\alpha^{\mathrm{IV,maxblen}}, \mathrm{f}_\alpha^{\mathrm{IV,minblen}}, \mathrm{f}_\alpha^{\mathrm{IV,midblen}}\Big] \end{aligned} \tag{5.41a}$$

where the atom-related final feature is

$$\mathrm{f}_\alpha^{\mathrm{IV,atom}} = \mathrm{h}_{\mathrm{acc},\alpha}^{\mathrm{IV,atom}}, \tag{5.41b}$$

the bond-related final feature is

$$\mathrm{f}_\alpha^{\mathrm{IV,bond}} = \mathrm{h}_{\mathrm{acc},\alpha}^{\mathrm{IV,bond}}, \tag{5.41c}$$

the atom-related constant feature is

$$\mathrm{f}_\alpha^{\mathrm{IV,catom}} = \varphi_{\mathrm{atom},\alpha}^{\mathrm{IV}}, \tag{5.41d}$$

the bond-related constant feature is

$$\mathrm{f}_\alpha^{\mathrm{IV,cbond}} = \varphi_{\mathrm{bond},\alpha}^{\mathrm{IV}}, \tag{5.41e}$$

and the bond order of the central bond is

$$\begin{gathered}\mathrm{f}_{\alpha}^{\mathrm{IV,corder}} = T_{ij}^{\mathrm{I}}, \\ v_i^{\mathrm{I}}, v_j^{\mathrm{I}} \in v_p^{\mathrm{III}} \cap v_q^{\mathrm{III}}, \quad \nu_p^{\mathrm{III}}, \nu_q^{\mathrm{III}} \in \nu_{\alpha}^{\mathrm{IV}} \quad \text{and} \quad \varepsilon_p^{\mathrm{III}}, \varepsilon_q^{\mathrm{III}} \in \varepsilon_{\alpha}^{\mathrm{IV}}, \\ i \neq j \quad \text{and} \quad p \neq q.\end{gathered} \tag{5.41f}$$

The minimum size of the rings that the torsion is in is

$$\mathrm{f}_{\alpha}^{\mathrm{IV,sring}} = \begin{cases} r_{\alpha}^{\mathrm{IV}}, & \text{if torsion } \alpha \text{ is located in rings,} \\ 0, & \text{otherwise,} \end{cases} \tag{5.41g}$$

where r_{α}^{IV} denotes the minimum ring size. The number of connections for this node is

$$\mathrm{f}_{\alpha}^{\mathrm{IV,cnum}} = \sum_{i} A_{i\alpha}^{\mathrm{IV}}, \tag{5.41h}$$

the maximum value of the connection type is

$$\mathrm{f}_{\alpha}^{\mathrm{IV,maxctype}} = \max\left\{ T_{i\alpha}^{\mathrm{IV}} \middle| T_{i\alpha}^{\mathrm{IV}} \neq 0 \right\}, \tag{5.41i}$$

the minimum value of the connection type is

$$\mathrm{f}_{\alpha}^{\mathrm{IV,minctype}} = \min\left\{ T_{i\alpha}^{\mathrm{IV}} \middle| T_{i\alpha}^{\mathrm{IV}} \neq 0 \right\}, \tag{5.41j}$$

the maximum bond angle value in the torsion (according to the predicted value from the level-III angle graph) is

$$\begin{gathered}\mathrm{f}_{\alpha}^{\mathrm{IV,maxbangle}} = \max(\mathrm{y}_i^{\mathrm{III}}, \mathrm{y}_j^{\mathrm{III}}), \\ \nu_i^{\mathrm{III}}, \nu_j^{\mathrm{III}} \in \nu_{\alpha}^{\mathrm{IV}} \quad \text{and} \quad \varepsilon_i^{\mathrm{III}}, \varepsilon_j^{\mathrm{III}} \in \varepsilon_{\alpha}^{\mathrm{IV}}, \\ i \neq j,\end{gathered} \tag{5.41k}$$

the minimum bond angle value in the torsion (according to the predicted value from level-III angle graph) is

$$\begin{gathered}\mathrm{f}_{\alpha}^{\mathrm{IV,minbangle}} = \min(\mathrm{y}_i^{\mathrm{III}}, \mathrm{y}_j^{\mathrm{III}}), \\ \nu_i^{\mathrm{III}}, \nu_j^{\mathrm{III}} \in \nu_{\alpha}^{\mathrm{IV}} \quad \text{and} \quad \varepsilon_i^{\mathrm{III}}, \varepsilon_j^{\mathrm{III}} \in \varepsilon_{\alpha}^{\mathrm{IV}}, \\ i \neq j,\end{gathered} \tag{5.41l}$$

the maximum bond length value in the torsion (according to the predicted value from the level-II bond graph) is

$$\begin{gathered}\mathrm{f}_{\alpha}^{\mathrm{IV,maxblen}} = \max(\mathrm{y}_i^{\mathrm{II}}, \mathrm{y}_j^{\mathrm{II}}, \mathrm{y}_p^{\mathrm{II}}), \\ \nu_i^{\mathrm{II}}, \nu_j^{\mathrm{II}}, \nu_p^{\mathrm{II}} \in \nu_{\alpha}^{\mathrm{IV}} \quad \text{and} \quad \varepsilon_i^{\mathrm{II}}, \varepsilon_j^{\mathrm{II}}, \varepsilon_p^{\mathrm{II}} \in \varepsilon_{\alpha}^{\mathrm{IV}}, \\ i \neq j \neq p,\end{gathered} \tag{5.41m}$$

the minimum bond length value in the torsion (according to the predicted value from the

level-II bond graph) is

$$\begin{aligned} \mathrm{f}_{\alpha}^{\mathrm{IV,minblen}} &= \min(\mathrm{y}_i^{\mathrm{II}}, \mathrm{y}_j^{\mathrm{II}}, \mathrm{y}_p^{\mathrm{II}}), \\ \nu_i^{\mathrm{II}}, \nu_j^{\mathrm{II}}, \nu_p^{\mathrm{II}} \in \nu_{\alpha}^{\mathrm{IV}} \quad &\text{and} \quad \varepsilon_i^{\mathrm{II}}, \varepsilon_j^{\mathrm{II}}, \varepsilon_p^{\mathrm{II}} \in \varepsilon_{\alpha}^{\mathrm{IV}}, \\ i \neq j &\neq p, \end{aligned} \tag{5.41n}$$

and the medium bond length value in the torsion (according to the predicted value from level-II bond graph) is

$$\begin{aligned} \mathrm{f}_{\alpha}^{\mathrm{IV,midblen}} &= \mathrm{y}_i^{\mathrm{II}} + \mathrm{y}_j^{\mathrm{II}} + \mathrm{y}_p^{\mathrm{II}} - \max(\mathrm{y}_i^{\mathrm{II}}, \mathrm{y}_j^{\mathrm{II}}, \mathrm{y}_p^{\mathrm{II}}) - \min(\mathrm{y}_i^{\mathrm{II}}, \mathrm{y}_j^{\mathrm{II}}, \mathrm{y}_p^{\mathrm{II}}), \\ \nu_i^{\mathrm{II}}, \nu_j^{\mathrm{II}}, \nu_p^{\mathrm{II}} \in \nu_{\alpha}^{\mathrm{IV}} \quad &\text{and} \quad \varepsilon_i^{\mathrm{II}}, \varepsilon_j^{\mathrm{II}}, \varepsilon_p^{\mathrm{II}} \in \varepsilon_{\alpha}^{\mathrm{IV}}, \\ i \neq j &\neq p. \end{aligned} \tag{5.41o}$$

NN Preprocessing The normalisation method used in the level-IV torsion graph is also the same as that in previous layers. The normalisation for the final feature vector of node α reads

$$\begin{aligned} \mathrm{f}'^{\mathrm{IV},j}_{\alpha} &= \frac{\mathrm{f}_{\alpha}^{\mathrm{IV},j} - \mu_j^{\mathrm{IV,preset}}}{\sigma_j^{\mathrm{IV,preset}}}, \\ j \in \big\{&\text{atom, bond, catom, cbond, corder, sring, cnum, maxctype,} \\ &\text{minctype, maxbangle, minbangle, maxblen, minblen, midblen}\big\}. \end{aligned} \tag{5.42}$$

NN Implementation Here, a 5×100 fully-connected neural network is used for the torsion angle prediction. The layer index l in this neural network is defined the same as Equation (4.12a). The number of neurons in each layer for this neural network is

$$\begin{cases} q_0^{\mathrm{IV}} = 14, & \text{input layer,} \\ q_l^{\mathrm{IV}} = 100, l \in \{1,2,3,4,5\}, & \text{hidden layers,} \\ q_6^{\mathrm{IV}} = 1, & \text{output layer.} \end{cases} \tag{5.43}$$

In this neural network, there are 42,001 trainable parameters in the model. Here, tanh is used as the activation function for hidden layers. Therefore, the neural network can be written as

$$\begin{aligned} \mathbf{G}_{\alpha}^{\mathrm{IV},0} &= \mathbf{f}'^{\mathrm{IV}}_{\alpha} && \text{input layer,} \\ \mathbf{G}_{\alpha}^{\mathrm{IV},l} &= \tanh(\mathbf{W}^{\mathrm{IV},l-1}\mathbf{G}_{\alpha}^{\mathrm{IV},l-1} + \mathbf{b}^{\mathrm{IV},l-1}) && \text{hidden layers, } l \in \{1,2,3,4,5\}, \text{ and} \\ \mathbf{y}'^{\mathrm{IV}}_{\alpha} &= [\mathrm{y}'^{\mathrm{IV}}_{\alpha}] = \sigma_{\mathrm{IV}}\left(\mathbf{W}^{\mathrm{IV},5}\mathbf{G}_{\alpha}^{\mathrm{IV},5} + \mathbf{b}^{\mathrm{IV},5}\right) && \text{output layer.} \end{aligned} \tag{5.44}$$

Here, σ_{IV} is the extra activation function to constraint the output angle to avoid invalid output angle, and is defined as

$$\sigma_{\mathrm{IV}}(x) = \begin{cases} \mathrm{norm}_{\mathrm{IV}}(180), & \text{if } x > \mathrm{norm}_{\mathrm{IV}}(180), \\ \mathrm{norm}_{\mathrm{IV}}(0), & \text{if } x < \mathrm{norm}_{\mathrm{IV}}(0), \\ x, & \text{otherwise.} \end{cases} \tag{5.45}$$

Here, $\mathrm{norm}_{\mathrm{IV}}$ denotes the normalisation process (see Equation (5.42)) in the level-IV NN

model. Although the torsion angles range from $-180°$ to $180°$ by definition, I here introduced this activation function to ensure the output range is $0° \sim 180°$ to eliminate the sign ambiguity due to atom ordering. The model is trained for 100 epoch with a batch size of 256. The optimiser and relative setting are the same as previous levels.

NN Post-processing The denormalisation process for the level-IV torsion graph reads

$$\begin{aligned} \mathbf{Y}_{\mathrm{IV}} &= [\mathbf{y}_1^{\mathrm{IV}}, \mathbf{y}_2^{\mathrm{IV}}, \ldots, \mathbf{y}_{n^{\mathrm{IV}}}^{\mathrm{IV}}], \\ \mathbf{y}_\alpha^{\mathrm{IV}} &= [\mathrm{y}_\alpha^{\mathrm{IV}}], \\ \mathrm{y}_\alpha^{\mathrm{IV}} &= \sigma_{\mathrm{y}}^{\mathrm{IV,preset}} \cdot \mathrm{y}'^{\mathrm{IV}}_\alpha + \mu_{\mathrm{y}}^{\mathrm{IV,preset}}. \end{aligned} \tag{5.46}$$

5.2 Results and Discussion on Accuracy

In project MSP, three independent models are trained based on dataset "Struct_Train", and they are tested on both GDB-based "Struct_Test" and ChEMBLE-based "LM_Struct_Test". In project MSP, each model contains three sub-models for bond length, bond angle and torsion angle, namely "BG", "AG" and "TG" respectively. In the following sections, I present the evaluation of each model in the order of BG, AG and TG.

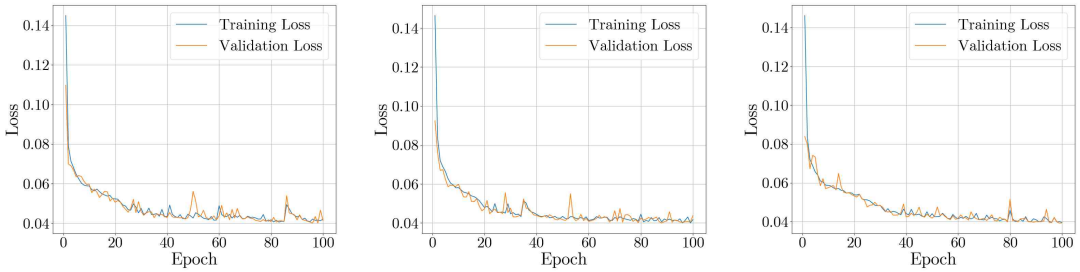
For each sub-model (BG, AG or TG), a training loss analysis is first carried out, in order to evaluate the training performance. This can tell us how well the models are trained and whether they are valid. For each testing dataset three types of figure are presented, including the heatmaps (predicted value vs. true value), error histograms and absolute error histograms with cumulative percentage. By analysis of these three types of figure the general accuracy of the models can be obtained. In addition, the quantitative analysis is carried for detailed inspection on the accuracy of the models. Due to the distribution of the value of bond angle and torsion angle, where most of the data locate around zero in project MSP, I only carried out the metrics based on absolute error in order to avoid dividing by zero. Finally, a thorough molecular geometry assessment is conducted to examine the overall prediction accuracy, determine the underlying causes of errors, and provide insights for refining the algorithm in future studies.

The training and testing input files for "prj_acdm" are provided in Appendix M. Please note that the training on the semi-empirical dataset (ZINC-based) was terminated by the system watchdog due to memory overflow, with the last recorded memory usage reaching approximately 78 GB.

5.2.1 Evaluation of Bond Length (BG) Prediction

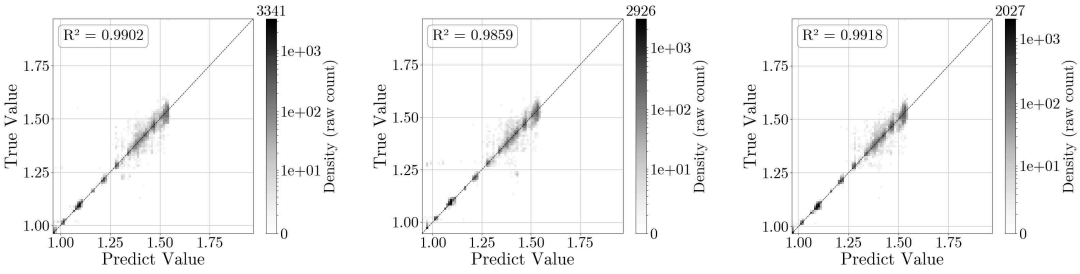
Training Analysis

The variation of the training and validation loss with epoch of BG models during training is provided in Figure 5.3. From these three plots it can be found that both the training and validation losses decrease with increasing epoch. In addition, after 100 epochs of training both the training and validation losses approach convergence, and the plots show



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.3: The variation of training loss and validation loss with epoch for sub-model BG for "Model_Struct" during training.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.4: Heatmaps analysis for sub-model BG of "Model_Struct" on dataset "Struct_Test", with a maximum value of the density given on top of the colour bar. The unit of the predicted and true values are angstrom (Å).

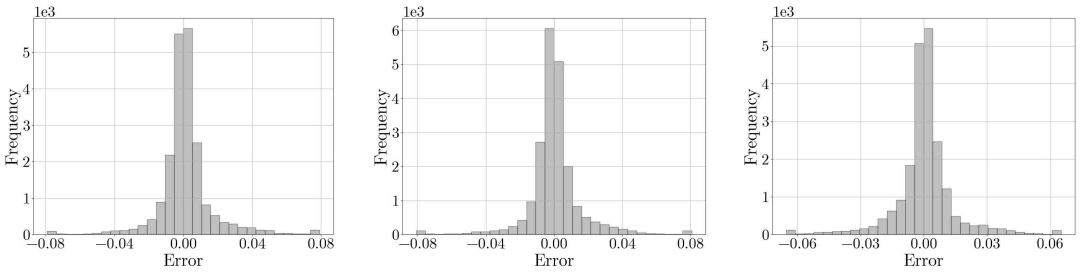
no evidence of over-fitting. Thus, these three BG models are valid for the following tests.

Testing on "Struct_Test"

The heatmaps of the predicted value against the true value for sub-model BG are presented in Figure 5.4. These heatmaps show that the data points are located close to the diagonal direction, indicating a strong correlation between the predicted value and true value. Moreover, the R-squared value of the sub-model BG of all three models is greater than 0.985. This shows that the GLaCE- and MPF-based model can predict the bond length with good accuracy.

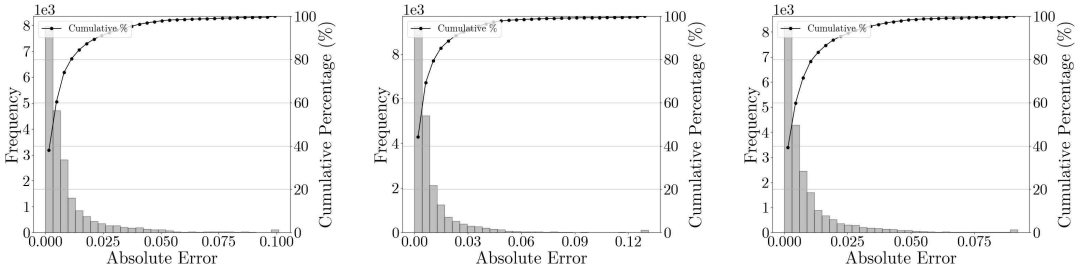
Figure 5.5 gives the histograms of the error distribution. From that histogram it can be found that the distribution of error is relatively narrow, compared to the range of value of bond length (0.9Å$\sim$ 2Å). The peaks of the distributions are located precisely at zero. In addition, from Figure 5.6, which shows the distribution of the absolute errors, it can be observed that about 80% of the data have errors less than 0.015Å. The above analysis on error distributions shows that the sub-model BG can predict the bond length with high accuracy.

Table 5.2 gives the value of metrics that enable us to carry out a quantitative analysis on the accuracy of prediction on bond length. Here, the average mean absolute error (MAE) for three models is 0.0097Å, while the average median absolute error for bond



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.5: Error histograms for sub-model BG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is angstrom (Å).

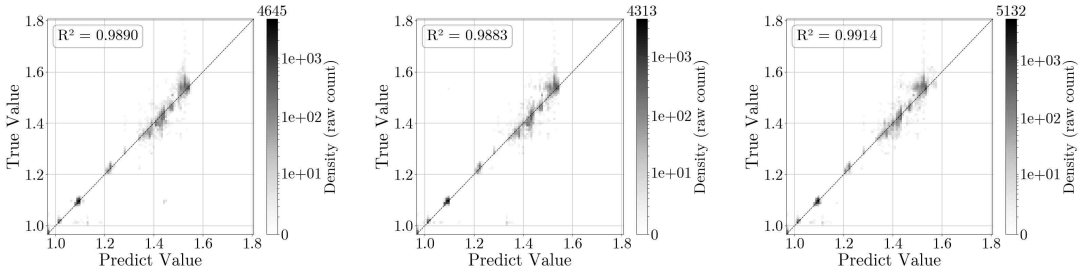


(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.6: Absolute error histograms for sub-model BG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is angstrom (Å).

Table 5.2: The value of metrics for test on dataset "Struct_Test" for the sub-model BG of three "Model_Struct" models. The acronym TL80AE is short for the top low 80% absolute error.

	MAE (Å)	**MedAE (Å)**	**TL80AE (Å)**	**RMSE (Å)**	R^2
Average	0.0097	0.0047	0.0128	0.0193	0.9893
Model 1	0.0097	0.0047	0.0128	0.0187	0.9902
Model 2	0.0103	0.0050	0.0131	0.0223	0.9859
Model 3	0.0090	0.0043	0.0125	0.0170	0.9918



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.7: Heatmaps analysis for sub-model BG of "Model_Struct" on dataset "LM_Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are angstrom (Å).

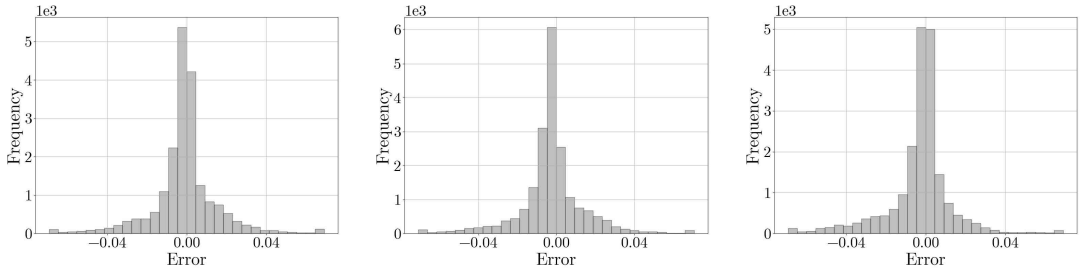
length prediction is only 0.0047Å. These two values show that the accuracy of sub-model BG is satisfactory. Moreover, the values of TL80AE and RMSE are 0.0128Å and 0.0193Å, respectively. By comparing to the range of bond length (0.9 $\sim$ 2Å), the proportion of the error on 80% of data is roughly 0.6% $\sim$ 1.4%. The above evaluation indicates that the GLaCE- and MPF-based strategy has good performance in predicting bond length on GDB-based testing set.

Testing on "LM_Struct_Test"

These models are also tested on a larger molecule dataset "LM_Struct_Test". Figure 5.7 gives the heatmaps of three trained models for bond length prediction. From these heatmaps I can be observed that the data points lie close to the diagonal direction, indicating a good accuracy in bond length prediction. According to the R-squared values (larger than 0.988) of these three BG sub-models, the predicted result has a strong correlation with the true value, showing a satisfactory accuracy of the BG sub-models when making predictions on larger molecules.

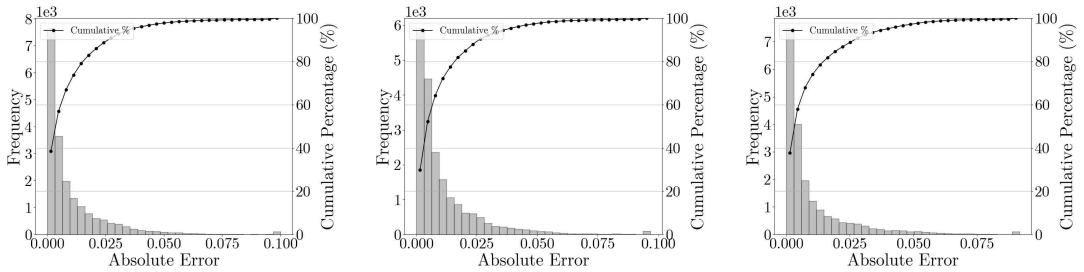
Figure 5.8 presents the error distribution of these BG sub-models when predicting bond length for dataset "LM_Struct_Test". Here, a symmetrical distribution can be observed in each histogram, centring on zero precisely. Compared to Figure 5.5, the distribution of errors given in Figure 5.8 is slightly broader, indicating a minor loss in the predicted accuracy.

Figure 5.9 gives the histogram of the absolute error of the predicted results for dataset



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.8: Error histograms for sub-model BG of "Model_Struct" on dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is angstrom (Å).



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.9: Absolute error histograms for sub-model BG of "Model_Struct" for dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is angstrom (Å).

"LM_Struct_Test". From these histograms it can be found that the error distributions have a similar decrease rate in frequency as the error increases compared to the predicted results for dataset "Strcut_Test". Additionally, the cumulative percentage curve indicates that the errors of 90% of the data are less than 0.025Å. Further analysis is given below.

Table 5.3 presents the metrics used to evaluate the performance for larger molecule datasets. The average MAE of the predicted results for dataset "LM_Struct_Test" is 0.0109Å, slightly higher than that for dataset "Struct_Test" (0.0097Å). And the MedAE and RMSE values also marginally increase, while the value of TL80AE (0.0168Å) increases significantly compared to 0.0128Å(about a 30% increase). This quantitative analysis result matches the broader error distribution observed in previous figures.

Thus, it can be concluded that the GLaCE- and MPF-based models perform well in bond length prediction.

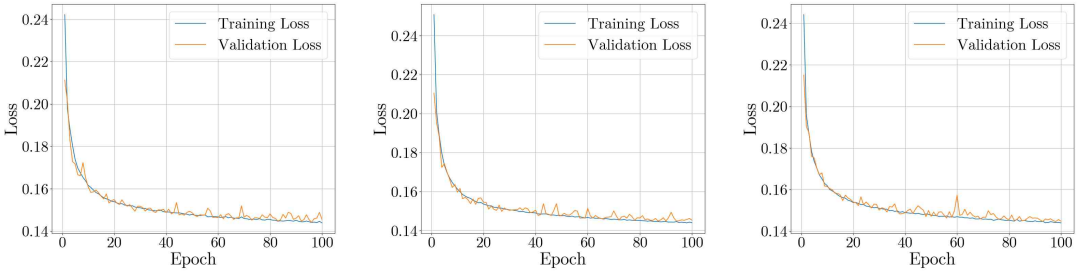
5.2.2 Evaluation of Bond Angle (AG) Prediction

Training Analysis

The curves of training and validation loss of the AG sub-model is presented in Figure 5.10. These three plots show that, during the training process, the training loss and validation

Table 5.3: The value of metrics for test on dataset "LM_Struct_Test" for the sub-model BG of three "Model_Struct" models. The TL80AE is short for the top low 80% absolute error.

	MAE (Å)	**MedAE (Å)**	**TL80AE (Å)**	**RMSE (Å)**	R^2
Average	0.0109	0.0051	0.0168	0.0204	0.9896
Model 1	0.0109	0.0049	0.0170	0.0210	0.9890
Model 2	0.0115	0.0058	0.0174	0.0217	0.9883
Model 3	0.0104	0.0045	0.0161	0.0186	0.9914



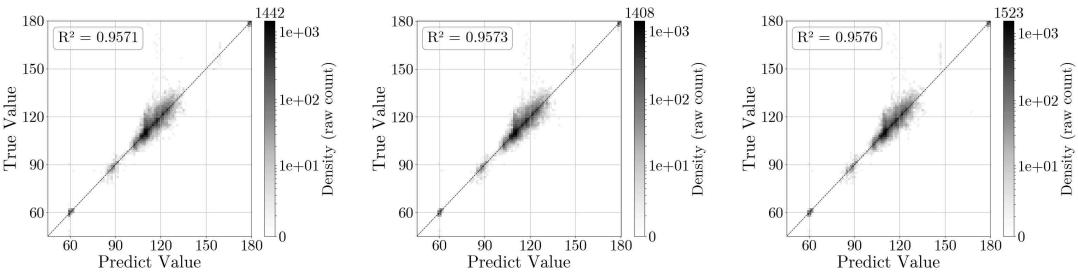
(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.10: The training loss and validation loss of sub-model AG of "Model_Struct" during training epochs.

loss both decrease as the number of epochs increases. In addition, both curves has no large gaps in between, indicating that there is no obvious over-fitting behaviour. Therefore, the trained AG sub-models are suitable for tests.

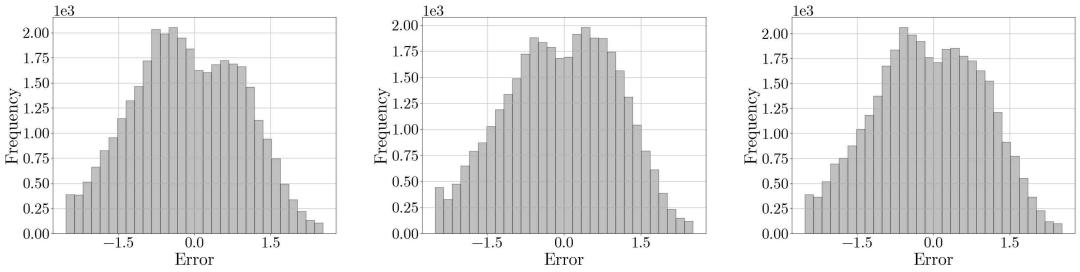
Testing on "Struct_Test"

The heatmaps of the predicted bond angle against the DFT true value for dataset "Struct_Test" are provided in Figure 5.11. It can be observed that, the data points are mostly located in the range of $[100°, 135°]$ (inclusive), with three small clusters of data points around 60°, 90° and 180°. These heatmaps show that, the predicted bond angles have strong correlation with the true values. In addition, the R-squared value of these three AG sub-models are all above 0.95, supporting the observed feature of strong correlation.



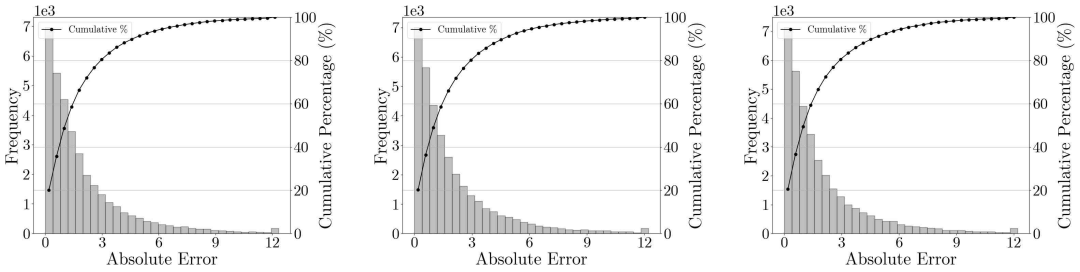
(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.11: Heatmaps analysis for sub-model AG of "Model_Struct" for dataset "Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values is degrees.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.12: Error histograms for sub-model AG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.13: Absolute error histograms for sub-model AG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees.

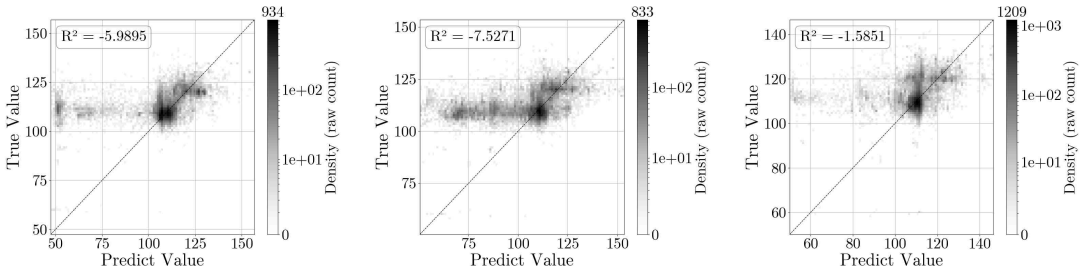
The error histograms are given in Figure 5.12. Here, broad distributions of error can be observed, with two slightly elevated peaks lying around zero. This phenomenon indicates that the GLaCE- and MPF- models have both over- and under-estimations of bond angles. Such a pattern may suggest that the bond angles in the testing set "Struct_Test" consists of two distinct subgroups, or systematic differences in the model performance across different molecule types or property range.

From the absolute error histograms shown in Figure 5.13, the absolute error distributions have relatively long tails. However, from the ticks on the x-axis, the value of absolute error is relatively small, compared to the distribution range of the bond angle ($50° \sim 180°$). Nearly 80% of the predicted data have errors of 3° or less, which is only $1.7\% \sim 6\%$. This shows that, although the models may not have enough knowledge of the physical origin of bond angles, the errors are still relatively small and acceptable.

The metrics shown in Table 5.4 also indicate that the predict results are relatively accurate. The average MAE and MedAE values of the AG sub-models are only 1.9933° and 1.2227° respectively, with a value of TL80AE of 3.1233°. These metrics provide quantitative evidence that support the high accuracy of the models in most cases.

Table 5.4: The value of metrics for tests on dataset "Struct_Test" for the sub-model AG for three "Model_Struct" models. The acronym TL80AE is short for the top low 80% absolute error.

	MAE (°)	**MedAE (°)**	**TL80AE (°)**	**RMSE (°)**	R^2
Average	1.9933	1.2227	3.1233	3.0984	0.9573
Model 1	2.0086	1.2442	3.1344	3.1078	0.9571
Model 2	1.9960	1.2207	3.1446	3.0986	0.9573
Model 3	1.9753	1.2032	3.0908	3.0887	0.9576



(a) Model_1 (b) Model_2 (c) Model_3

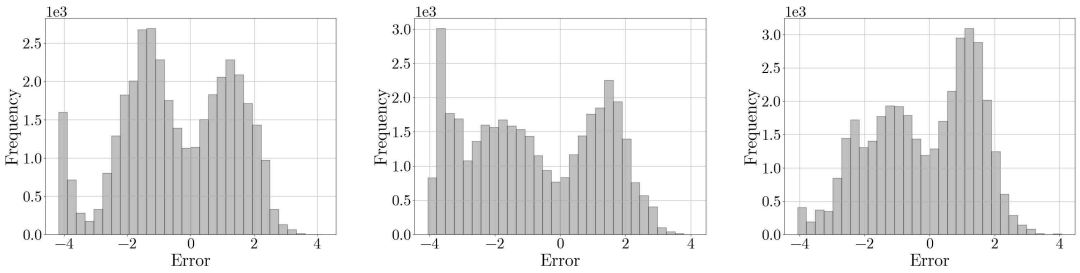
Figure 5.14: Heatmaps analysis for sub-model AG of "Model_Struct" on dataset "LM_Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values is degrees.

Testing on "LM_Struct_Test"

The testing on larger molecule dataset "LM_Struct_Test" shows several limitations of these GLaCE- and MPF-based models. From the heatmaps provided in Figure 5.14, a clear deviation from the diagonal line can be observed. Here, a horizontal distribution of the data points can be observed, indicating that the AG sub-models fail to predict bond angles correctly in most cases, especially for "Model_2". Moreover, the R-squared values of these predictions is less than zero, demonstrating that the predicted values and true values have little correlation.

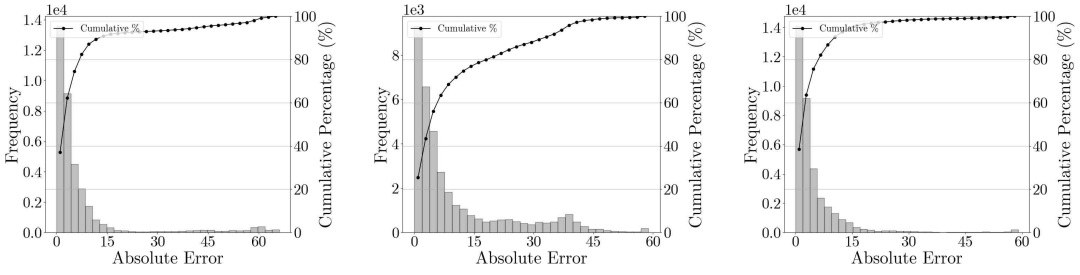
The error distribution presented in Figure 5.15 confirms the pattern observed in the heatmaps. "Model_1" and "Model_3" show relatively symmetric distributions of error with two clear peaks, while the errors of "Model_2" are distributed asymmetrically. These observations prove that the AG sub-models cannot predict bond angle for larger molecules accurately. From Figure 5.16 the distributions of the absolute errors have long tails, extending to over 60°. In addition, the performance of these three models varies significantly. "Model_2" has longer and broader tails compared to the other two models. "Model_1" and "Model_3" have the better accuracy among these three tested models, with round 80% of data having less than 8° of absolute error.

A quantitative analysis is carried out for further proof of the performance of AG sub-models, with the table of metrics given in Table 5.5. From the table, the average values for MAE (7.6032°), TL80AE (11.3117°) and RMSE (13.5515°) are significantly larger than those from the tests on the GDB-based database "Struct_Test", where the average



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.15: Error histograms for sub-model AG of "Model_Struct" on dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees.

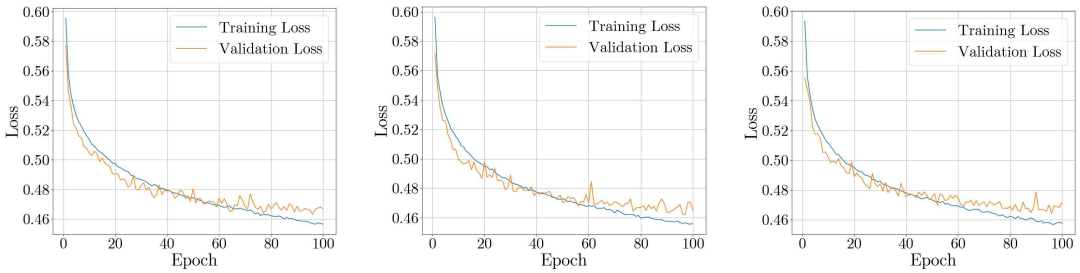


(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.16: Absolute error histograms for sub-model AG of "Model_Struct" for dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees.

Table 5.5: The value of metrics for test on dataset "LM_Struct_Test" for the sub-model AG of three "Model_Struct" models. The TL80AE is short for the top low 80% absolute error.

	MAE (°)	**MedAE (°)**	**TL80AE (°)**	**RMSE (°)**	R^2
Average	7.6032	3.4575	11.3117	13.5515	-5.0339
Model 1	7.3416	3.0582	7.7345	14.9868	-5.9895
Model 2	10.4847	4.6885	19.2784	16.5534	-7.5271
Model 3	4.9833	2.6256	6.9221	9.1143	-1.5851



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.17: The training loss and validation loss of sub-model TG of "Model_Struct" during training epochs.

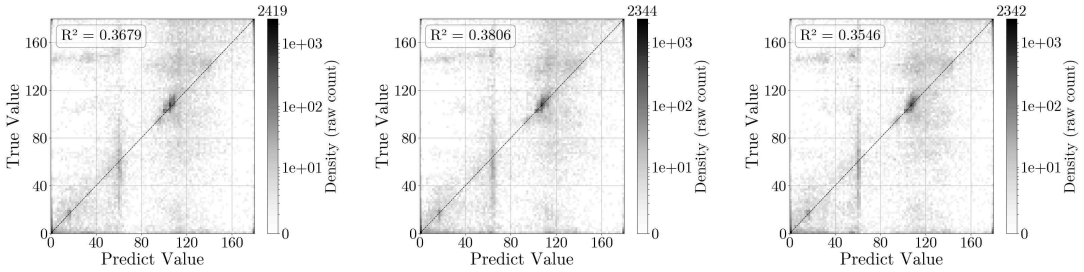
MAE, TL80AE and RMSE values are 1.9933°, 3.1233° and 3.0984° respectively. This also matches the pattern of long tails observed in previous error histograms.

It can be concluded that, the performance of AG sub-models in the larger molecule dataset "LM_Struct_Test" are significantly less reliable than the test on "Struct_Test". This indicates that the GLaCE- and MPF-based models has important limitations when tested on the dataset with larger molecules (compared to the size of molecules in the training set). The reason for such limitations might be the increasing complexity of the factors that affect the value of bond angle in larger and longer molecules, caused by the long range interaction between atoms and secondary bonding (like hydrogen bond, van der Waals force, etc.)

5.2.3 Evaluation of Torsion Angle (TG) Prediction

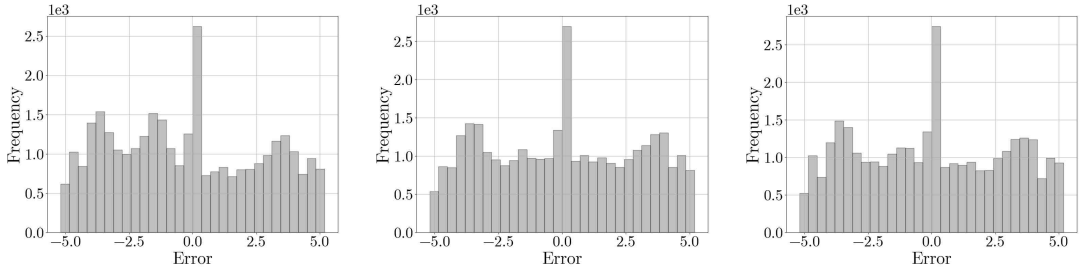
Training Analysis

The curves of training and validation loss of the TG sub-models are provided in Figure 5.17. From these plots, both the training and validation loss decrease during the training process, approaching convergence after 80 epochs. In addition, there is no obvious sign of over-fitting problems from these curves. Therefore, these models are well trained and valid for the following tests.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.18: Heatmaps analysis for sub-model TG of "Model_Struct" on dataset "Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values is degrees.



(a) Model_1 (b) Model_2 (c) Model_3

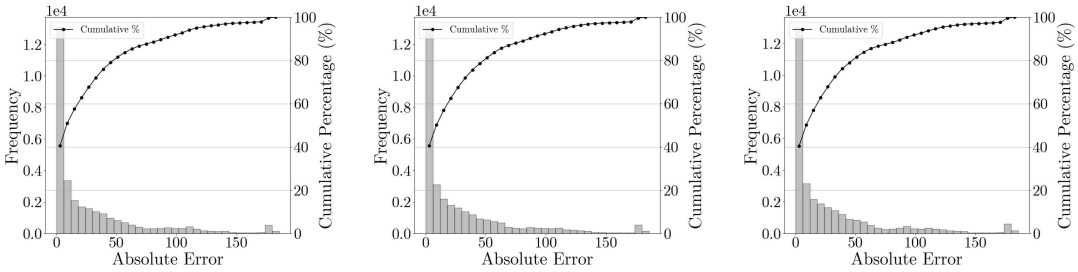
Figure 5.19: Error histograms for sub-model TG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees.

Testing on "Struct_Test"

The heatmaps for the TG sub-model prediction are presented in Figure 5.18. Here, a dispersed distribution of data points can be observed, with three clusters of data around 0°, 100° and 180°. This pattern indicates that the model fails to adapt to the complex features and environment of torsion angles. But for the torsions around 0°, 100° and 180°, the models can predict some of the cases correctly. In addition, the R-squared values of these three models are around $0.35 \sim 0.4$, indicating the correlation between the true value and predicted value is weak.

The error histograms of the TG sub-model predictions are given in Figure 5.19. The distribution of errors of these three models are even and broad, with a sharp peak locating around zero. This shows that the models can predict torsion angles correctly for some specific situations, with nearly random prediction for other cases. From the absolute error histograms shown in Figure 5.20, it can be found that the distributions have a sharp peak which decrease rapidly, and the long tails extend to nearly 180°. This observation indicates that the models have relatively high accuracy for specific cases, but lack in robustness.

The evaluation metrics of the TG sub-models tested on dataset "Struct_Test" are presented in Table 5.6. The average MAE is 29.7784°, showing that the overall accuracy of the TG sub-models is disappointing. Based on the MedAE (i.e. TL50AE) and TL80AE values it



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.20: Absolute error histograms for sub-model TG of "Model_Struct" on dataset "Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees.

Table 5.6: The value of metrics for test on dataset "Struct_Test" for the sub-model TG of three "Model_Struct" models. The acronym TL80AE is short for the top low 80% absolute error.

	MAE (°)	**MedAE (°)**	**TL80AE (°)**	**RMSE (°)**	R^2
Average	29.7784	11.6574	50.7048	50.6823	0.3677
Model 1	29.7681	11.2482	50.2926	50.6773	0.3679
Model 2	29.5969	11.8587	51.3126	50.1623	0.3806
Model 3	29.9702	11.8654	50.5092	51.2071	0.3546

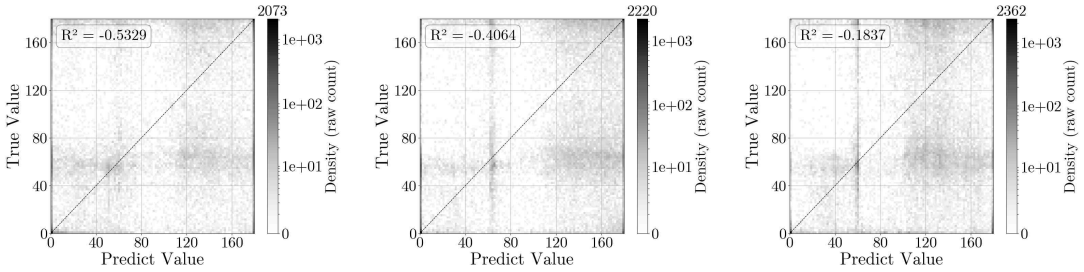
can be observed that the models can predict half of the situation well, but the accuracy decreases rapidly in other situations.

The above observation indicates that the presented models lack robustness, resulting in a poorly predicted torsion angles. The predominant contributing factor can be the conformational problem. Since only one conformation is given in the training set (see "Open Babel Conversion and MD Optimisation" part of Section 3.1.3), the models are trained without the information of different conformational structure from the same SMILES. Moreover, in longer and larger molecules, the number of possible conformations can be enormous, complicating the prediction of torsion angle. These two reasons can greatly reduce the reliability of the predicted results from the models, which can explain the above observation.

Testing on "LM_Struct_Test"

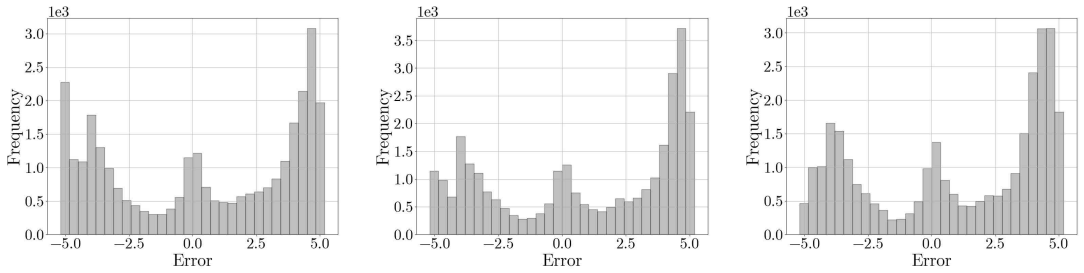
The heatmaps of the predicted values for "LM_Struct_Test" against the true values are provided in Figure 5.21. According to the heatmaps the distribution of data points are largely scattered, with two small clusters lying around 0° and 180°. In addition, both vertical and horizontal distributions can be observed at around 60° on the x-axis and y-axis respectively, suggesting systematic errors and a lack of robustness. This pattern shows that the TG sub-models fail to accurately predict torsion angle in larger molecules.

From the error histograms provided in Figure 5.22, the distribution of the errors exhibit a three peaks pattern, a relatively small peak lies around zero, and two higher peaks



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.21: Heatmaps analysis for sub-model TG of "Model_Struct" on dataset "LM_Struct_Test", with a maximum value of density given on top of the colour bar. The unit of the predicted and true values are degree.



(a) Model_1 (b) Model_2 (c) Model_3

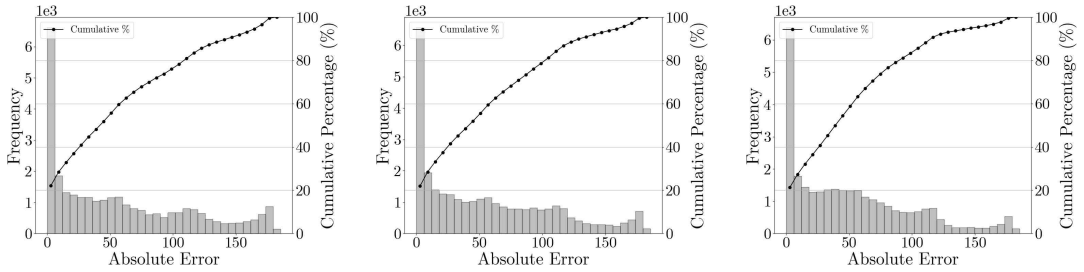
Figure 5.22: Error histograms for sub-model TG of "Model_Struct" on dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees.

locate on far left and right. This shows that the TG sub-models have large magnitude prediction errors, and the predicted results can only match the DFT true values in a few cases. According to the absolute error histograms given in Figure 5.23, the errors distribute evenly across the available range of values, except for a sharp peak at zero. This characteristic matches the previous observation from the heatmaps and error histograms, showing that the TG sub-models lack generalisation and robustness, particularly when applied to larger molecules.

The quantitative analysis also reflects the same problem of the TG sub-model. From the table of metrics given in Table 5.7, the MAE of the results for dataset "LM_Struct_Test" is 55.5091°, while the MedAE and TL80AE values are 43.8989° and 103.3795° respectively. In addition, the R-squared values of all tested models are negative, indicating little correlation between the true value and predicted value. It can be summarised that, this GLaCE- and MPF-based models are unreliable when predicting torsion angles in larger molecule dataset.

5.3 Comprehensive Molecular Geometry Assessment

A comprehensive evaluation of molecular structure prediction accuracy requires both quantitative and qualitative assessments. Using Gaussian Z-Matrix files — which encode bond



(a) Model_1 (b) Model_2 (c) Model_3

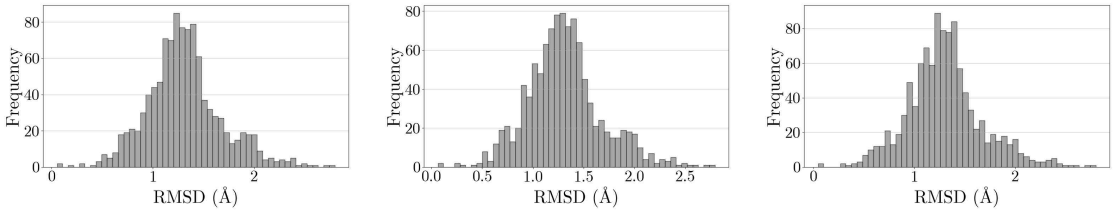
Figure 5.23: Absolute error histograms for sub-model TG of "Model_Struct" on dataset "LM_Struct_Test", with a frequency unit given on the top of the y-axis. Overflow bins are applied to the 0.5% extreme data. The unit of the error (x-axis) is degrees.

Table 5.7: The value of metrics for tests on dataset "LM_Struct_Test" for the sub-model TG of three "Model_Struct" models. The acronym TL80AE is short for the top low 80% absolute error.

	MAE (°)	**MedAE (°)**	**TL80AE (°)**	**RMSE (°)**	R^2
Average	55.5091	43.8989	103.3795	75.4161	-0.3743
Model 1	58.4706	45.0433	112.0450	79.7606	-0.5329
Model 2	56.2600	44.5757	105.2590	76.3982	-0.4064
Model 3	51.7967	42.0778	92.8346	70.0895	-0.1837

lengths, bond angles, and torsion angles — I compute the root-mean-square deviation (RMSD) between predicted structures and their corresponding DFT-optimized reference geometries. To complement this metric, I visually inspect representative cases to identify systematic deviations and assess overall prediction quality. Initial analysis revealed significant discrepancies in bond and torsion angle predictions for the "LM_Struct_Test" dataset; consequently, all subsequent validation focuses exclusively on the more reliable "Struct_Test" dataset. This selective approach enhances the robustness of algorithmic validation while enabling targeted investigation of prediction errors.

The conversion of predicted geometric parameters into Gaussian Z-Matrix format follows a two-step procedure: (1) Open Babel generates a template Z-Matrix using atomic connectivity data from `.mol2` files, and (2) a custom Python script (provided in Appendix N) populates this template with predicted values before converting it to Cartesian coordinates (`.xyz` format). As the models predict angle values exclusively for constrained torsions, the Python script automatically assigns a default value of 0° to any unpredicted torsion angles. This ensures complete Z-matrix specification while maintaining consistency with the model's partial prediction capability. For quantitative benchmarking, RDKit[114] calculates RMSD values between predicted and reference structures across all molecules in the test set, with comparisons made for three model variants (Model_1, Model_2, and Model_3). The Python script for RMSD calculation is provided in Appendix O.



(a) Model_1 (b) Model_2 (c) Model_3

Figure 5.24: The histogram of root-mean-square deviation of Model_1, Model_2 and Model_3 tested on dataset "Struct_Test".

Table 5.8: Statistical analysis of RMSD values (Å) for molecular structure predictions across three models evaluated on the "Struct_Test" dataset

Model	Mean	Median	Std. Dev.	Min	Max
Model 1	1.31222	1.28727	0.37219	0.05499	2.79892
Model 2	1.31084	1.28958	0.37211	0.06504	2.79056
Model 3	1.30279	1.28182	0.37186	0.04832	2.80091

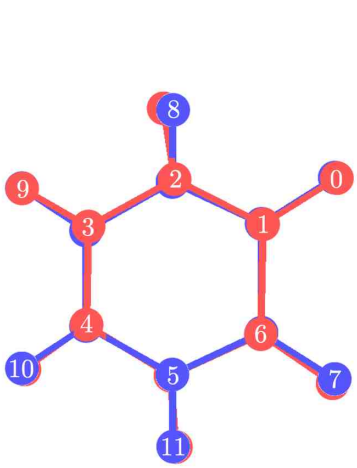
5.3.1 RMSD-Based Evaluation of Predicted Molecular Structures

Figure 5.24 displays histograms of RMSD values for individual molecules predicted by Model_1, Model_2, and Model_3. All three models exhibit similar distributions, with peaks centred at approximately 1.3 Å and broad ranges spanning 0.2–2.7 Å. These results indicate that while the models occasionally produce accurate predictions (RMSD < 0.5 Å), they typically generate structures with significant deviations from the DFT reference geometries. The consistent distribution patterns across all three models suggest systematic errors in the GLaCE algorithm that require further investigation and refinement.

Table 5.8 presents the RMSD evaluation metrics for the three proposed models. All models exhibit consistent performance with median RMSD values of approximately 1.28 Å and mean values around 1.31 Å, accompanied by a standard deviation of $\sim$0.37 Å. This consistency across independent training runs indicates robust algorithmic stability. However, the magnitude of these RMSD values (median: 1.28 Å, mean: 1.31 Å) suggests limited predictive accuracy relative to the DFT reference structures. While minimum RMSD values as low as 0.05 Å demonstrate near-perfect alignment in select cases, maximum deviations approaching 2.80 Å indicate substantial errors in certain predictions. The significant discrepancy between minimum and maximum RMSD values highlights inconsistent model performance that warrants further investigation into failure modes and error sources.

5.3.2 Visual Analysis of Representative Molecular Structures

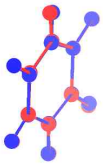
For visual analysis of molecular structures, I focus on Model_1 predictions due to the consistent performance observed across all three models. To represent the full spectrum of prediction quality, five representative cases were selected by sorting the RMSD distribution in ascending order and sampling at key percentiles:



(a) Front view



(b) Left view



(c) Free view

Index	0	1	2	3	4	5	6	7	8–11
Element	O	C	N	C	C	N	C	O	H

(d) List of atoms

Figure 5.25: Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_15016`. The atomic labels (white digits) are shown only for the reference structure due to the close alignment between both configurations.

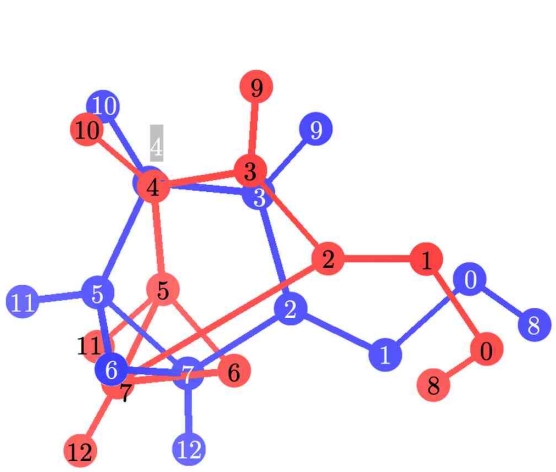
- **Minimum RMSD (0.0550 Å)**: Molecule `cno8_15016` (best prediction)
- **25th Percentile (1.0871 Å)**: Molecule `cno8_16580`
- **Median (1.2873 Å)**: Molecule `cno8_16399`
- **75th Percentile (1.4847 Å)**: Molecule `cno8_16261`
- **Maximum RMSD (2.7989 Å)**: Molecule `cno8_15931` (worst prediction)

This stratified sampling captures the complete error profile, from near-perfect alignment to significant deviations, enabling comprehensive structural analysis. These molecules (Figures 5.25 to 5.29) are visualised by software VMD[115].

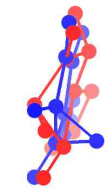
The first result demonstrates GLaCE's best predictive performance. Figure 5.25 compares the DFT reference structure (blue) with the ML-predicted structure (red). As shown in Figure 5.25(a), the predicted geometry of molecule `cno8_15016` closely matches the reference DFT structure, with only a minor positional deviation observed for atom 8.

Figure 5.26 displays molecule `cno_16580`, which falls in the 25th percentile of RMSD values in the test set. Figure 5.26(a) reveals two key discrepancies:

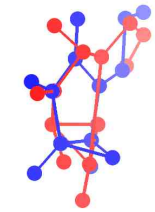
1. The branch containing atoms 0, 1, and 8 curves in the opposite direction relative to



(a) Front view



(b) Left view



(c) Free view

Index	0	1	2	3	4	5	6	7	8–12
Element	O	N	C	C	C	C	O	C	H

(d) List of atoms

Figure 5.26: Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_16580`. The atomic labels are shown for the reference structure (white digits) and the ML-predicted structure (black digits).

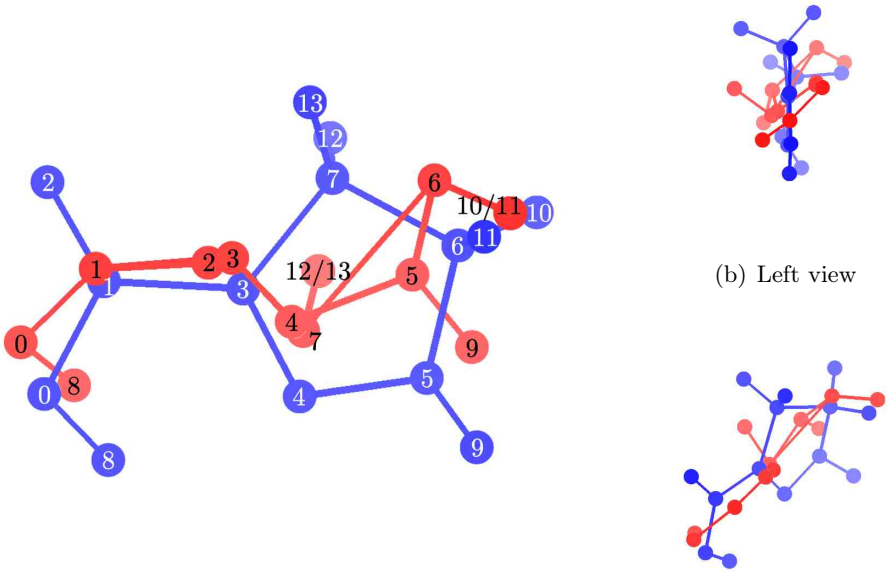
the DFT reference.

2. The 5-atom ring (atoms 2–5 and 7) exhibits a twisted geometry in the predicted structure.

Further analysis suggests that atoms 6 and 7 are positionally swapped in the prediction. These errors likely arise from a torsion angle misassignment in the Gaussian ZMatrix, where the model confuses 0° and 180° orientations, resulting in a 180° flip of these fragments. This observation is further supported by re-examining Figure 5.18, where distinct clustering patterns emerge in the top-left and bottom-right regions of the heatmaps. These clusters indicate systematic model confusion between 0° and 180° torsion angle assignments, with the model frequently predicting one configuration when the other is correct. Additionally, the arbitrary assignment of unpredicted torsion angles to 0° may exacerbate this issue, introducing further systematic bias into the model's predictions.

The molecule at the 50th percentile of the RMSD distribution is `cno8_16399`, visualised in Figure 5.27. Figure 5.27(a) reveals several discrepancies between the ML-predicted (red) and DFT reference (blue) structures:

1. Atom 2 is positionally displaced



(a) Front view

(b) Left view

(c) Free view

Index	0	1	2	3	4	5	6	7	8–13
Element	O	C	O	C	N	N	C	C	H

(d) List of atoms

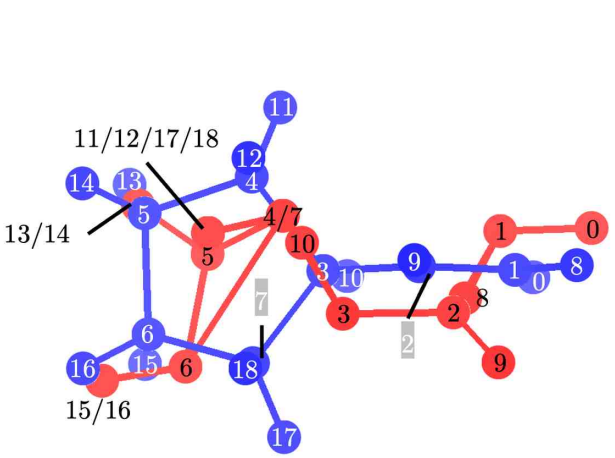
Figure 5.27: Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_16399`. The atomic labels are shown for the reference structure (white digits) and the ML-predicted structure (black digits).

2. The ring (comprising atoms 3–7) is twisted due to the displacement of atom 7
3. Atoms 10 and 11 are superimposed at identical coordinates (see Appendix P.3 for coordinate)
4. Atoms 12 and 13 are superimposed at identical coordinates (see Appendix P.3 for coordinate)

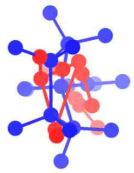
These observations demonstrate greater prediction errors compared to previous cases. Items 1 and 2 likely share the same origin as molecule `cno8_16580` — misassignment of torsion angles due to model confusion between 0° and 180° configurations. For items 3 and 4, the atomic superposition occurs because the feature vector design in GLaCE (Equation (5.41)) cannot distinguish between topologically equivalent atom pairs (10/11 and 12/13), resulting in identical predicted positions.

Figure 5.28 presents molecule `cno8_16261` from three viewing directions. As shown in Figure 5.28(a), the ML-predicted structure exhibits significant deviations from the reference DFT structure. The major discrepancies are:

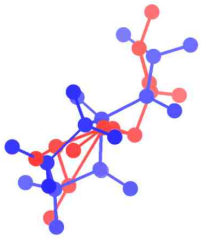
1. Positional displacement of atom 8
2. Disruption of the ring structure (atoms 3–7) due to identical coordinates for atoms



(a) Front view



(b) Left view



(c) Free view

Index	0	1	2	3	4	5	6	7	8–18
Element	O	C	N	C	C	C	C	C	H

(d) List of atoms

Figure 5.28: Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_16261`. The atomic labels are shown for the reference structure (white digits) and the ML-predicted structure (black digits).

4 and 7 (see Appendix P.4 for coordinate)

3. Superposition of atoms 11, 12, 17, and 18 at identical coordinates (see Appendix P.4 for coordinate)

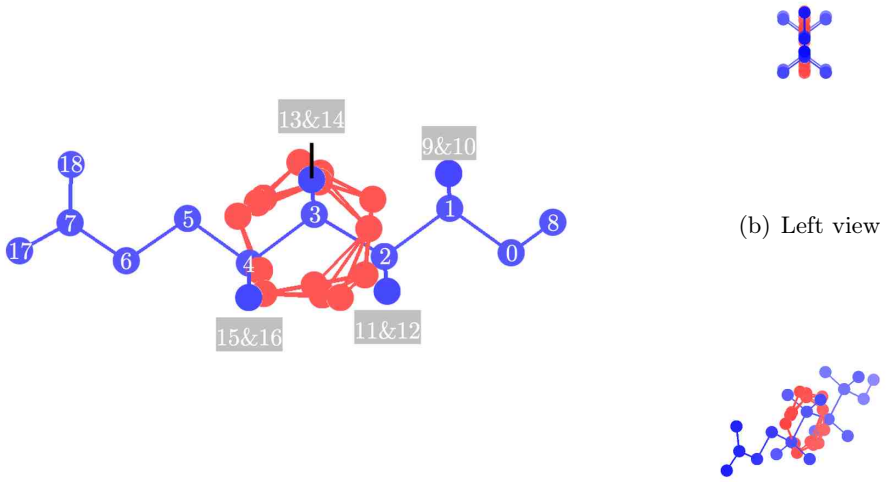
4. Superposition of atoms 13 and 14 at identical coordinates (see Appendix P.4 for coordinate)

5. Superposition of atoms 15 and 16 at identical coordinates (see Appendix P.4 for coordinate)

In `cno8_16261`, only error Item 1 stems from incorrect torsion angle assignment. Items 2–5 all result from atomic superposition effects (as described previously), which have severely distorted the molecular geometry and consequently produced a high RMSD value.

The final molecule examined is `cno8_15931`, which exhibits the highest RMSD value in the test set. Figure 5.29 presents a comparative visualisation between the ML-predicted and DFT reference structures. The predicted structure shows substantial deviation from the reference, manifesting as a clustered, tangled arrangement of overlapping rings, in stark contrast to the linear configuration of the reference molecule.

This extreme structural distortion renders detailed error analysis particularly challenging,



(b) Left view

(a) Front view

(c) Free view

Index	0	1	2	3	4	5	6	7	8–18
Element	O	C	C	C	C	O	N	C	H

(d) List of atoms

Figure 5.29: Structural comparison between the DFT reference (blue) and ML-predicted (red) configurations of molecule `cno_15931`. The atomic labels are shown for the reference structure (white digits) only due to the clustered atoms in the ML-predicted structure.

as the complexity of the displacement makes it difficult to identify individual fragment errors. Examination of the atomic coordinates in Appendix P.5 further reveals severe superposition issues among hydrogen atoms. These compounding factors — the gross structural distortion and multiple atomic superpositions — collectively account for the molecule's position as the highest-RMSD outlier in the dataset.

5.4 Result and Discussion on Time Efficiency

In project MSP, I also measured the time cost for both training and testing. The used machine for both training and predicting is the same personal desktop computer (Apple Mac mini) as the one used in project ACP. Further specification of the machine can be found at Section 4.3.1. In this section, a short summary and analysis of the time efficiency of the GLaCE- and MPF-based models is given.

5.4.1 Training Time Efficiency

The training dataset of these models is "Struct_Train", consist of 12,124 molecules with 203,052 atoms (see Section 3.1 for further information). From the training dataset, 200,449 bond lengths, 344,549 bond angles and 241,945 torsion angles are extracted and processed for training the BG, AG and TG sub-models respectively (see Table 5.9). The training

Table 5.9: Numbers of extracted values for bond lengths, bond angles and torsion angles for training and testing datasets in Project MSP

Dataset	Molecule	Atom	Bond	Bond Angle	Torsion Angle
"Struct_Train"	12,124	203,052	200,449	344,549	241,945
"Struct_Test"	1,393	21,044	20,941	34,796	32,235
"LM_Struct_Test"	139	19,151	19,835	36,552	29,912

Table 5.10: The time consumption for training GLaCE- and MPF-based molecule structure prediction models. Due to the current implementation of the program "prj_acdm", the cost of time can only be measured with a precision of seconds.

Unit: Seconds	Data Read-in	Processing Time	Training Time			Total
			BG	AG	TG	
Model_1	2	178	86	153	97	516
Model_2			77	125	97	299
Model_3			87	153	95	335
Average	2	178	83	144	96	383

parameters are presented in Section 5.1.4.

The time to complete the training of the above models is presented in Table 5.10. The total time is divided into three parts: the data read-in time, the data processing time with GLaCE & MPF algorithms, and the NN prediction time for each sub-model. It can be observed that the machine spends most of its time in processing the data and training the models. It takes only 178 seconds to apply both the GLaCE and MPF algorithms on the data for bond lengths, bond angles and torsion angles. In addition, the cost of training depends on the amount of data in the training set, where the AG sub-models take more time than the BG & TG sub-models. The average total time for each model is about 383 seconds, which is affordable for personal computers.

5.4.2 Predicting Time Efficiency

In project MSP, the trained models are tested using two datasets, "Struct_Test" and "LM_Struct_Test". The numbers of bond lengths, bond angles and torsion angles in each dataset are presented in Table 5.9. These two datasets have similar numbers of atoms, but they are largely different in the size of each molecule. By comparing the time cost for both of these datasets the differences in time efficiency of the models used in project MSP can be found.

The time costs for both tests are recorded and presented in Table 5.11. For the test on GDB-based dataset "Struct_Test", the average time cost of the whole prediction process is only 12 seconds. This means that the GLaCE- and MPF-based models are highly efficient in predicting structure for datasets consisting of small molecules. The time cost for the test on the larger molecule dataset "LM_Struct_Test" shows that the efficiency of the model decreases, especially in the GLaCE & MPF processing stages. A possible reason might

Table 5.11: The time cost when predicting structures from datasets "Struct_Test" and "LM_Struct_Test". Due to the current implementation of the program "prj_acdm" the time cost can only be measured with a precision of seconds.

Unit: Seconds		**Data Read-in**	**Processing Time**	**Predicting** BG	AG	TG	**Total**
Dataset "Struct_Test"	Model_1	<1	10	<1	1	1	12
	Model_2			<1	1	1	12
	Model_3			<1	1	1	12
	Average	<1	10	<1	1	1	12
Dataset "LM_Struct_Test"	Model_1	1	115	<1	1	1	117
	Model_2			<1	1	1	116
	Model_3			<1	1	1	116
	Average	1	115	<1	1	1	116

be that the time complexity of the graph-based operations (including GLaCE and MPF) is $O(n^2)$ (where n denotes the number of atoms in a molecule) for each molecule. With the larger size of each molecule, the time cost can increase significantly. Apart from that, the differences in efficiency in the data read-in and NN predicting stages are negligible. Although the time cost increases more than 10 times when predicting the structure for larger molecules, the total time is still affordable, with an average value of 116 seconds.

It can be concluded that, the time efficiency of these GLaCE- and MPF-based models in both training and testing is affordable on personal desktop computers, satisfying the design objective of the GLaCE and MPF algorithm.

5.5 Summary of Project MSP

In this chapter, I introduced an original method that can extract the potential connectivity relations from molecular graphs, namely the graph latent connectivity enhancement (GLaCE) algorithm. I also presented an example application of the combination of GLaCE and MPF algorithms in molecular structure prediction. From the later tests on different datasets it can be observed that the models can predict bond lengths and bond angles with good accuracy, but are less reliable for torsion angle prediction. In addition, I also found that these GLaCE- and MPF-based models perform poorly for larger molecule dataset, especially in predicting torsion angles. I also analysed the time efficiency of the models for both training and testing operations. Results show that the GLaCE algorithm is affordable for personal computers.

It can be concluded that the GLaCE algorithm is capable of effectively extracting latent connectivity from the molecular graphs. When combined with the MPF algorithm, it enables accurate 2D structural prediction (only bond lengths and bond angles) for molecules that have similar size to those in the training set. However, a comprehensive assessment reveals that the model exhibits systematic confusion in distinguishing between 0° and 180°

torsion angle assignments, and demonstrates limited capability in differentiating atoms occupying equivalent topological positions. Therefore, further optimisations for the GLaCE algorithm are also possible and are necessary to address the great loss in accuracy when predicting bond angles and torsion angles in datasets that contain larger molecules than those in the training set.

Chapter 6

Program: "prj_acdm"

As part of the implementation of the above projects (ACP & MSP), a modular C++ program has been developed, referred to as the platform program for the Atomic Charge and Dipole Moment project ("prj_acdm"). This program originates from the previous Python version. The Python version was released with the publication that described the model it implemented [113] and is open source online[1].

6.1 General Design

The design of "prj_acdm" follows the modular concept and uses object-oriented programming techniques. To improve the performance of the program, many operations such as data read-in and GLaCE & MPF process are multithreaded. In addition, the program is compiled using CMake, making "prj_acdm" available on multiple platforms, including Windows, Linux, and macOS. The program is controlled by input files where the instructions are saved and the program proceeds according to those instructions. For further information about the input files, please see the readme file with the source code.

The "prj_acdm" program implements both the project ACP and project MSP as well as all necessary supporting operations. These operations are implemented in different modules. "prj_acdm" currently consists of three modules: module "cppgd" (short for C++ version of Gaussian data management[2]); module "predac" (short for predict atomic charge); module "predco" (short for predict coordinate[3]). Readers are referred to Section 6.3 for further information about functionality.

Please note that the program is not yet available online at this time and will be available as open-source with later publication. For any inquiries regarding "early-access" to the source code or executable binary, please contact the author.

[1] https://doi.org/10.5281/zenodo.10149110

[2] Older version of "cppgd" is distributed independently on GitHub at https://github.com/xieqin74123/cppgd.

[3] Refers to the molecular structure here, as the program does not output coordinates directly.

6.2 Compilation Requirements and Dependencies

Program "prj_acdm" requires following software to compile

- `C++` compiler with `C++17` support (GCC, Clang, MSVC, etc.),
- CMake 3.8 or later, and
- Python 3.7 or later.

The following software are required to run "prj_acdm":

- Python $\geq$ 3.7 and $\leq$ 3.11, and
- Open Babel 2.4 or later.

In addition, the following Python package are required:

- Numpy
- TensorFlow $\geq$ 2.7 and $\leq$ 2.15.1 (Not compatible with Keras3),
- pandas, and
- openpyxl.

6.3 Functionality

The primary functionality of "prj_acdm" includes data read-in, data processing (like filtering), MPF algorithm operations, GLaCE algorithm operations and NN training & predicting. In this section, the functionality and dependencies of each module are presented.

6.3.1 Module "cppgd"

Module "cppgd" is the base module of "prj_acdm". It controls the functions of data input and output, for both training and predicting purposes. The current version of "prj_acdm" supports the following data format:

- mol2 file (read only),
- chargemol output file (read only),
- Gaussian log file (read only),
- SMILES file (read only),
- xyz file (write only), and
- Numpy npy file (write only).

Dataset merging is another important function of module "cppgd". This operation can combine two datasets into one unified dataset. This can be useful where the original data

are sourced from different files. For example, when creating dataset for "Model_DDEC" in project ACP, the topological data are from mol2 files and the DDEC charge data are from chargemol output files. After reading-in these two parts of data into two dataset separately, a combined training or testing dataset can be generated by dropping unused part of data and merging these two dataset.

Another function of this model is data filtering. This operation can remove those molecules without required data. For example, the data extracted from ZINC database can lack some information (like charge). By filtering through the read-in data, these faulty molecules are dropped ensuring the quality of data in the NN model training and predicting.

6.3.2 Module "predac"

The "predac" module implements the MPF algorithm introduced in project ACP. Module "predac" creates an independent job container based on selected datasets that is created by module "cppgd", i.e. "cppgd" is the dependent module of "predac". After the job container is built, users can change the training parameters by using adding instructions in input files. The training and predicting operations can be also run within this module, where a Python script will be written to a temporary directory and run. The model and other output from the training or testing process will be saved to the specified path.

6.3.3 Module "predco"

Module "predco" is the module that performs the molecular structure prediction. Similar to module "predac", module "predco" also requires module "cppgd" to acquired data and create a job container. The training parameters are also adjustable within the job container. When performing a training or prediction, a Python script will be run with all necessary output written to a specified directory.

6.4 Implementation

The entire program mainly consists of two programming languages, `C++` and Python.

`C++` (coding according to the 2017 standard) is the primary language for "prj_acdm", with 20,585 lines of code and constitutes a proportion of 93.40% across all files. Here, the `C++` part of the code operates the main operating logic of the program, including reading input files, generating datasets in memory and storing the parameters of training and predicting. In addition, since the MPF and GLaCE algorithms are relatively costly, these two operations are run only in the `C++` code and are multithreaded, ensuring the efficiency of program.

The secondary language for the program is Python (version 3.10), which has 772 lines (3.50%) of code. The uses of Python is mainly NN-related, since both training and predicting in module "predac" and "predco" are based on TensorFlow [57]. Here, the Python scripts are embedded as a `C++` function by a Python sub-program "PEC" during compi-

lation[4]. It is included in the "prj_acdm" source code, and "PEC" is short for Python code embedder for C++) . Thus, only one independent executable is needed when runing "prj_acdm".

The remaining 3.12% of the code consists of CMake files and a Markdown readme file. The use of CMake ensures the cross-platform availability of the program, and the readme file provides a detailed guideline for writing input files with descriptions for each instructions.

6.5 Summary of Program "prj_acdm"

In this chapter, I briefly introduce the general design and current implementation of program "prj_acdm", which integrates project ACP and project MSP. The modular architecture, platform compatibility and multithreading availability ensure the maintainability, extensibility and efficiency of the program. The program guarantees the training and testing mentioned in previous chapters is carried out correctly and effectively. Further optimisation and extensions are planned for the program "prj_acdm" to improve readability, functionality and standardisation in future versions.

[4]Open-sourced on GitHub in https://github.com/xieqin74123/pec.

Chapter 7

Summary and Further Work

In this thesis I investigated the potential of machine learning models to efficiently predict molecular properties from molecular connectivity. This strategy is especially suitable for applications such as searching for optimal molecules in large databases and predicting the properties of large molecules. In this thesis I present two original algorithms: the message-passing featuriser (MPF) and the graph latent connectivity enhancement (GLaCE) algorithm. The MPF operates on graphs, and passes messages from one node to its neighbouring nodes. This process aggregates the information from a nearby environment, and the information is extracted later as a machine learning input feature for each node. The GLaCE algorithm also focuses on graphs. It clusters two nodes in a graph into a new node, and rebuilds the connectivity network based on these newly-created nodes. By applying GLaCE iteratively on a graph, where the output graph from each iteration serves as the input for the next, the latent connectivity can be made explicit.

These two algorithms enable the possibility of filtering large numbers of molecules in a short time, based on only the connectivity information. In this thesis two example applications for MPF and GLaCE algorithms are presented: atomic charge prediction (project ACP) and molecular structure prediction (project MSP). In project ACP, the MPF algorithm has been validated as effective and accurate. The GLaCE algorithm also has been proved to be feasible in project MSP, especially for the molecules with size similar to those in the training set. Additionally, all models in both project ACP and MSP can accept SMILES format for molecules of arbitrary size as input. Moreover, the models used in this thesis demonstrate high efficiency in both training and prediction, enabling a personal computer to filter large molecular databases effectively.

However, the above tests also exhibit some limitations of the models. Firstly, both the MPF and GLaCE algorithms require further decoupling to enabling multithreading for a single large molecule, which would further improve the runtime performance. Secondly, the MPF algorithm has an important limitation when considering long-range interactions between nodes, which can be important in large molecules. The third limitation of this work is that the datasets are calculated in vacuum condition ($\varepsilon = 1$, see Section 3.1.3).

Thus, the current results can only fundamentally validate the MPF and GLaCE algorithms, and the presented models are not yet suitable for any application in drug development. In addition, both MPF and GLaCE algorithms are based on graphs only, neglecting the conformational structures of molecules. This problem requires further improvement in the algorithm to include the conformation information in the models. Apart from that, project MSP exposes a great limitation of the combined usage of MPF and GLaCE, where the models fail to predict bond angle (level-III) and torsion angle (level-IV) in larger molecules with satisfactory accuracy. More importantly, the prediction accuracy for torsion angles in project MSP is also less reliable, indicating potential information loss in multiple iterations of the GLaCE operations. This results in a poor quality of predicted 3D structures of molecules. Last, but not least, the MPF and GLaCE algorithms are still at the very beginning stage of development. Thus, the current implementation of these algorithms is not ready for testing on benchmark datasets (such as datasets in Kaggle[1]).

To overcome these limitations, several improvements and modifications need to be made. First, different conformation structure for one SMILES string should be generated and added to the training dataset. these structures can be distinguished by adding a conformational label (a number) into the input feature vector. This can help the neural network model distinguish different conformations and learn the relations between different conformational structures when training. In addition, by properly labelling the conformers in the training dataset, the model can produce a high-confidence result by default in the prediction phase, and leverage alternative conformational labels to produce diverse structural outputs from a single SMILES input. This adaptation could improve the GLaCE framework's performance in conformation-dependent tasks. Secondly, a new training and dataset calculated in correct solvent conditions can further test the performance of MPF and GLaCE framework models in applications on molecular docking in drug development. Furthermore, to compensate for the potential loss of information after several iterations of GLaCE operations, the application could be designed to combine several levels of graphs. Take the torsion angle prediction as an example. The model can extract NN input features for each torsion from the molecular graph (level-I), bond graph (level-II), angle graph (level-III) and torsion graph (level-IV) at the same time. By combining information from different sourced graphs this loss of information might be compensated, which could help the model distinguish the 0° and 180° torsions. Additionally, implementing a validation mechanism for molecular structures during torsion angle assignment could mitigate errors, such as fragment inversion. Also, incorporating an additional feature — such as the count of topologically equivalent atoms (quantified via the atom-level descriptor $f_{\alpha}^{\mathrm{IV,atom}}$) — into the ML input feature vector (see Equation (5.41)) may enhance the model's ability to differentiate between these atoms. Finally, a full implementation of "prj_acdm" with a Python interface is planned: by doing so, direct comparisons of both the MPF and GLaCE algorithms to existing models can be achieved.

In summary, this thesis introduced two algorithms — MPF and GLaCE — for molecular-

[1] A platform for data science competitions and dataset dissemination. Home page: https://www.kaggle.com/, accessed 15-May-2025.

oriented machine learning. While both algorithms still possess certain limitations, the findings highlight their strong potential to enable efficient molecular property prediction. In particular, these methods make it possible to search large molecular databases using a normal personal computer. I believe this work provides a useful basis for future developments in virtual screening, as well as molecular machine learning more generally.

Data and Software Availability

GDB-13 Database

The entire GDB-13 database and its subsets are available on Reymond Research Group[2]. The subset used in this work is "gdb13.cno.tgz".

ChEMBL-35 Database

The entire ChEMBL-35 database is available online[3].

ZINC Database

The used ZINC20 database is available online[4].

Open Babel Software

Open Babel is open-sourced under GPL-2.0 license and available on Github[5].

Gaussian16 DFT Software

Gaussian16 is a commercially licensed software. Further information can be found on Gaussian16 homepage[6].

Chargemol Software

Chargemol is open-sourced on Github[7]. Latest version available on SOURCEFORGE[8].

[2] https://gdb.unibe.ch/downloads/, accessed 13-April-2025
[3] https://chembl.gitbook.io/chembl-interface-documentation/downloads, accessed 13-April-2025
[4] https://zinc20.docking.org/, accessed 13-April-2025
[5] https://github.com/openbabel/openbabel, accessed 13-April-2025
[6] https://gaussian.com/gaussian16/, accessed 13-April-2025
[7] https://github.com/berquist/chargemol, accessed 13-April-2025
[8] https://sourceforge.net/projects/ddec/, accessed 13-April-2025

TensorFlow

TensorFlow is open-sourced under Apache License 2.0 on GitHub[9]. Further information can be found on TensorFlow honepage[10].

Program "prj_acdm"

Program "prj_acdm" is currently not available online. For any inquiries regarding the source code or executable binary, please contact the author via email[11].

RDKit Package

The Python package RDKit is open-sourced under the Creative Commons Attribution-ShareAlike 4.0 License. Further information can be found here[12].

VMD (Visual Molecular Dynamics)

VMD is an open-sourced software which is available here[13].

[9] https://github.com/tensorflow/tensorflow, accessed 15-May-2025
[10] https://www.tensorflow.org/, accessed 15-May-2025
[11] Primary: q.xie21@imperial.ac.uk. Alternative: xieqin74123@outlook.com
[12] https://www.rdkit.org/
[13] https://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=VMD

Bibliography

[1] Jessica Vamathevan et al. "Applications of machine learning in drug discovery and development". In: *Nature reviews Drug discovery* 18.6 (2019), pp. 463–477.

[2] Clémence Réda, Emilie Kaufmann, and Andrée Delahaye-Duriez. "Machine learning applications in drug development". In: *Computational and structural biotechnology journal* 18 (2020), pp. 241–252.

[3] Suresh Dara et al. "Machine learning in drug discovery: a review". In: *Artificial intelligence review* 55.3 (2022), pp. 1947–1999.

[4] Matheus Nunes da Rocha et al. "Ligand and structure-based virtual screening approaches in drug discovery: minireview". In: *Molecular Diversity* (2024), pp. 1–11.

[5] Francesco Gentile et al. "Artificial intelligence–enabled virtual screening of ultra-large chemical libraries with deep docking". In: *Nature Protocols* 17.3 (2022), pp. 672–697.

[6] Bharath Ramsundar et al. "Is multitask deep learning practical for pharma?" In: *Journal of chemical information and modeling* 57.8 (2017), pp. 2068–2076.

[7] Xujun Zhang et al. "Efficient and accurate large library ligand docking with KarmaDock". In: *Nature Computational Science* 3.9 (2023), pp. 789–804.

[8] Xujun Zhang et al. "Advancing ligand docking through deep learning: challenges and prospects in virtual screening". In: *Accounts of chemical research* 57.10 (2024), pp. 1500–1509.

[9] Miguel Gallegos, José Manuel Guevara-Vela, and Ángel Martín Pendás. "NNAIMQ: A neural network model for predicting QTAIM charges". In: *The Journal of Chemical Physics* 156.1 (2022), p. 014112.

[10] Jike Wang et al. "DeepAtomicCharge: a new graph convolutional network-based architecture for accurate prediction of atomic charges". In: *Briefings in bioinformatics* 22.3 (2021), bbaa183.

[11] Dong-Xia Zhao et al. "A model of atoms in molecules based on potential acting on one electron in a molecule: I. Partition and atomic charges obtained from ab initio calculations". In: *International Journal of Quantum Chemistry* 118.15 (2018), e25610.

[12] James X Mao. "Atomic charges in molecules: a classical concept in modern computational chemistry". In: *Journal of Postdoctoral Research* 2.2 (2014).

[13] Christopher I Bayly et al. "A well-behaved electrostatic potential based method using charge restraints for deriving atomic charges: the RESP model". In: *The Journal of Physical Chemistry* 97.40 (1993), pp. 10269–10280.

[14] Friedrich Hund. "Zur deutung der molekelspektren. i". In: *Zeitschrift für Physik* 40.10 (1927), pp. 742–764.

[15] Robert S Mulliken. "The assignment of quantum numbers for electrons in molecules. I". In: *Physical Review* 32.2 (1928), p. 186.

[16] Robert S Mulliken. "Electronic population analysis on LCAO–MO molecular wave functions. I". In: *The Journal of Chemical Physics* 23.10 (1955), pp. 1833–1840.

[17] CW Kern and M Karplus. "Analysis of charge distributions: hydrogen fluoride". In: *The Journal of Chemical Physics* 40.5 (1964), pp. 1374–1389.

[18] L Ch Cusachs and P Politzer. "On the problem of defining the charge on an atom in a molecule". In: *Chemical Physics Letters* 1.11 (1968), pp. 529–531.

[19] Per-Olov Löwdin. "On the non-orthogonality problem connected with the use of atomic wave functions in the theory of molecules and crystals". In: *The Journal of Chemical Physics* 18.3 (1950), pp. 365–375.

[20] Alan E Reed, Robert B Weinstock, and Frank Weinhold. "Natural population analysis". In: *The Journal of chemical physics* 83.2 (1985), pp. 735–746.

[21] Alan E Reed and Frank Weinhold. "Natural bond orbital analysis of internal rotation barriers and related phenomena". In: *Israel journal of chemistry* 31.4 (1991), pp. 277–285.

[22] Alberte Pullman and Bernard Pullman. "Molecular electrostatic potential of the nucleic acids". In: *Quarterly reviews of biophysics* 14.3 (1981), pp. 289–380.

[23] Tian Lu and Fei-Wu CHEN. "Comparison of computational methods for atomic charges". In: *Acta Physico-Chimica Sinica* 28.1 (2012), pp. 1–18.

[24] U Chandra Singh and Peter A Kollman. "An approach to computing electrostatic charges for molecules". In: *Journal of computational chemistry* 5.2 (1984), pp. 129–145.

[25] Lisa Emily Chirlian and Michelle Miller Francl. "Atomic charges derived from electrostatic potentials: A detailed study". In: *Journal of Computational Chemistry* 8.6 (1987), pp. 894–905.

[26] Curt M Breneman and Kenneth B Wiberg. "Determining atom-centered monopoles from molecular electrostatic potentials. The need for high sampling density in formamide conformational analysis". In: *Journal of computational chemistry* 11.3 (1990), pp. 361–373.

[27] Emma Sigfridsson and Ulf Ryde. "Comparison of methods for deriving atomic charges from the electrostatic potential and moments". In: *Journal of Computational Chemistry* 19.4 (1998), pp. 377–395.

[28] Richard FW Bader and PM Beddall. "Virial field relationship for molecular charge distributions and the spatial partitioning of molecular properties". In: *The Journal of Chemical Physics* 56.7 (1972), pp. 3320–3329.

[29] Fred L Hirshfeld. "Bonded-atom fragments for describing molecular charge densities". In: *Theoretica chimica acta* 44.2 (1977), pp. 129–138.

[30] Aleksandr V Marenich et al. "Charge model 5: An extension of Hirshfeld population analysis for the accurate description of molecular interactions in gaseous and condensed phases". In: *Journal of chemical theory and computation* 8.2 (2012), pp. 527–541.

[31] Joey W Storer et al. "Class IV charge models: A new semiempirical approach in quantum chemistry". In: *Journal of computer-aided molecular design* 9 (1995), pp. 87–110.

[32] Jiabo Li et al. "New class IV charge model for extracting accurate partial charges from wave functions". In: *The Journal of Physical Chemistry A* 102.10 (1998), pp. 1820–1831.

[33] Paul Winget et al. "Charge Model 3: A class IV charge model based on hybrid density functional theory with variable exchange". In: *The Journal of Physical Chemistry A* 106.44 (2002), pp. 10707–10717.

[34] Joseph M Brom et al. "A class IV charge model for boron based on hybrid density functional theory". In: *The Journal of Physical Chemistry A* 107.33 (2003), pp. 6483–6488.

[35] Casey P Kelly, Christopher J Cramer, and Donald G Truhlar. "Accurate partial atomic charges for high-energy molecules using class IV charge models with the MIDI! basis set". In: *Theoretical Chemistry Accounts* 113.3 (2005), pp. 133–151.

[36] Casey P Kelly, Christopher J Cramer, and Donald G Truhlar. "SM6: A density functional theory continuum solvation model for calculating aqueous solvation free energies of neutrals, ions, and solute- water clusters". In: *Journal of chemical theory and computation* 1.6 (2005), pp. 1133–1152.

[37] Ryan M Olson et al. "Charge model 4 and intramolecular charge polarization". In: *Journal of chemical theory and computation* 3.6 (2007), pp. 2046–2054.

[38] Thomas A Manz and Nidia Gabaldon Limas. "Introducing DDEC6 atomic population analysis: part 1. Charge partitioning theory and methodology". In: *RSC advances* 6.53 (2016), pp. 47771–47801.

[39] Nidia Gabaldon Limas and Thomas A Manz. "Introducing DDEC6 atomic population analysis: part 2. Computed results for a wide range of periodic and nonperiodic materials". In: *RSC advances* 6.51 (2016), pp. 45727–45747.

[40] Thomas A Manz. "Introducing DDEC6 atomic population analysis: part 3. Comprehensive method to compute bond orders". In: *RSC advances* 7.72 (2017), pp. 45552–45581.

[41] Nidia Gabaldon Limas and Thomas A Manz. "Introducing DDEC6 atomic population analysis: part 4. Efficient parallel computation of net atomic charges, atomic spin moments, bond orders, and more". In: *RSC advances* 8.5 (2018), pp. 2678–2707.

[42] Patrick Bleiziffer, Kay Schaller, and Sereina Riniker. "Machine Learning of Partial Charges Derived from High-quality Quantum-mechanical Calculations". In: *J. Chem. Inf. Model.* 58.3 (2018), pp. 579–590.

[43] Mohammad Sufian Badar et al. "Molecular dynamics simulations: concept, methods, and applications". In: *Transdisciplinarity*. Springer, 2022, pp. 131–151.

[44] Berni Julian Alder, Thomas Everett Wainwright, et al. "Phase transition for a hard sphere system". In: *The Journal of chemical physics* 27.5 (1957), p. 1208.

[45] Berni J Alder and Thomas Everett Wainwright. "Studies in molecular dynamics. I. General method". In: *The Journal of Chemical Physics* 31.2 (1959), pp. 459–466.

[46] Helmut Grubmüller et al. "Generalized Verlet algorithm for efficient molecular dynamics simulations with long-range interactions". In: *Molecular Simulation* 6.1-3 (1991), pp. 121–142.

[47] Nicos S Martys and Raymond D Mountain. "Velocity Verlet algorithm for dissipative-particle-dynamics-based models of suspensions". In: *Physical Review E* 59.3 (1999), p. 3733.

[48] DECHIN. *from Liouville equation to velocity-Verlet algorithm*. https://www.cnblogs.com/dechinphy/p/liouville.html. Accessed: 13-MAR-2025.

[49] Scott A Hollingsworth and Ron O Dror. "Molecular dynamics simulation for all". In: *Neuron* 99.6 (2018), pp. 1129–1143.

[50] Ying Hu and Honglai Liu. *Density Function Theory*. China Science Publishing & Media Ltd., 2016.

[51] Walter Kohn and Lu Jeu Sham. "Self-consistent equations including exchange and correlation effects". In: *Physical review* 140.4A (1965), A1133.

[52] Ulf Von Barth and Lars Hedin. "A local exchange-correlation potential for the spin polarized case. i". In: *Journal of Physics C: Solid State Physics* 5.13 (1972), p. 1629.

[53] John P Perdew and Wang Yue. "Accurate and simple density functional for the electronic exchange energy: Generalized gradient approximation". In: *Physical review B* 33.12 (1986), p. 8800.

[54] Philip J Stephens et al. "Ab initio calculation of vibrational absorption and circular dichroism spectra using density functional force fields". In: *The Journal of physical chemistry* 98.45 (1994), pp. 11623–11627.

[55] Jing Kang et al. “Research progress of artificial neural networks in material science”. In: *Materials Reports* 34.21 (2020), pp. 21172–21179.

[56] François Chollet et al. *Keras*. https://keras.io. 2015.

[57] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[58] Wenxiang Liu et al. “Applications of machine learning in computational nanotechnology”. In: *Nanotechnology* 33.16 (2022), p. 162501.

[59] Shun-ichi Amari. “Backpropagation and stochastic gradient descent method”. In: *Neurocomputing* 5.4-5 (1993), pp. 185–196.

[60] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[61] Tijmen Tieleman. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), p. 26.

[62] Siddarth K Achar, Leonardo Bernasconi, and J Karl Johnson. “Machine learning electron density prediction using weighted smooth overlap of atomic positions”. In: *Nanomaterials* 13.12 (2023), p. 1853.

[63] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. “Understanding of a convolutional neural network”. In: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, pp. 1–6.

[64] Tohru Nitta. “An extension of the back-propagation algorithm to complex numbers”. In: *Neural Networks* 10.8 (1997), pp. 1391–1415.

[65] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. “Dynamic routing between capsules”. In: *Advances in neural information processing systems* 30 (2017).

[66] Alexander Novikov et al. “Tensorizing neural networks”. In: *Advances in neural information processing systems* 28 (2015).

[67] Masaki Kobayashi, Jun Muramatsu, and Haruaki Yamazaki. “Construction of high-dimensional neural networks by linear connections of matrices”. In: *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* 86.11 (2003), pp. 38–45.

[68] Jeffrey L Elman. “Finding structure in time”. In: *Cognitive science* 14.2 (1990), pp. 179–211.

[69] Hojjat Salehinejad et al. “Recent advances in recurrent neural networks”. In: *arXiv preprint arXiv:1801.01078* (2017).

[70] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. “Lstm neural networks for language modeling.” In: *Interspeech*. Vol. 2012. 2012, pp. 194–197.

[71] George Trigeorgis et al. “Adieu features? end-to-end speech emotion recognition using a deep convolutional recurrent network”. In: *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2016, pp. 5200–5204.

[72] Bing Shuai et al. “Dag-recurrent neural networks for scene labeling”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3620–3629.

[73] Kyunghyun Cho et al. “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259* (2014).

[74] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[75] Chang Weng et al. “A Thorough Investigation into the Application of Deep CNN for Enhancing Natural Language Processing Capabilities”. In: *arXiv preprint arXiv:2412.15900* (2024).

[76] Ali Bou Nassif et al. “Speech recognition using deep neural networks: A systematic review”. In: *IEEE access* 7 (2019), pp. 19143–19165.

[77] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[78] Bingbing Xu et al. “A survey on graph convolutional neural network”. In: *Chinese Journal of Computers* 43.5 (2020), pp. 755–780.

[79] Zonghan Wu et al. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[80] Justin Gilmer et al. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[81] David K Duvenaud et al. “Convolutional networks on graphs for learning molecular fingerprints”. In: *Advances in neural information processing systems* 28 (2015).

[82] Yujia Li et al. “Gated graph sequence neural networks”. In: *arXiv preprint arXiv:1511.05493* (2015).

[83] Peter Battaglia et al. “Interaction networks for learning about objects, relations and physics”. In: *Advances in neural information processing systems* 29 (2016).

[84] Justin Gilmer et al. “Message passing neural networks”. In: *Machine learning meets quantum physics*. Springer, 2020, pp. 199–214.

[85] Lorenz C Blum and Jean-Louis Reymond. “970 million druglike small molecules for virtual screening in the chemical universe database GDB-13”. In: *Journal of the American Chemical Society* 131.25 (2009), pp. 8732–8733.

[86] Barbara Zdrazil et al. “The ChEMBL Database in 2023: a drug discovery platform spanning multiple bioactivity data types and time periods”. In: *Nucleic acids research* 52.D1 (2024), pp. D1180–D1192.

[87] Mark Davies et al. “ChEMBL web services: streamlining access to drug discovery data and utilities”. In: *Nucleic acids research* 43.W1 (2015), W612–W620.

[88] John J Irwin and Brian K Shoichet. “ZINC- a free database of commercially available compounds for virtual screening”. In: *Journal of chemical information and modeling* 45.1 (2005), pp. 177–182.

[89] John J Irwin et al. “ZINC: a free tool to discover chemistry for biology”. In: *Journal of chemical information and modeling* 52.7 (2012), pp. 1757–1768.

[90] John J Irwin et al. “ZINC20—a free ultralarge-scale chemical database for ligand discovery”. In: *Journal of chemical information and modeling* 60.12 (2020), pp. 6065–6073.

[91] David Weininger. “SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules”. In: *Journal of chemical information and computer sciences* 28.1 (1988), pp. 31–36.

[92] David Weininger, Arthur Weininger, and Joseph L Weininger. “SMILES. 2. Algorithm for generation of unique SMILES notation”. In: *Journal of chemical information and computer sciences* 29.2 (1989), pp. 97–101.

[93] Noel O’Boyle and Andrew Dalke. “DeepSMILES: an adaptation of SMILES for use in machine-learning of chemical structures”. In: *ChemRxiv* (2018).

[94] Gabriel A Pinheiro et al. “Machine learning prediction of nine molecular properties based on the SMILES representation of the QM9 quantum-chemistry dataset”. In: *The Journal of Physical Chemistry A* 124.47 (2020), pp. 9854–9866.

[95] Muhamad Akrom, Supriadi Rustad, and Hermawan Kresno Dipojono. “SMILES-based machine learning enables the prediction of corrosion inhibition capacity”. In: *MRS Communications* 14.3 (2024), pp. 379–387.

[96] Tobias Fink, Heinz Bruggesser, and Jean-Louis Reymond. "Virtual exploration of the small-molecule chemical universe below 160 daltons". In: *Angewandte Chemie International Edition* 44.10 (2005), pp. 1504–1508.

[97] Noel M O'Boyle et al. "Open Babel: An open chemical toolbox". In: *Journal of cheminformatics* 3.1 (2011), pp. 1–14.

[98] M. J. Frisch et al. *Gaussian˜16 Revision C.01*. Gaussian Inc. Wallingford CT. 2016.

[99] Naruki Yoshikawa and Geoffrey R Hutchison. "Fast, efficient fragment-based coordinate generation for Open Babel". In: *Journal of cheminformatics* 11.1 (2019), p. 49.

[100] Thomas A Halgren. "Merck molecular force field. I. Basis, form, scope, parameterization, and performance of MMFF94". In: *Journal of computational chemistry* 17.5-6 (1996), pp. 490–519.

[101] Thomas A Halgren. "Merck molecular force field. II. MMFF94 van der Waals and electrostatic parameters for intermolecular interactions". In: *Journal of Computational Chemistry* 17.5-6 (1996), pp. 520–552.

[102] Thomas A Halgren. "Merck molecular force field. III. Molecular geometries and vibrational frequencies for MMFF94". In: *Journal of computational chemistry* 17.5-6 (1996), pp. 553–586.

[103] Thomas A Halgren and Robert B Nachbar. "Merck molecular force field. IV. Conformational energies and geometries for MMFF94". In: *Journal of computational chemistry* 17.5-6 (1996), pp. 587–615.

[104] Thomas A Halgren. "Merck molecular force field. V. Extension of MMFF94 using experimental data, additional computational data, and empirical rules". In: *Journal of Computational Chemistry* 17.5-6 (1996), pp. 616–641.

[105] Open Babel Team. *Open Babel User Guide*. Accessed: 11-April-2025. URL: https://openbabel.org/docs/index.html.

[106] aGA Petersson et al. "A complete basis set model chemistry. I. The total energies of closed-shell atoms and hydrides of the first-row elements". In: *The Journal of chemical physics* 89.4 (1988), pp. 2193–2218.

[107] GA Petersson and Mohammad A Al-Laham. "A complete basis set model chemistry. II. Open-shell systems and the total energies of the first-row atoms". In: *The Journal of chemical physics* 94.9 (1991), pp. 6081–6090.

[108] A Daniel Boese, Jan ML Martin, and Nicholas C Handy. "The role of the basis set: Assessing density functional theory". In: *The Journal of chemical physics* 119.6 (2003), pp. 3005–3014.

[109] Narbe Mardirossian and Martin Head-Gordon. "Thirty years of density functional theory in computational chemistry: an overview and extensive assessment of 200 density functionals". In: *Molecular physics* 115.19 (2017), pp. 2315–2372.

[110] Thomas A Manz and Nidia Gabaldon Limas. *Chargemol program for performing DDEC analysis*. ddec.sourceforge.net. Accessed: 13-April-2025. 2017.

[111] Christopher J Cramer and Donald G Truhlar. "AM1-SM2 and PM3-SM3 parameterized SCF solvation models for free energies in aqueous solution". In: *Journal of Computer-Aided Molecular Design* 6 (1992), pp. 629–666.

[112] Jike Wang et al. "Fast and accurate prediction of partial charges using Atom-Path-Descriptor-based machine learning". In: *Bioinformatics* 36.18 (2020), pp. 4721–4728.

[113] Qin Xie and Andrew P Horsfield. "Coordinate-Free and Low-Order Scaling Machine Learning Model for Atomic Partial Charge Prediction for Any Size of Molecules". In: *Journal of Chemical Information and Modeling* 64.11 (2024), pp. 4419–4425.

[114] *RDKit: Open-source cheminformatics*. URL: https://www.rdkit.org.

[115] William Humphrey, Andrew Dalke, and Klaus Schulten. “VMD: visual molecular dynamics”. In: *Journal of molecular graphics* 14.1 (1996), pp. 33–38.

Appendix A

Notation Table for Project ACP

Notation	Typeset	Description	Domain	Def. Eq.
$\mathbf{A}$	bold Roman	adjacency matrix	$\{0,1\}^{n\times n}$	Equation (4.1d)
A_{ij}	italic	element in adjacency matrix with index i and j	$\{0,1\}$	Equation (4.1d)
$\text{Bond}(v_i, v_j)$	Roman	the function interpreting the bond order between v_i and v_j	$V \times V \rightarrow \{0,1,2,3,4,5\}$	Equation (4.1f)
$\mathbf{b}^l$	bold Roman	the bias vector of layer l in NN	$\mathbb{R}^{q_l}$	N/A
C	italic	atomic constant properties set	$\mathbb{R}^{q_l}$	Equations (4.1a) and (4.6a)
$\mathbf{c}_\chi$	bold Roman	Pauling electronegativity vector	$\mathbb{R}^{118}$	Equation (4.6b)
d	italic	number of feature at each atom	$\mathbb{N}^+$	N/A
E	italic	bond set	N/A	Equation (4.1c)
e_{ij}	italic	bond connecting atom i and j	E	Equation (4.1c)
$\mathbf{F}$	bold Roman	molecular final feature vector	$\mathbb{R}^{n\times s}$	Equation (4.5b)
$\mathbf{F}'$	bold Roman	normalised molecular final feature vector	$\mathbb{R}^{n\times s}$	Equation (4.11b)
$\mathbf{f}_i$	bold Roman	atomic final feature vector for atom i	$\mathbb{R}^s$	Equations (4.5b) and (4.10a)
$\mathbf{f}'_i$	bold Roman	normalised atomic final feature vector for atom i	$\mathbb{R}^s$	Equation (4.11b)
f_i^j	Roman	final feature j for atom i	$\mathbb{R}$	Equation (4.5b)
f'^j_i	Roman	normalised final feature j for atom i	$\mathbb{R}$	Equation (4.11a)
$\mathrm{f}_i^{\text{MNE}}$	Roman	mean neighbouring electronegativity final feature for atom i	$\mathbb{R}$	Equation (4.10e)
$\mathrm{f}'^{\text{MNE}}_i$	Roman	normalised mean neighbouring electronegativity final feature for atom i	$\mathbb{R}$	Equation (4.11a)
$\mathrm{f}_i^{\text{SEN}}$	Roman	self electronegativity final feature for atom i	$\mathbb{R}$	Equation (4.10f)
$\mathrm{f}'^{\text{SEN}}_i$	Roman	normalised self electronegativity final feature for atom i	$\mathbb{R}$	Equation (4.11a)
$\mathrm{f}_i^{\text{atom}}$	Roman	atom contribution final feature for atom i	$\mathbb{R}$	Equation (4.10c)
$\mathrm{f}'^{\text{atom}}_i$	Roman	normalised atom contribution final feature for atom i	$\mathbb{R}$	Equation (4.11a)
$\mathrm{f}_i^{\text{bond}}$	Roman	bond contribution final feature for atom i	$\mathbb{R}$	Equation (4.10b)

Notation	Typeset	Description	Domain	Def. Eq.
$\mathrm{f}'^{\mathrm{bond}}_i$	Roman	normalised bond contribution final feature for atom i	$\mathbb{R}$	Equation (4.11a)
$\mathrm{f}^{\mathrm{self}}_i$	Roman	self final feature for atom i	$\mathbb{R}$	Equation (4.10d)
$\mathrm{f}'^{\mathrm{self}}_i$	Roman	normalised self final feature for atom i	$\mathbb{R}$	Equation (4.11a)
$\mathbf{G}^l_i$	bold Roman	NN feature in layer l	$\mathbb{R}^{q_l}$	Equation (4.12c)
$\mathbf{H}_t$	bold Roman	molecular feature matrix at time step t	$\mathbb{R}^{n\times d}$	Equations (4.2b) and (4.4b)
$\mathbf{h}_{t,i}$	bold Roman	atomic feature vector for atom$i$ at time step t	$\mathbb{R}^d$	Equations (4.2b), (4.4b) and (4.7a)
$\mathrm{h}^j_{t,i}$	Roman	feature j for atom i at time step t	$\mathbb{R}$	Equations (4.2b) and (4.4b)
$\mathrm{h}^{\mathrm{self}}_{t,i}$	Roman	self feature for atom i at time step t	$\mathbb{R}$	Equations (4.7b) and (4.9)
$I(X)$	italic	molecular initialisation function	$X\to\mathbb{R}^{n\times d}$	Equation (4.2c)
$I'(X,i)$	italic	atomic initialisation function	$X\times\mathbb{N}^*\to\mathbb{R}^d$	Equation (4.2c)
$I'_j(X,i)$	italic	feature j initialisation function	$X\times\mathbb{N}^*\to\mathbb{R}$	Equation (4.2c)
I'_{self}	italic	self feature initialisation function	$\emptyset\to\{1\}$	Equation (4.7b)
i	italic	indexing variable	$\mathbb{N}$	N/A
j	italic	indexing variable	$\mathbb{N}$ or specified domain	N/A
k	italic	number of messages in each atomic message vector $\mathbf{m}_{t,i}$	$\mathbb{N}^*$	N/A
l	italic	layer index of neural network	$\mathbb{N}$	Equation (4.12a)
$\mathbf{M}_t$	bold Roman	molecular message vector at time step t	$\mathbb{R}^{n\times k}$	Equations (4.3a) and (4.3b)
$M_t(P_{t-1})$ or $M(P_{t-1})$	italic	molecular message generation function at time t	$P\to\mathbb{R}^{n\times k}$	Equation (4.3b)
$M'_t(P_{t-1},i)$ or $M'(P_{t-1},i)$	italic	atomic message generation function at time t	$P\times\mathbb{N}^*\to\mathbb{R}^k$	Equation (4.3b)
$M'_{t,j}(P_{t-1},i)$ or $M'_j(P_{t-1},i)$	italic	message j generation function at time t	$P\times\mathbb{N}^*\to\mathbb{R}$	Equation (4.3b)
$M'_{\mathrm{atom}}(P_{t-1},i)$	italic	atom contribution message generation function at time t	$P\times\mathbb{N}^*\to\mathbb{R}$	Equation (4.8c)
$M'_{\mathrm{bond}}(P_{t-1},i)$	italic	bond contribution message generation function at time t	$P\times\mathbb{N}^*\to\mathbb{R}$	Equation (4.8b)
$\mathbf{m}_{t,i}$	bold Roman	atomic message vector for atom i at time step t	$\mathbb{R}^k$	Equations (4.3a), (4.3b) and (4.8a)
$\mathrm{m}^j_{t,i}$	Roman	message j for atom i at time step t	$\mathbb{R}$	Equations (4.3a) and (4.3b)
$\mathrm{m}^{\mathrm{atom}}_{t,i}$	Roman	atom contribution message for atom i at time step t	$\mathbb{R}$	Equation (4.8c)
$\mathrm{m}^{\mathrm{bond}}_{t,i}$	Roman	bond contribution message for atom i at time step t	$\mathbb{R}$	Equation (4.8b)
m	italic	maximum iteration number of MPF	$\mathbb{N}^*$	N/A
n	italic	number of atoms in molecule	$\mathbb{N}^*$	N/A
o	italic	number of NN output for each atom in output vector $\mathbf{y}_i$	$\mathbb{N}^*$	N/A
P_t	italic	molecular feature graph at time step t	N/A	Equations (4.2a) and (4.4a)
q_l	italic	number of nodes in layer l of NN	$\mathbb{N}^*$	Equation (4.12b)
$R(P_m,\mathbf{M}_m)$	italic	molecular readout function	$P\times\mathbb{R}^{n\times k}\to\mathbb{R}^{n\times s}$	Equation (4.5b)
$R'(P_m,\mathbf{M}_m,i)$	italic	atomic readout function	$P\times\mathbb{R}^{n\times k}\times\mathbb{N}^*\to\mathbb{R}^s$	Equations (4.5b) and (4.10a)
$R'_j(P_m,\mathbf{M}_m,i)$	italic	final feature j readout function	$P\times\mathbb{R}^{n\times k}\times\mathbb{N}^*\to\mathbb{R}$	Equation (4.5b)

Notation	Typeset	Description	Domain	Def. Eq.
$R'_{\text{MNE}}(P_m, i)$	italic	MNE final feature readout function	$P \times \mathbb{N}^* \to \mathbb{R}$	Equation (4.10e)
$R'_{\text{SEN}}(P_m, i)$	italic	SEN final feature readout function	$P \times \mathbb{N}^* \to \mathbb{R}$	Equation (4.10f)
$R'_{\text{atom}}(\mathbf{M}_m, i)$	italic	atom contribution final feature readout function	$\mathbb{R}^{n\times k} \times \mathbb{N}^* \to \mathbb{R}$	Equation (4.10c)
$R'_{\text{bond}}(\mathbf{M}_m, i)$	italic	bond contribution final feature readout function	$\mathbb{R}^{n\times k} \times \mathbb{N}^* \to \mathbb{R}$	Equation (4.10b)
$R'_{\text{self}}(\mathbf{M}_m, i)$	italic	self final feature readout function	$\mathbb{R}^{n\times k} \times \mathbb{N}^* \to \mathbb{R}$	Equation (4.10d)
s	italic	number of final feature for each atom	$\mathbb{N}^*$	N/A
$\mathbf{T}$	bold Roman	topology matrix	$\mathbb{N}^{n\times n}$	Equation (4.1e)
T_{ij}	italic	element in topology matrix with index i and j	$\mathbb{N}$	Equation (4.1e)
t	italic	time step	$\mathbb{N}$	N/A
$U_t(P_{t-1}, \mathbf{M}_t)$ or $U(P_{t-1}, \mathbf{M}_t)$	italic	molecular update function at time step t	$X \times \mathbb{R}^{n\times k} \to \mathbb{R}^{n\times d}$	Equation (4.4b)
$U'_t(P_{t-1}, \mathbf{M}_t, i)$ or $U'(P_{t-1}, \mathbf{M}_t, i)$	italic	atomic update function at time step t	$X \times \mathbb{R}^{n\times k} \times \mathbb{N}^* \to \mathbb{R}^d$	Equation (4.4b)
$U'_{t,j}(P_{t-1}, \mathbf{M}_t, i)$ or $U'_j(P_{t-1}, \mathbf{M}_t, i)$	italic	update function for feature j at time step t	$X \times \mathbb{R}^{n\times k} \times \mathbb{N}^* \to \mathbb{R}$	Equation (4.4b)
$U'_{\text{self}}(P_t, \mathbf{M}_{t+1}, i)$	italic	self feature update function	$X \times \mathbb{R}^{n\times k} \times \mathbb{N}^* \to \mathbb{R}$	Equation (4.9)
V	italic	atom set of the molecule	N/A	Equation (4.1b)
v_i	italic	atom i in the molecule	V	Equation (4.1b)
$\mathbf{W}^l$	bold Roman	NN weight matrix between layer l and $l+1$	$\mathbb{R}^{q_l \times q_{l+1}}$	N/A
X	italic	input molecule	N/A	Equation (4.1a)
$\mathbf{Y}$	bold Roman	molecular predicted result vector from NN	$\mathbb{R}^{n\times o}$	Equation (4.13b)
$\mathbf{Y}'$	bold Roman	normalised molecular predicted result vector from NN	$\mathbb{R}^{n\times o}$	Equation (4.12e)
$\mathbf{y}_i$	bold Roman	atomic predicted result vector for atom i from NN	$\mathbb{R}^o$	Equation (4.13b)
$\mathbf{y}'_i$	bold Roman	normalised atomic predicted result vector for atom i from NN	$\mathbb{R}^o$	Equation (4.12d)
y_i	Roman	predicted charge from atom i	$\mathbb{R}$	Equation (4.13a)
y'_i	Roman	normalised predicted charge from atom i	$\mathbb{R}$	Equation (4.12d)
Z_i	italic	atomic number of atom i	$\mathbb{N}^*$	N/A
μ_j^{preset}	Roman	average value of feature j in training set	$\mathbb{R}$	N/A
$\mu_{\mathrm{y}}^{\text{preset}}$	Roman	average value of true value in training set	$\mathbb{R}$	N/A
σ_j^{preset}	Roman	standard deviation of feature j in training set	$\mathbb{R}$	N/A
$\sigma_{\mathrm{y}}^{\text{preset}}$	Roman	standard deviation of true value in training set	$\mathbb{R}$	N/A
χ_i	Roman	Pauling electronegativity of atom i	$\mathbb{R}$	Equation (4.6c)

Appendix B

Notation Table for Project ACP

Notation	Typeset	Description	Domain
$\mathbf{A}_\lambda$	bold Roman	the adjacency matrix for level-λ graph	$\{0,1\}^{n^\lambda \times n^\lambda}$
A_{ij}^λ	italic	the element of the adjacency matrix $\mathbf{A}_\lambda$ indexed by i and j	$\{0,1\}$
$\mathbf{b}^{\lambda,l}$	bold Roman	the bias of layer l in the NN for level-λ graph	$\mathbb{R}^{q_l^\lambda}$
C_λ	italic	the set of atomic constant properties for the level-λ graph	N/A
$\mathbf{c}_\chi$	bold Roman	the Pauling-scaling electronegativity vector	$\mathbb{R}^{118}$
E_λ	italic	the edge set of graph in level λ	N/A
e_{ij}^λ	italic	edge connecting node i and j in level-λ graph	N/A
$\mathbf{F}_\lambda$	bold Roman	final feature matrix for level-λ graph	$\mathbb{R}^{n^\lambda \times s^\lambda}$
$\mathbf{F'}_\lambda$	bold Roman	normalised final feature matrix for level-λ graph	$\mathbb{R}^{n^\lambda \times s^\lambda}$
$\mathbf{f}_\alpha^\lambda$	bold Roman	final feature vector for node α in level-λ graph	$\mathbb{R}^{s^\lambda}$
$\mathbf{f'}_\alpha^\lambda$	bold Roman	normalised final feature vector for node α in level-λ graph	$\mathbb{R}^{s^\lambda}$
$\mathrm{f}_\alpha^{\lambda,j}$	bold Roman	final feature j for node α in level-λ graph	$\mathbb{R}^{s^\lambda}$
$\mathrm{f'}_\alpha^{\lambda,j}$	bold Roman	normalised final feature j for node α in level-λ graph	$\mathbb{R}^{s^\lambda}$
$\mathbf{G}_\alpha^{\lambda,l}$	bold Roman	NN feature vector for node α (of level-λ graph) in NN layer l	$\mathbb{R}^{q_l^\lambda}$
$\mathbf{H}_t^\lambda$	bold Roman	molecular feature matrix at time step t for level-λ graph	$\mathbb{R}^{n^\lambda \times d^\lambda}$
$\mathbf{h}_{t,i}^\lambda$	bold Roman	atomic feature vector for atom$i$ at time step t or level-λ graph	$\mathbb{R}^{d^\lambda}$
$\mathrm{h}_{t,i}^{\lambda,j}$	Roman	feature j for atom i at time step t or level-λ graph	$\mathbb{R}$
I_λ	italic	molecular initialisation function for level-λ graph	N/A
I'_λ	italic	molecular initialisation function for level-λ graph	N/A
I'^λ_j	italic	feature j initialisation function for	$X \times \mathbb{N}^* \to \mathbb{R}$
i	italic	indexing variable	$\mathbb{N}$ or specified domain
j	italic	indexing variable	$\mathbb{N}$ or specified domain
k^λ	italic	number of messages in the message vector for each atom in level-λ graph	$\mathbb{N}^*$
l	italic	neural network layer indexing variable	$\mathbb{N}$
$\mathbf{M}_t^\lambda$	bold Roman	graph message vector for level λ at time step t	$\mathbb{R}^{n^\lambda \times k^\lambda}$
M_t^λ	italic	message generation function at time t for level-λ graph	N/A

Notation	Typeset	Description	Domain
${M'}^{\lambda}_{t,\alpha}$	italic	message generation function at time t for node α in level-λ graph	N/A
${M'}^{\lambda,j}_{t,\alpha}$	italic	message j generation function at time t for node α in level-λ graph	N/A
$\mathbf{m}^{\lambda}_{t,\alpha}$	bold Roman	atomic message vector for node α in level-λ graph at time step t	$\mathbb{R}^{k^{\lambda}}$
$\mathrm{m}^{\lambda,j}_{t,\alpha}$	Roman	message j for node α in level-λ graph at time step t	$\mathbb{R}$
m^{λ}	italic	the number of layers (exclude input layer) in the NN for level-λ graph	$\mathbb{N}^*$
norm_{λ}	Roman	refers to the normalisation process of the NN in level-λ	N/A
n^{λ}	italic	number of node in level-λ graph	$\mathbb{N}^*$
o^{λ}	italic	number of NN output for each node in the output vector $\mathbf{y}^{\lambda}_{\alpha}$ in level-λ graph	$\mathbb{N}^*$
P^{λ}_{t}	italic	the level-λ graph at time step t	N/A
p	Roman	indexing variable	$\mathbb{N}$ or specified domain
q	Roman	indexing variable	$\mathbb{N}$ or specified domain
q^{λ}_{l}	italic	the number of neurons in layer l of the NN model in level-λ graph	$\mathbb{N}^*$
r^{λ}_{α}	italic	the minimum size of rings that atom/bond/angle/-torsion α (in level-λ graph) is in	$\mathbb{N}^*$
s^{λ}	italic	the number of final features for each node in level-λ graph	$\mathbb{N}^*$
$\mathbf{T}_{\lambda}$	bold Roman	the topology matrix for level-λ graph	$\mathbb{N}^{n^{\lambda}\times n^{\lambda}}$
T^{λ}_{ij}	italic	the element of topology matrix $\mathbf{T}_{\lambda}$ indexed by i and j	$\mathbb{N}$
t	italic	time step indexing variable	$\mathbb{N}$
U^{λ}_{t}	italic	level-λ graph update function at time step t	
${U'}^{\lambda}_{t}$	italic	update function for each node in level-λ graph at time step t	N/A
${U'}^{\lambda,j}_{t}$	italic	update function for feature j of each node in level-λ graph at time step t	N/A
V_{λ}	italic	the node set of graph in level λ	N/A
v^{λ}_{i}	italic	node i in level-λ graph	N/A
$\mathbf{W}^{\lambda,l}$	bold Roman	the weight matrix of layer l in the NN for level-λ graph	$\mathbb{R}^{q^{\lambda}_{l}\times q^{\lambda}_{l+1}}$
X_{λ}	italic	the graph in level λ in GLaCE	N/A
$\mathbf{Y}_{\lambda}$	bold Roman	graph predicted result vector from NN	$\mathbb{R}^{n^{\lambda}\times o^{\lambda}}$
$\mathbf{Y'}_{\lambda}$	bold Roman	normalised graph predicted result vector from NN	$\mathbb{R}^{n^{\lambda}\times o^{\lambda}}$
$\mathbf{y}^{\lambda}_{\alpha}$	bold Roman	node predicted result vector for atom α from NN in level-λ graph	$\mathbb{R}^{o^{\lambda}}$
${\mathbf{y}'}^{\lambda}_{\alpha}$	bold Roman	normalised node predicted result vector for atom α from NN in level-λ graph	$\mathbb{R}^{o^{\lambda}}$
$\mathrm{y}^{\lambda}_{\alpha}$	Roman	predicted result from node α in level-λ graph	$\mathbb{R}$
${\mathrm{y}'}^{\lambda}_{\alpha}$	Roman	normalised predicted result from node α in level-λ graph	$\mathbb{R}$
Z_i	Roman	the atomic number of atom i	$\mathbb{N}^*$
α	Roman	indexing variable	$\mathbb{N}$ or specified domain
Γ	Roman	the GLaCE operation	N/A
γ	Roman	angle indicator	N/A
$\varepsilon^{\lambda}_{\alpha}$	Roman	the bond set of node α in level-λ graph	N/A
ζ_i	Roman	the number of valence electron for atom i	$\mathbb{N}^*$
θ	Roman	angle indicator	N/A
λ	Roman	the index of level of graph in GLaCE	$\mathbb{N}^*$

Notation	Typeset	Description	Domain
$\mu_j^{\lambda,\text{preset}}$	Roman	the average value of final feature j in level-λ graph based on the training dataset	$\mathbb{R}$
$\mu_{\text{y}}^{\lambda,\text{preset}}$	Roman	the average value of true value in level-λ graph based on the training dataset	$\mathbb{R}$
ν_α^λ	Roman	the atom set of node α in level-λ graph	N/A
Ξ_λ	Roman	the interaction function for level-λ	N/A
σ_λ	Roman	the activation function of the outpur layer in the NN of level-λ	N/A
$\sigma_j^{\lambda,\text{preset}}$	Roman	the standard deviation of final feature j in level-λ graph based on the training dataset	$\mathbb{R}$
$\sigma_{\text{y}}^{\lambda,\text{preset}}$	Roman	the standard deviation of true value in level-λ graph based on the training dataset	$\mathbb{R}$
Φ_λ	Roman	the node formation function for level-λ	N/A
$\varphi_{j,\alpha}^\lambda$	Roman	the constant feature j of node α in graph level-λ	$\mathbb{R}$
χ_i	Roman	the Pauling-scaling electronegativity of atom i	$\mathbb{R}$

Appendix C

Selected Molecules in GDB-13-based Datasets

Readers are noted that, the data listing here refers to those molecules that successfully pass the MD optimisation and the DFT calculation. The instruction of obtaining original GDB-13.CNO subset is presented in Chapter 7. The following strings and numbers mean the name of the SMILES files and the numbers of lines that the molecules are in (in GDB-13 dataset, each SMILES string occupies one line in file).

E.g., string "cno4_[2-5]" denotes that our dataset contains the molecules from line 2 (inclusive) to line 5 (inclusive) in file "cno4.smi".

C.1 Training Set

Corresponding datasets: Mulliken_Train, Hirshfeld_Train, CM5_Train, DDEC6_Train and Struct_Train.

cno4_[2-5]

cno5_[1-5, 7-10, 12-15, 17-22, 24, 27, 29, 31]

cno6_[1-2, 4-13, 16, 18-22, 24-30, 32, 34, 36-37, 39-41, 43-45, 48, 50-51, 54, 56, 59-62, 64, 66-67, 69-73, 76-79, 81-84, 87-90, 93-96, 98, 100, 102-107, 109-116, 118-124, 126-128, 130, 132-136, 138-139, 144-145, 147, 150-156, 159, 161-166, 168, 171, 173, 175, 177, 180-184, 186-188, 190, 192]

cno7_[2-3, 5, 7-11, 13-19, 21-22, 24-25, 27-29, 31, 34, 36-37, 39, 41-44, 47-59, 61-64, 66, 68, 70-72, 74-77, 79-92, 94-95, 97-98, 100-102, 104-105, 107, 109, 111-115, 117-120, 122-123, 125-127, 130-132, 136, 140-142, 145-151, 153-154, 156-157, 159, 161, 163, 165-168, 171-174, 176, 179-181, 183, 185, 187-188, 190, 192-193, 195-197, 199-200, 202-208, 211-214, 216-218, 220, 224, 226-228, 230, 233, 235-236, 238-239, 241-245, 247, 250-254, 256-257, 259-261, 263-265, 268-272, 274, 276, 278-279, 283, 285-288, 290-296, 300, 302-307, 309-310, 312, 314-316, 318-321, 324-327, 330-334, 336-337, 340, 342, 344-345, 347-348, 350, 352-353, 355, 357-361, 363, 365, 367-368, 371-372, 374, 376, 379-381, 383-384, 386-389, 392, 394-397, 399, 403-410, 412, 414-415, 418-424, 426-427, 430-432, 434-435, 441-444, 446-450, 453, 455, 457, 459, 461, 463-467, 471, 473, 476, 478-487, 489, 495, 497-510, 513, 515-516, 518-520, 523-526, 528-533, 535-544, 546-547, 549-551, 553-563, 565-569, 571, 575, 577, 579, 581-589, 591-597, 600-605, 609-612, 615-619, 621, 623-630, 634-635, 637-642, 644, 646, 648-650, 652-654, 657-658, 660-664, 669-670, 673-686, 688-689, 691-693, 696, 698-702, 706-711, 713, 715-716, 718-720, 722, 724, 727, 729-730, 732, 734-735, 737-749, 752-753, 755-756, 758, 760-761, 765, 767-769, 772-774, 776-779, 781-789, 791-794, 796, 798-799, 805, 807-812, 814-816, 818, 820, 823-825, 827, 831-833, 836, 838-840, 842-843, 846-848, 850-855, 857, 859-860, 863-864, 866-868, 871-874, 878-880, 882-884, 886-888, 891-892, 896-899, 901, 903-906, 908, 910, 912, 914-916, 918, 921, 924-928, 930-937, 939-942, 944-950, 952-954, 956-962, 965-967, 969-976, 978-979, 981-993, 995, 998-1003, 1005, 1007-1013, 1018-1021, 1023, 1025, 1028, 1030, 1033-1034, 1036, 1038-1042, 1044-1045, 1048-1049, 1051, 1053-1057, 1060-1067, 1069-1073, 1076, 1078, 1081-1084, 1087-1100, 1102, 1105-1106, 1108-1110, 1112-1116, 1118-1119, 1121-1125, 1127-1129, 1132-1133, 1135-1136, 1138-1139, 1141-1143, 1145-1146, 1149-1153, 1156, 1158, 1161, 1164-1165, 1167-1169, 1171, 1175, 1177, 1180, 1182, 1186-1187, 1190, 1193, 1196-1198, 1200, 1202-1209, 1211-1219, 1222-1228, 1230, 1232-1234, 1238, 1241-1242, 1244, 1247, 1249-1252, 1255-1257, 1259, 1261, 1263, 1265-1266, 1272, 1274, 1276-1277, 1279-1280, 1283-1284, 1286-1288, 1290-1291, 1293-1294, 1298, 1301-1302, 1304, 1307-1309, 1311-1312, 1314-1316, 1318, 1320-1324, 1326, 1328-1329, 1331-1332, 1335-1338, 1340-1344, 1348-1350, 1354-1359, 1363-1366, 1369, 1372-1375, 1377, 1379-1381, 1383, 1386-1390, 1393, 1395-1396, 1398, 1400-1402, 1405, 1409-1413, 1415, 1417-1424, 1427, 1429-1437, 1440, 1442-1448, 1450-1455, 1457-1458, 1460-1461,

1463-1464, 1466, 1468-1474, 1476, 1478-1479, 1481-1482, 1485-1486, 1488, 1490, 1492-1498, 1502-1505, 1508-1510, 1514-1515, 1519-1520, 1522-1527, 1529-1530, 1533, 1537, 1539-1543, 1545-1547, 1549, 1551-1555, 1558-1564, 1566-1571, 1575-1579, 1581, 1583-1585, 1587-1594, 1596-1597, 1599-1601, 1604, 1606-1608, 1610-1612, 1615, 1617-1625, 1627-1635, 1637-1644, 1646, 1649-1652, 1655-1656, 1659-1660, 1662, 1664, 1666, 1668-1671, 1674-1676, 1678-1680, 1682, 1684-1686, 1688-1694, 1696-1697, 1699-1704, 1706-1711, 1713-1714, 1717, 1720-1721, 1723-1724, 1728-1731, 1734-1739, 1741-1744, 1746-1750, 1752-1753, 1755-1756, 1758-1760, 1762, 1764-1772, 1774, 1776-1780, 1782-1789, 1792-1794, 1796-1797, 1799-1800, 1802-1810, 1813-1820, 1825-1828, 1831-1836, 1838-1847, 1850, 1852, 1854, 1857-1861, 1863-1866, 1868, 1870-1877, 1881-1883, 1885-1888, 1892-1897, 1901-1908, 1910-1911, 1913-1916, 1919-1928, 1930-1940, 1943-1944, 1947-1951, 1955, 1959, 1961-1970, 1972-1973, 1975, 1977-1980, 1982-1983, 1985-1990, 1992, 1994-1998, 2000-2002, 2005-2006, 2008, 2010-2013, 2015-2018, 2021-2022, 2024-2025, 2027, 2029-2032, 2034, 2036, 2038-2042, 2045-2047, 2049-2053, 2056-2059, 2061-2070, 2072-2074, 2077-2081, 2083-2084, 2086-2091, 2094-2095, 2097, 2099-2112, 2115, 2117, 2121-2125, 2128-2129, 2131-2137, 2140-2148, 2150, 2152-2155, 2157, 2159, 2161, 2163-2166, 2170-2171, 2173-2177, 2179, 2181-2186, 2189-2195, 2197, 2199-2202, 2204-2205, 2207, 2209, 2214, 2216-2218, 2220-2221, 2223-2224, 2226-2231, 2233-2235, 2238-2246, 2248-2252, 2254-2257, 2259-2262, 2265-2267, 2270-2281, 2283, 2286-2289, 2292-2298, 2300-2302, 2305-2312, 2314, 2317, 2320-2325, 2327-2334, 2336-2337, 2340, 2342, 2344-2346, 2348, 2351]

cno8_[3, 6-11, 14-15, 17-18, 21-26, 28-32, 35, 37, 39-40, 42-44, 47-50, 52, 55, 57-58, 63-64, 66, 68, 70, 74-79, 81-83, 85, 87-89, 91-94, 96-98, 103-104, 106-110, 112-118, 121-123, 125-126, 130-134, 136, 138-139, 141, 144-145, 147-149, 151-152, 154-158, 160-161, 164-166, 169-172, 175-176, 178, 180-181, 183-184, 186-190, 193, 195-196, 200, 202-203, 205-207, 209, 211-217, 219-220, 222-228, 230-237, 239-240, 242, 244, 247-249, 251-255, 259-264, 266-270, 273-275, 277, 279, 281-296, 298-310, 315-317, 319, 322-323, 326, 329, 334-338, 341-342, 344-348, 350, 352-354, 356, 358-362, 367-368, 372-373, 375-381, 383-388, 390-393, 395-400, 403, 405-407, 409, 412-415, 417, 419-421, 423, 425-426, 432-433, 435-436, 438-439, 441-444, 446, 448-450, 452-454, 456-458, 460, 463-465, 467, 469-474, 480-486, 488, 493, 497, 499-500, 504-505, 507-513, 515, 517, 519-522, 524, 526-529, 531, 533, 536, 540, 543-547, 549, 551-557, 559-560, 562-567, 569, 571, 573-584, 586-587, 589, 591-594, 596-600, 602-605, 607-615, 617-618, 620-634, 638-640, 642-643, 645-647, 649, 651, 653-654, 656-657, 659, 661-663, 667-668, 671-684, 686-688, 690-691, 694, 696-699, 701-702, 705-706, 708-714, 717-724, 726-727, 729, 731, 733-735, 738, 740-742, 744-746, 748-749, 751-753, 756-757, 759, 761-764, 767, 769-770, 772-776, 778-779, 781-782, 785-790, 792, 794-797, 799-803, 807-809, 811, 813-816, 819-820, 822, 824-827, 829-830, 832, 835-836, 838-840, 843-845, 848, 850-851, 853-866, 869, 871-872, 876, 878-883, 885, 887-893, 895-898, 902-904, 906-910, 912-915, 917, 920-925, 927-930, 935-942, 944, 946-952, 954, 957-963, 965-971, 973, 975-977, 979-983, 986-988, 990, 993, 995-996, 998-1000, 1002-1006, 1008-1015, 1017, 1019-1021, 1023-1026, 1028-1034, 1036-1042, 1044-1045, 1048-1054, 1056-1058, 1060, 1062-1064, 1066-1067, 1069-1074, 1077-1079, 1084-1086, 1088, 1091, 1093-1095, 1097, 1099, 1101-1106, 1108, 1111, 1113-1114, 1116-1121, 1123-1124, 1126, 1128-1135, 1137, 1140-1141, 1143-1148, 1150-1155, 1157, 1159, 1161, 1168, 1171-1175, 1177-1180, 1182-1185, 1189, 1191-1196, 1198-1200, 1204, 1207, 1210-1211, 1214, 1218-1219, 1222, 1224-1228, 1230, 1232-1236, 1238-1240, 1242-1252, 1254-1256, 1258, 1260-1262, 1264-1270, 1275-1277, 1279-1288, 1290, 1293, 1296-1298, 1301-1302, 1305-1310, 1312-1314, 1316-1326, 1328-1329, 1331-1333, 1337-1338, 1340-1341, 1344-1346, 1348-1354, 1356-1357, 1359-1360, 1362-1369, 1371-1375, 1377-1378, 1380-1387, 1389-1390, 1393-1396, 1398, 1400, 1402, 1405, 1407, 1409, 1411, 1413-1414, 1416-1417, 1419, 1422-1423, 1425-1428, 1431, 1433, 1436-1441, 1443-1444, 1448-1451, 1456-1459, 1461, 1463, 1465, 1469, 1471, 1473, 1475-1481, 1483, 1485-1486, 1488-1489, 1491-1494, 1496-1497, 1499-1501, 1504-1507, 1511, 1513-1514, 1517-1523, 1525-1532, 1535, 1537-1539, 1541, 1543-1544, 1546-1548, 1551, 1553, 1555-1558, 1560, 1564, 1567-1572, 1574-1579, 1583-1591, 1593-1600, 1604, 1606-1612, 1614, 1616-1617, 1619, 1621-1622, 1625-1629, 1631-1632, 1634, 1637-1639, 1641-1642, 1644-1650, 1652-1655, 1657-1660, 1662-1664, 1666-1676, 1678-1683, 1685-1686, 1688-1693, 1695-1697, 1699-1703, 1708-1709, 1711, 1713-1715, 1719-1722, 1724, 1726-1734, 1736, 1738-1745, 1747-1751, 1753-1756, 1758-1759, 1761-1772, 1774-1775, 1778-1780, 1782-1783, 1785-1788, 1791-1794, 1796-1801, 1803-1808, 1811-1816, 1818, 1821, 1823, 1825-1827, 1830-1832, 1834-1835, 1837-1840, 1842-1843, 1846, 1848-1852, 1854, 1856-1858, 1861-1863, 1865-1867, 1869-1870, 1872-1880, 1882-1884, 1888-1890, 1892-1894, 1896-1899, 1902-1903, 1905, 1907-1910, 1912, 1914-1915, 1917-1923, 1927-1928, 1930, 1932-1934, 1936-1940, 1942, 1945, 1947-1950, 1952-1954, 1957, 1959-1962, 1965, 1968-1973, 1976, 1978-1981, 1984-1989, 1991-1996, 1998, 2000, 2002-2011, 2014-2023, 2025-2030, 2033-2052, 2056-2059, 2061-2067, 2069-2074, 2076, 2079-2080, 2082-2083, 2087-2090, 2092-2093, 2095, 2097-2098, 2100, 2102, 2105-2107, 2109-2113, 2115-2117, 2121-2126, 2128-2134, 2136, 2139-2143, 2145, 2147-2149, 2151, 2153-2157, 2160-2161, 2165, 2167-2170, 2175, 2177-2181, 2183, 2185-2186, 2188-2192, 2196-2197, 2199-2202, 2204-2206, 2210-2212, 2217-2218, 2222-2228, 2230-2231, 2235-2239, 2241-2242, 2244-2245, 2248-2249, 2251, 2253-2254, 2256-2258, 2261, 2263-2264, 2266, 2270, 2273-2283, 2285, 2288-2290, 2292, 2295-2297, 2299-2303, 2305-2306, 2308-2309, 2311-2312, 2314-2317, 2319-2322, 2324-2328, 2330-2337, 2340, 2342-2355, 2358, 2360, 2362-2365, 2367, 2369-2370, 2372-2378, 2380-2381, 2385-2388, 2390-2392, 2394, 2398, 2400-2402, 2404, 2406-2414, 2418-2421, 2423-2429, 2431-2432, 2434-2439, 2441-2444, 2446-2448, 2450-2454, 2456-2459, 2462-2464, 2466, 2468, 2470, 2473-2477, 2479, 2481-2482, 2485, 2487-2488, 2491-2493, 2496-2500, 2502-2503, 2505, 2507-2510, 2512-2513, 2515-2521, 2523-2524, 2531-2532, 2535-2539, 2541-2543, 2545-2547, 2549-2550, 2553, 2556-2557, 2559-2565, 2570, 2572-2583, 2585-2587, 2589-2592, 2594-2597, 2599, 2601, 2603-2604, 2608-2610, 2612, 2615-2616, 2618, 2620-2622, 2625-2628, 2630-2632, 2634-2637, 2642-2643, 2645-2646, 2649-2652, 2654-2657, 2659-2660, 2662-2663, 2665, 2667, 2669-2672, 2676-2684, 2686-2689, 2691-2692, 2694-2696, 2698-2700, 2702-2706, 2708-2709, 2711-2714, 2716-2720, 2722-2723, 2725-2727, 2729, 2731-2733, 2735, 2737, 2739-2740, 2743-2744, 2748-2754, 2756-2759, 2761, 2763, 2766-2773, 2775-2780, 2783-2785, 2788, 2792-2796, 2800-2805, 2808-2811, 2814-2816, 2818-2819, 2821-2822, 2824-2825, 2830-2833, 2835-2841, 2843, 2846-2849, 2851, 2854-2857, 2859, 2861-2863, 2866-2869, 2872, 2874-2877, 2879-2880, 2883, 2887-2891, 2893-2898, 2900-2901, 2903, 2905-2906, 2910-2911, 2915-2917, 2919, 2921-2923, 2926-2927, 2929-2933, 2935-2936, 2938-2940, 2942-2943, 2945-2954, 2956-2963, 2965, 2967-2975, 2977-2981, 2983-2991, 2994-2996, 2998-3000, 3004, 3006, 3008-3010, 3016-3018, 3020-3021, 3024-3025, 3027-3032, 3034-3039, 3041-3044, 3046-3055, 3059-3061, 3063, 3065, 3067-3070, 3072-3074, 3076-3080, 3082-3088, 3091-3092, 3094-3095, 3097-3098, 3101-3104, 3106, 3109-3126, 3128-3129, 3131-3135, 3137-3139, 3142, 3144, 3146-3148, 3152-3153, 3158-3159, 3162, 3165-3166, 3168, 3170-3171, 3176, 3178, 3181, 3183-3189, 3191-3193, 3195, 3197, 3199-3200, 3202-3205, 3207, 3209, 3212-3220, 3222, 3224-3226, 3228, 3230, 3233, 3235-3236, 3238-3241, 3244, 3248-3254, 3256-3259, 3261, 3264, 3266-3267, 3269-3270, 3272, 3275-3277, 3279-3280, 3282, 3284-3286, 3288, 3290-3295, 3299-3300, 3302, 3304, 3306-3308, 3310, 3313-3314, 3316-3317, 3319, 3321-3323, 3325-3326, 3328-3329, 3331-3344, 3347, 3349-3351, 3354-3355, 3357-3358, 3361-3369, 3371-3380, 3382-3388, 3391, 3393-3396, 3399-3400, 3402-3409, 3411, 3414-3416, 3420-3421, 3424-3425, 3427-3435, 3437-3439, 3441-3442, 3444-3446, 3450-3452, 3454-3456, 3458-3460, 3465-3469, 3471-3476, 3478, 3481, 3483-3487, 3489-3490, 3493-3496, 3498, 3500, 3503-3505, 3507-3508, 3510-3513, 3515-3518, 3520-3524, 3527, 3530, 3534-3535, 3537-3545, 3548, 3550-3558, 3560, 3562-3563, 3565, 3570, 3573-3574, 3576-3579, 3581-3584, 3586, 3588-3592, 3594, 3596-3599, 3602-3605, 3609-3615, 3617-3619, 3622-3623, 3626-3634, 3636-3639, 3641, 3643, 3645-3646, 3648-3649, 3651-3653, 3655-3659, 3662-3667, 3669, 3671, 3673-3678, 3680-3681, 3683-3684, 3686-3687, 3689, 3692, 3695, 3697, 3699-3701, 3706, 3710-3711, 3713-3715, 3717-3724, 3726, 3728, 3730-3736, 3738-3744, 3746-3750, 3752-3753, 3757-3759, 3762-3763, 3767-3773, 3775-3778, 3780-3787, 3789-3790, 3792-3794, 3797, 3799-3800, 3802-3805, 3809-3812, 3814, 3816-3821, 3823-3827, 3829-3831, 3833, 3835-3837, 3841-3842, 3844, 3846-3850, 3855-3856, 3860-3864, 3866, 3868, 3870-3875, 3879-3880, 3882-3886, 3888-3890, 3892-3901, 3906-3908, 3910, 3912-3915, 3919, 3921-3927, 3930-3936, 3938-3940, 3942-3943, 3946-3952, 3954-3956, 3958-3961, 3963-3965, 3967-3971, 3973, 3975, 3978, 3981-3986, 3988, 3990-3992, 3994-3995, 3997, 3999-4003, 4005-4007, 4009-4010, 4012-4016, 4018, 4025, 4027-4031, 4036, 4039, 4041-4044, 4046-4048, 4050, 4052, 4054-4058, 4061-4063, 4065-4068, 4071, 4073, 4077-4080, 4085-4089, 4091, 4093, 4095-4096, 4099-4101, 4103-4109, 4111-4112, 4114, 4117-4119, 4121-4131, 4133, 4136-4139, 4141-4145, 4150-4151, 4153, 4155-4162, 4165-4175, 4177-4178, 4181, 4183, 4185-4187, 4189-4192, 4195-4198, 4200-4202, 4204-4205, 4207, 4209-4216, 4219-4221, 4223-4224, 4228-4235, 4237, 4239, 4241, 4243-4244, 4248-4250, 4253-4260, 4263, 4268-4274, 4276-4279, 4283-4289, 4291, 4293-4294, 4297, 4299-4307, 4310-4312, 4315-4325, 4328-4329, 4332-4336, 4338-4339, 4341-4344, 4347, 4349-4358, 4360-4362, 4364, 4366-4371, 4373-4375, 4377, 4379, 4381-4387, 4389-4390, 4392-4399, 4403, 4406, 4409-4412, 4414-4417, 4419-4421, 4423, 4426-4432, 4434, 4438-4439, 4441, 4443-4444, 4446-4447, 4449, 4452-4454,

4456-4459, 4461-4464, 4466-4475, 4477-4478, 4480-4483, 4485-4490, 4492-4493, 4496-4497, 4499-4502, 4504-4505, 4507, 4509-4510, 4512-4514, 4517-4518, 4520-4521, 4524-4526, 4528-4537, 4539, 4541-4547, 4550, 4552-4557, 4559-4560, 4564, 4566, 4568-4571, 4574-4575, 4577-4579, 4582, 4584, 4586-4597, 4599-4601, 4603, 4605, 4610-4612, 4614-4615, 4617, 4619-4622, 4624-4625, 4627-4628, 4630-4635, 4637-4651, 4657, 4659-4661, 4663-4666, 4669-4671, 4673, 4676-4684, 4687, 4690, 4694, 4696-4697, 4699, 4701-4706, 4708-4712, 4714, 4716-4724, 4727, 4729-4731, 4733-4736, 4739, 4742-4745, 4747-4750, 4752-4753, 4755-4756, 4758-4763, 4765-4769, 4771-4772, 4774-4775, 4777-4781, 4783-4784, 4786-4790, 4793-4804, 4807, 4811-4822, 4824, 4828-4829, 4831-4837, 4839-4842, 4844, 4848-4858, 4860, 4862-4863, 4865, 4867-4881, 4883, 4887, 4891-4897, 4899-4901, 4903-4906, 4908-4912, 4916-4917, 4920-4921, 4923-4924, 4926-4929, 4931, 4933, 4939-4944, 4946-4947, 4949-4950, 4952-4956, 4958-4964, 4966-4974, 4976-4980, 4983-4984, 4986-4987, 4989-4990, 4992-4993, 4995, 4998, 5001, 5004, 5006, 5008-5011, 5015, 5019-5021, 5024, 5026-5027, 5029-5030, 5032-5033, 5035-5038, 5040-5041, 5043-5044, 5046-5052, 5057, 5059, 5062-5064, 5067-5069, 5072-5075, 5077-5078, 5080, 5082-5088, 5090, 5093-5094, 5096-5097, 5101-5103, 5105, 5107-5116, 5118-5121, 5123-5124, 5128-5129, 5133-5141, 5143-5148, 5150-5153, 5155, 5157-5160, 5162-5163, 5165-5167, 5169, 5172-5174, 5177-5179, 5181, 5186, 5189, 5191, 5196-5197, 5199-5205, 5208-5209, 5211-5215, 5217, 5220-5221, 5223-5224, 5226-5227, 5231, 5233-5236, 5238-5242, 5244, 5246-5249, 5251-5254, 5256-5259, 5261-5262, 5264, 5266-5267, 5271-5279, 5281-5282, 5284, 5286-5298, 5301-5303, 5305-5308, 5310-5315, 5317-5318, 5320, 5322, 5324-5326, 5328, 5330-5331, 5333-5339, 5341-5349, 5351-5354, 5358-5363, 5365-5369, 5371-5375, 5378, 5380-5381, 5383-5388, 5390-5394, 5396-5403, 5405-5414, 5416, 5418-5419, 5423-5425, 5428-5429, 5431-5436, 5438-5439, 5443, 5447-5452, 5454-5456, 5461-5462, 5464-5465, 5467, 5470-5474, 5476-5477, 5479, 5481, 5483-5485, 5487-5492, 5494-5499, 5501-5505, 5507, 5509-5513, 5516-5517, 5519-5529, 5531-5532, 5536-5539, 5541-5545, 5547, 5549-5556, 5558-5559, 5561-5563, 5565, 5567, 5569-5570, 5572-5575, 5580, 5582-5583, 5585, 5587-5588, 5590-5592, 5596-5597, 5599-5600, 5602-5607, 5610-5611, 5614-5617, 5621-5622, 5624-5632, 5636, 5638, 5640, 5643-5646, 5648-5649, 5652-5653, 5656-5657, 5659-5661, 5663-5666, 5668, 5670, 5672-5674, 5676-5678, 5680-5686, 5688-5689, 5694, 5696, 5698-5699, 5701, 5703, 5706, 5711-5718, 5721, 5723, 5727, 5730, 5732-5734, 5736, 5738-5741, 5743-5745, 5747, 5749, 5751, 5754-5759, 5763-5764, 5766-5772, 5775-5777, 5779-5791, 5793, 5795-5800, 5802-5806, 5808-5809, 5811-5812, 5814-5820, 5824, 5826-5827, 5831-5835, 5839-5843, 5847-5854, 5857-5858, 5860, 5862-5864, 5866, 5868-5875, 5877-5879, 5883, 5887-5892, 5894-5895, 5898-5899, 5903-5905, 5907-5908, 5911-5918, 5920-5921, 5923, 5926-5941, 5945-5949, 5951-5960, 5962, 5964-5967, 5969-5970, 5972-5977, 5979-5984, 5987-5988, 5990, 5992-5993, 5995-6000, 6002-6003, 6005, 6007-6013, 6015, 6017-6019, 6022, 6026, 6028-6033, 6035, 6038, 6040, 6042-6043, 6045-6047, 6049-6056, 6059-6060, 6063-6064, 6066-6068, 6071, 6073-6074, 6076-6081, 6083-6085, 6087-6088, 6090-6095, 6097, 6099-6102, 6104, 6106-6108, 6110-6111, 6114-6116, 6118-6119, 6121, 6123-6124, 6130, 6132-6133, 6135-6137, 6139, 6142-6144, 6147-6148, 6151-6154, 6158-6160, 6162-6168, 6170, 6172, 6174-6176, 6178-6180, 6182-6183, 6185-6191, 6193-6197, 6199-6201, 6203-6204, 6206-6207, 6209, 6211, 6214-6215, 6217-6222, 6224, 6226-6228, 6232-6234, 6236-6240, 6242-6247, 6249, 6251-6255, 6258-6260, 6262-6265, 6267, 6269-6273, 6275-6276, 6278-6285, 6287, 6289-6292, 6294-6302, 6304-6306, 6309-6317, 6319, 6324-6325, 6327, 6334-6340, 6342, 6344, 6347, 6349-6350, 6352-6356, 6358-6360, 6363-6367, 6370, 6373-6374, 6377-6378, 6380-6384, 6387-6388, 6392-6393, 6395-6399, 6401-6405, 6407, 6409, 6411-6417, 6420, 6423, 6425-6426, 6428, 6430, 6434-6435, 6437-6444, 6446-6449, 6451-6452, 6454, 6458-6461, 6464-6467, 6469-6474, 6476, 6478, 6480-6481, 6483, 6485, 6487-6490, 6492-6494, 6496-6500, 6502-6503, 6506-6507, 6509, 6512-6513, 6516, 6518-6523, 6525-6527, 6532, 6534-6536, 6538-6539, 6541-6543, 6546, 6550-6556, 6560-6563, 6565-6570, 6574-6576, 6578, 6580-6588, 6590-6596, 6598-6601, 6605, 6607, 6609-6614, 6617-6618, 6621-6626, 6628-6630, 6633, 6635, 6637, 6639-6641, 6644-6647, 6649-6653, 6655-6656, 6661, 6663-6665, 6667-6671, 6673, 6675-6678, 6680-6682, 6684-6685, 6688, 6690-6692, 6694-6700, 6703, 6705, 6709-6710, 6713-6714, 6717-6728, 6731-6732, 6735-6737, 6741-6746, 6748-6751, 6753, 6755-6762, 6765, 6767-6768, 6772, 6777-6780, 6782-6783, 6788-6789, 6792-6796, 6798-6801, 6803-6804, 6806-6807, 6810, 6813-6815, 6817, 6819, 6823-6828, 6831-6835, 6837, 6839, 6842-6844, 6846, 6849-6853, 6856-6857, 6859-6862, 6866, 6868-6871, 6874-6875, 6877-6878, 6880-6884, 6886-6890, 6893-6896, 6898-6901, 6903, 6907-6909, 6911-6914, 6916-6920, 6922-6923, 6925-6928, 6930-6933, 6936-6941, 6944-6946, 6948, 6950, 6952, 6954-6958, 6962-6963, 6965-6970, 6972, 6975-6978, 6982, 6985-6990, 6992-6993, 6995, 6998, 7000-7002, 7004-7006, 7008-7010, 7013-7017, 7019-7029, 7032-7033, 7035, 7037, 7039-7045, 7047-7050, 7052, 7054, 7057-7061, 7063-7065, 7067-7069, 7073-7075, 7077-7080, 7082-7083, 7086, 7088-7091, 7093-7097, 7099-7100, 7102-7103, 7105, 7107-7108, 7111-7113, 7117, 7119-7127, 7129-7133, 7135, 7137-7141, 7143-7148, 7150, 7152-7154, 7156-7162, 7166, 7168, 7170-7171, 7173-7175, 7179, 7182-7185, 7187-7188, 7190-7193, 7195-7198, 7200, 7203-7205, 7208-7209, 7211, 7213, 7217, 7223-7226, 7228-7230, 7232-7238, 7240-7241, 7244, 7246-7248, 7250-7252, 7254-7255, 7258-7259, 7261-7262, 7264-7267, 7270-7272, 7274-7275, 7279-7280, 7282-7283, 7286, 7289-7290, 7292-7296, 7301-7304, 7307-7308, 7310, 7312-7322, 7324-7325, 7328, 7330-7331, 7335-7338, 7341, 7343-7347, 7349-7350, 7354-7358, 7361, 7363, 7365-7368, 7370, 7372, 7374, 7378-7379, 7381, 7385-7389, 7391-7394, 7396-7401, 7404-7406, 7408-7411, 7413-7418, 7421, 7426, 7428-7431, 7433-7448, 7450-7453, 7455-7458, 7460, 7464-7467, 7471-7472, 7474-7476, 7481-7483, 7485-7486, 7488-7491, 7493-7504, 7507, 7510, 7512-7518, 7520, 7522-7525, 7528-7530, 7532-7534, 7536-7539, 7541, 7543, 7545-7546, 7548-7549, 7551-7552, 7555-7557, 7559, 7562-7563, 7565-7568, 7570-7573, 7575-7577, 7579-7581, 7583, 7585-7589, 7591-7592, 7594-7596, 7600-7608, 7610, 7612-7617, 7619, 7622-7640, 7642-7643, 7645, 7647-7648, 7650-7652, 7654-7656, 7658-7659, 7662-7667, 7669, 7671, 7673-7676, 7678, 7681-7683, 7686-7688, 7690, 7692-7693, 7696, 7698, 7700-7702, 7704-7713, 7715-7723, 7725-7732, 7734-7737, 7742-7744, 7746-7747, 7750-7758, 7760, 7762, 7765-7769, 7771-7772, 7775-7778, 7784-7785, 7787-7790, 7794-7798, 7800-7803, 7805-7814, 7816-7819, 7821-7824, 7826, 7828-7829, 7832, 7834-7837, 7839-7845, 7850-7852, 7854-7855, 7857-7860, 7862-7864, 7866-7867, 7869-7872, 7874, 7877-7878, 7883, 7885-7886, 7888, 7890, 7893, 7895-7899, 7901-7905, 7907, 7909, 7911-7912, 7914-7918, 7920, 7924-7928, 7930-7934, 7939-7943, 7946-7947, 7951-7957, 7959, 7962-7963, 7965-7966, 7969-7975, 7980-7981, 7983-7985, 7987-7993, 7995-8002, 8006, 8008-8012, 8014-8018, 8020-8022, 8024, 8026-8029, 8031-8035, 8037, 8040-8043, 8045, 8047-8050, 8052, 8056-8057, 8060-8063, 8066-8069, 8071, 8073, 8075-8077, 8079-8085, 8089-8099, 8102, 8104, 8106, 8108, 8110, 8112-8113, 8116-8117, 8120, 8122-8127, 8129-8131, 8134-8137, 8139-8147, 8150-8156, 8159-8162, 8164-8171, 8173-8174, 8177-8178, 8180, 8182-8184, 8186-8187, 8190-8194, 8196, 8198-8199, 8201, 8204-8206, 8208-8209, 8212-8213, 8215-8216, 8218-8220, 8223-8231, 8234, 8239, 8241-8243, 8245, 8247-8248, 8250, 8252-8258, 8260-8264, 8266, 8269, 8271-8274, 8276-8281, 8285-8287, 8289-8290, 8292-8294, 8296-8303, 8305-8310, 8312, 8314-8315, 8317-8320, 8324-8325, 8327-8329, 8331-8333, 8335-8336, 8338-8342, 8345, 8347-8349, 8351-8354, 8356, 8358-8363, 8365-8367, 8369, 8371, 8373, 8375, 8377-8381, 8384-8388, 8390-8391, 8393-8396, 8398-8399, 8401-8402, 8404-8409, 8411-8421, 8423, 8425-8429, 8433, 8437, 8439, 8441-8442, 8444-8445, 8447, 8449-8462, 8464-8471, 8473-8476, 8478, 8481, 8484-8485, 8488, 8490, 8494-8497, 8500-8504, 8506-8509, 8511-8514, 8516, 8520, 8522, 8524-8527, 8529, 8531-8533, 8535-8541, 8543-8554, 8557-8560, 8562-8566, 8568-8569, 8571-8572, 8574-8576, 8578-8584, 8586-8589, 8592-8594, 8596, 8598-8600, 8603-8604, 8606, 8611-8612, 8616-8617, 8619-8622, 8625, 8627, 8630-8631, 8633-8635, 8638, 8642-8646, 8649, 8651-8653, 8657-8659, 8661, 8664, 8666-8670, 8672-8675, 8677-8678, 8680, 8683, 8685, 8687, 8689, 8691-8692, 8694-8696, 8698-8702, 8704-8707, 8709, 8713-8721, 8723-8734, 8736, 8738-8740, 8742-8744, 8749-8750, 8753, 8755, 8757-8763, 8765-8766, 8768-8772, 8774-8777, 8779-8783, 8790-8791, 8793-8794, 8796-8799, 8801-8808, 8811, 8815-8816, 8820, 8823-8824, 8826-8829, 8831, 8836-8847, 8851, 8853-8854, 8856-8858, 8860, 8862-8866, 8868, 8870, 8872-8874, 8878-8882, 8884-8885, 8888-8890, 8892, 8894-8901, 8904-8914, 8916, 8918-8919, 8922-8928, 8930-8932, 8934, 8938, 8940-8942, 8944-8945, 8947-8949, 8951-8952, 8954-8960, 8964-8969, 8971-8972, 8975-8984, 8986, 8988, 8990-8996, 8998, 9000-9005, 9007-9011, 9013, 9017-9025, 9027, 9029-9031, 9033-9041, 9043-9045, 9047, 9050-9052, 9054, 9058, 9060-9061, 9064-9066, 9068, 9070-9072, 9075-9077, 9079, 9082-9083, 9086, 9088, 9090, 9092-9102, 9106, 9108, 9110-9113, 9115-9116, 9118-9124, 9126, 9128-9131, 9133, 9135, 9137-9138, 9141, 9143-9145, 9147-9148, 9152-9154, 9157-9158, 9161-9163, 9165-9166, 9168-9173, 9177, 9180-9190, 9193, 9195-9196, 9198-9202, 9204, 9207-9208, 9210-9213, 9215-9216, 9218-9222, 9224-9225, 9227-9230, 9233, 9235, 9237-9239, 9242, 9244, 9247-9248, 9253, 9255-9258, 9260-9261, 9263, 9265-9268, 9270, 9272-9273, 9276-9279, 9284-9286, 9288-9289, 9292-9295, 9297-9298, 9302-9303, 9305-9306, 9308-9309, 9311, 9314-9324, 9326-9327, 9331-9335, 9340-9345, 9347-9348, 9351, 9353-9354, 9356, 9358, 9360-9361, 9365-9367, 9369-9370, 9372-9373, 9376-9379, 9381-9387, 9389-9391, 9395-9400, 9403-9418, 9420-9421, 9424-9425, 9428-9430, 9437-9440, 9442-9448, 9450, 9452-9453, 9455, 9457-9460, 9462-9464, 9466-9470, 9474, 9477-9478, 9481-9482, 9484-9486, 9488, 9490-9491, 9493-9501, 9506-9509, 9511-9513, 9515-9522, 9529-9530, 9532-9535, 9539-9545, 9548, 9550-9551, 9553-9554, 9558-9566, 9568, 9571, 9574-9577, 9579-9581, 9583-9584, 9587-9589, 9592-9594, 9597, 9599, 9602-9603, 9605, 9607, 9610, 9614, 9616-9626, 9628-9629, 9631, 9633-9637, 9639-9645,

9648-9651, 9654-9655, 9658-9661, 9663-9667, 9669-9673, 9675, 9677-9683, 9686-9691, 9694-9700, 9703-9705, 9708, 9710, 9712-9713, 9716-9718, 9720-9721, 9723-9724, 9726-9727, 9732, 9735, 9738-9742, 9744-9745, 9747-9748, 9750-9751, 9754-9759, 9761-9762, 9764-9765, 9767-9773, 9775-9776, 9778-9779, 9781-9786, 9788-9792, 9794-9796, 9798-9800, 9802, 9806, 9809, 9813-9815, 9819-9827, 9829, 9832-9833, 9836, 9839-9844, 9846, 9849-9853, 9855, 9858, 9860, 9862, 9864-9865, 9867-9878, 9880-9881, 9883, 9885, 9887-9901, 9903-9908, 9911-9912, 9915, 9917-9927, 9929-9931, 9933-9934, 9937, 9939, 9941-9942, 9944-9949, 9951-9953, 9956-9960, 9962-9965, 9967, 9969-9971, 9973-9976, 9978-9983, 9985-9988, 9990-9991, 9993, 9995-9996, 9998, 10000-10003, 10005-10006, 10008-10012, 10014, 10016-10017, 10020-10023, 10025-10027, 10029-10031, 10034-10035, 10037-10039, 10043, 10045-10046, 10048-10050, 10052-10055, 10059, 10061-10068, 10070, 10072-10074, 10076-10077, 10079, 10081, 10084, 10088-10091, 10094, 10097, 10100-10105, 10107, 10111-10112, 10114, 10116, 10118-10120, 10124, 10126, 10129-10131, 10134-10135, 10137-10138, 10140-10143, 10146, 10148-10154, 10156-10157, 10160-10162, 10164-10172, 10174, 10181, 10185-10188, 10191-10192, 10194, 10198, 10202, 10204-10208, 10210-10212, 10214-10215, 10218-10220, 10222-10224, 10226-10227, 10229-10230, 10232-10233, 10237-10239, 10242-10246, 10248-10249, 10251-10255, 10257-10259, 10262, 10265, 10268-10273, 10275-10276, 10279-10281, 10286-10288, 10290-10291, 10293, 10296-10297, 10300-10307, 10309, 10311, 10316, 10318-10319, 10321-10324, 10326-10331, 10335-10338, 10340-10343, 10345-10346, 10348, 10353-10355, 10357, 10359-10361, 10363, 10366-10368, 10371-10378, 10380-10381, 10384-10394, 10396-10397, 10401-10403, 10405-10406, 10409-10410, 10415-10418, 10420-10422, 10424-10427, 10430-10435, 10438-10439, 10441, 10443, 10445-10446, 10448-10449, 10452-10453, 10455, 10460, 10462, 10467, 10469, 10471-10474, 10476-10480, 10483-10485, 10487-10488, 10490-10500, 10502-10503, 10505-10510, 10512-10516, 10520-10524, 10526, 10529-10531, 10533, 10535-10536, 10539-10547, 10549-10551, 10553, 10555, 10557-10560, 10563-10565, 10568, 10571-10572, 10574-10575, 10577-10579, 10581-10588, 10590-10593, 10595, 10597-10604, 10606, 10608-10617, 10621-10622, 10624-10627, 10629-10631, 10633, 10635-10636, 10638-10640, 10642-10645, 10647-10649, 10653-10654, 10657-10660, 10662-10663, 10666, 10668-10670, 10672-10675, 10677-10680, 10684-10687, 10689-10690, 10692-10694, 10696-10697, 10699-10704, 10707-10718, 10720-10728, 10730-10731, 10734-10737, 10739-10740, 10742, 10744-10745, 10747, 10749-10752, 10754-10757, 10761-10763, 10765-10772, 10774-10777, 10779-10784, 10786-10793, 10798, 10801-10804, 10806, 10809, 10812-10813, 10815, 10818-10820, 10823-10824, 10826, 10829, 10831-10836, 10838, 10840-10843, 10845-10848, 10850, 10852-10854, 10856-10860, 10865-10868, 10872, 10874-10878, 10880-10881, 10883-10887, 10889, 10892, 10894, 10896-10897, 10900-10903, 10905-10906, 10908-10909, 10911-10913, 10915, 10921-10922, 10924, 10927-10930, 10932-10935, 10937-10938, 10941-10942, 10944, 10948-10951, 10953-10954, 10957, 10960, 10962, 10964-10965, 10969-10972, 10974, 10976-10979, 10982-10986, 10991-10992, 10994, 10996, 11000-11003, 11005-11007, 11009-11010, 11012-11013, 11015, 11018, 11020-11021, 11023-11028, 11030-11032, 11034-11035, 11037-11041, 11043-11047, 11049-11059, 11062-11076, 11080-11086, 11089-11093, 11095, 11100-11103, 11105-11108, 11113, 11116-11132, 11134-11141, 11143-11145, 11147-11151, 11153-11154, 11156-11157, 11159, 11161-11168, 11170, 11172-11173, 11175, 11177-11179, 11181, 11183, 11185, 11187-11188, 11190-11191, 11194-11200, 11204-11212, 11215-11220, 11223, 11225-11227, 11230-11235, 11239-11242, 11244-11248, 11252-11259, 11261, 11264-11265, 11267-11268, 11270-11273, 11278, 11280-11282, 11284-11288, 11291, 11293, 11295, 11298-11301, 11303, 11305, 11307, 11309-11314, 11317-11318, 11321, 11324-11325, 11327-11329, 11331-11332, 11334-11336, 11338-11340, 11342-11344, 11347-11356, 11358-11359, 11362-11364, 11366-11369, 11371-11376, 11379-11381, 11383-11385, 11387-11389, 11391, 11395-11399, 11401-11405, 11408-11412, 11417, 11419-11420, 11422-11425, 11428-11429, 11432-11433, 11435-11437, 11439, 11441, 11443-11453, 11456-11459, 11461, 11465-11468, 11470, 11474-11475, 11477-11483, 11485-11490, 11492-11493, 11495, 11497-11502, 11504-11507, 11509-11512, 11514-11518, 11520-11523, 11526-11534, 11536-11537, 11539-11543, 11546-11554, 11557-11558, 11560, 11562, 11564-11565, 11567, 11569-11571, 11573-11574, 11576-11577, 11580-11581, 11583, 11585, 11588-11589, 11592-11593, 11596-11597, 11599-11610, 11613, 11615-11616, 11618-11623, 11626-11632, 11635-11639, 11642-11643, 11646-11649, 11651-11653, 11655-11658, 11661-11662, 11664-11676, 11678-11683, 11686-11691, 11693-11695, 11697, 11699, 11701-11703, 11705, 11709, 11711-11712, 11716, 11718-11725, 11727-11728, 11732, 11734-11739, 11741-11748, 11751, 11753-11755, 11757-11758, 11760-11761, 11763-11768, 11770-11772, 11774, 11776-11777, 11779-11781, 11783-11784, 11786-11797, 11800-11804, 11808-11811, 11813-11818, 11820-11822, 11824, 11826-11828, 11830-11833, 11835, 11837-11839, 11841-11845, 11847-11854, 11856-11859, 11861-11863, 11867, 11869-11870, 11872-11874, 11878-11879, 11881-11882, 11884-11886, 11888-11898, 11900-11902, 11904, 11907, 11909-11911, 11913-11914, 11916-11920, 11922, 11924-11925, 11927, 11929-11933, 11935-11943, 11946, 11948, 11950-11960, 11962-11964, 11967-11970, 11972-11973, 11975-11977, 11979-11981, 11983, 11985, 11988-11993, 11995, 11999-12000, 12004-12005, 12007-12008, 12012, 12016-12017, 12019-12027, 12031, 12034-12035, 12039-12041, 12044-12045, 12047-12048, 12050, 12052-12053, 12056-12061, 12064, 12066-12075, 12078-12083, 12085-12087, 12089-12090, 12092, 12094-12096, 12098-12099, 12101-12103, 12105-12110, 12113-12114, 12116-12118, 12121-12128, 12130-12132, 12135-12137, 12139-12141, 12143-12148, 12150-12151, 12153-12156, 12158, 12160-12162, 12165-12170, 12172-12173, 12176, 12178-12179, 12181-12184, 12186-12201, 12203-12205, 12209-12210, 12212-12215, 12217-12219, 12221-12222, 12224, 12226-12228, 12231-12240, 12242-12243, 12245, 12247-12248, 12252, 12254-12258, 12261, 12263, 12265-12268, 12270-12280, 12283-12287, 12290-12295, 12297-12298, 12300-12302, 12304-12308, 12311, 12314, 12320-12324, 12326-12329, 12331-12335, 12337, 12341, 12343-12345, 12347, 12349-12351, 12353, 12355-12365, 12368, 12370-12371, 12373-12375, 12377-12381, 12384-12389, 12393, 12396-12397, 12399-12401, 12403, 12405-12407, 12409-12410, 12412-12419, 12421-12422, 12424-12426, 12428-12430, 12435-12438, 12440-12443, 12445-12451, 12453, 12455-12460, 12462-12469, 12472, 12475-12477, 12479-12485, 12487-12504, 12507-12514, 12518-12520, 12522-12523, 12525, 12528-12529, 12532, 12534-12536, 12538, 12540-12545, 12547-12550, 12553-12557, 12559, 12561-12564, 12566-12567, 12569-12573, 12575-12576, 12578-12583, 12586-12587, 12589-12591, 12594-12597, 12599-12601, 12603-12605, 12607-12608, 12610-12611, 12613-12615, 12620-12622, 12624-12627, 12629-12630, 12632-12633, 12635-12640, 12643-12647, 12650, 12652-12654, 12656, 12658-12660, 12662-12664, 12667-12671, 12673, 12675-12678, 12680, 12682-12684, 12686-12697, 12699-12702, 12705, 12708-12709, 12711-12713, 12715-12726, 12728-12730, 12732-12735, 12741-12755, 12757-12758, 12761, 12763-12764, 12766-12767, 12771-12772, 12775, 12777-12780, 12782-12784, 12786-12788, 12790-12793, 12795-12799, 12801-12805, 12807-12808, 12810, 12812, 12814, 12820-12821, 12823-12825, 12827-12829, 12832-12833, 12835-12837, 12839-12840, 12842-12848, 12851, 12853, 12855-12857, 12859, 12862-12865, 12868-12874, 12877, 12879-12884, 12886, 12888-12890, 12895-12906, 12908-12910, 12913-12918, 12920-12926, 12929-12940, 12942-12947, 12951-12954, 12956-12960, 12964-12967, 12970-12976, 12978-12981, 12983, 12985-12987, 12990, 12992-12994, 12996-12997, 12999, 13003, 13007-13012, 13015, 13017, 13019-13020, 13022, 13026-13029, 13031-13035, 13037-13039, 13041-13049, 13051-13053, 13055-13060, 13062, 13064, 13066, 13069, 13071-13078, 13080, 13082-13083, 13085-13086, 13089-13098, 13101-13107, 13110-13113, 13115-13118, 13120-13122, 13124-13128, 13130, 13133, 13135, 13137, 13139-13142, 13144-13152, 13156-13160, 13162, 13164, 13167-13169, 13171-13176, 13178, 13181-13183, 13185-13187, 13190-13191, 13193, 13195-13198, 13200-13204, 13206-13213, 13215-13219, 13221-13227, 13229-13236, 13239-13240, 13242-13245, 13248-13251, 13253-13254, 13256-13259, 13264-13265, 13267-13273, 13275, 13277, 13279-13281, 13283, 13286, 13290-13297, 13299-13305, 13307, 13309, 13311-13313, 13315, 13318-13321, 13324-13325, 13327-13330, 13332, 13335-13340, 13342, 13344, 13348, 13351-13356, 13358-13359, 13361-13364, 13366, 13369, 13372, 13374-13375, 13377-13378, 13382-13383, 13386-13391, 13393, 13395-13402, 13407, 13409-13412, 13414-13415, 13419-13421, 13423-13425, 13427-13430, 13432-13435, 13437, 13439-13443, 13445-13450, 13453-13456, 13458, 13460-13461, 13463, 13466, 13468-13470, 13472-13477, 13481-13489, 13491, 13493-13494, 13496-13499, 13501-13502, 13505-13507, 13509, 13512-13518, 13521-13529, 13531-13532, 13534, 13538-13539, 13541-13544, 13546-13547, 13549-13550, 13553, 13556-13557, 13559-13561, 13563-13564, 13566-13567, 13569-13574, 13577-13581, 13583-13587, 13589-13592, 13594-13595, 13598-13600, 13605, 13607, 13609-13612, 13614-13617, 13619-13622, 13624-13625, 13627-13629, 13631-13636, 13638-13644, 13646, 13648-13652, 13654-13656, 13660-13662, 13664-13665, 13667-13677, 13680, 13682, 13684-13686, 13688-13689, 13691, 13693-13703, 13708-13709, 13712-13715, 13717-13722, 13724-13725, 13727-13729, 13731-13733, 13736, 13738-13739, 13741, 13744-13748, 13750, 13752, 13754-13756, 13758-13762, 13767-13770, 13772, 13774, 13776-13778, 13780, 13782, 13784-13785, 13788, 13790-13794, 13796, 13799, 13801, 13803-13806, 13809-13810, 13812-13825, 13828, 13830, 13832-13837, 13840-13841, 13843-13844, 13846-13857, 13859-13860, 13864-13869, 13872-13875, 13877-13880, 13882-13888, 13890-13892, 13895-13896, 13901-13902, 13905-13909, 13911, 13913-13914, 13917, 13919, 13921-13923, 13926-13927, 13930-13931, 13933-13934, 13936-13937, 13939, 13941-13945, 13948, 13952, 13954-13957, 13960-13962, 13964-13969, 13972, 13974, 13976-13978, 13980-13981, 13983-13988, 13990-13992, 13994-

13996, 13999-14003, 14005, 14007, 14009, 14011-14012, 14014-14015, 14017, 14020-14031, 14034-14037, 14039, 14042-14046, 14048-14051, 14054, 14056, 14058, 14063-14065, 14067-14069, 14071-14075, 14078-14086, 14088-14089, 14091-14095, 14097-14100, 14102-14107, 14109, 14111, 14113-14117, 14120, 14122, 14126-14136, 14138-14143, 14145-14146, 14148, 14150-14152, 14154-14156, 14158-14160, 14162-14178, 14180-14189, 14193-14198, 14200, 14202-14203, 14206-14216, 14218-14222, 14224-14227, 14230-14234, 14236-14243, 14246-14247, 14250, 14254-14255, 14259, 14261-14263, 14265-14269, 14273, 14275-14277, 14279-14280, 14283-14287, 14289-14296, 14298-14300, 14305-14307, 14309-14316, 14318-14319, 14321, 14323-14327, 14329-14331, 14334, 14336-14338, 14340, 14344, 14346-14362, 14364-14369, 14371, 14373-14374, 14376, 14378-14380, 14382, 14385, 14387, 14389, 14391, 14393, 14395-14399, 14401-14406, 14408-14409, 14411-14415, 14418, 14420-14424, 14426-14432, 14434, 14437-14438, 14440, 14442-14444, 14446-14447, 14449-14456, 14458, 14460, 14462-14464, 14466-14468, 14470-14492, 14494-14495, 14497, 14499-14500, 14505, 14509-14510, 14513-14514, 14516-14518, 14521, 14524-14529, 14532-14535, 14537, 14539-14541, 14543-14544, 14548, 14551-14556, 14558-14560, 14562-14564, 14568, 14570-14572, 14574, 14579-14580, 14582-14583, 14585, 14587, 14591-14592, 14596, 14598-14601, 14603-14604, 14606-14611, 14613-14614, 14616-14617, 14619-14623, 14626-14632, 14634-14637, 14639-14649, 14651-14652, 14656-14658, 14660, 14662-14663, 14666, 14670-14671, 14676-14679, 14681, 14683-14687, 14690, 14693-14696, 14698, 14700-14701, 14703-14704, 14707-14709, 14711, 14713, 14715-14716, 14720-14723, 14726, 14728, 14730-14732, 14734, 14736-14738, 14740-14743, 14745-14747, 14749, 14751-14753, 14755-14765, 14767-14768, 14770-14776, 14778, 14781-14786, 14789, 14791, 14794-14796, 14798, 14800, 14802-14810, 14812-14816, 14819, 14821-14825, 14827-14828, 14831, 14833, 14838, 14841, 14845-14846, 14848-14850, 14852, 14855-14857, 14861-14865, 14867-14868, 14871-14872, 14876-14880, 14882-14883, 14885-14891, 14894-14899, 14902-14903, 14905-14907, 14912-14921, 14923-14927, 14931, 14933-14934, 14937-14943, 14945, 14947, 14949-14950, 14952-14953, 14955-14957, 14959, 14961-14970, 14972-14978, 14980-14983, 14985-14988, 14990-14991, 14996-14999]

C.2 Testing Set

Corresponding datasets: Mulliken_Test, Hirshfeld_Test, CM5_Test, DDEC6_Test and Struct_Test.

cno8_[15000, 15002-15003, 15006, 15008-15010, 15012-15013, 15015-15016, 15018-15019, 15022-15024, 15027-15029, 15032-15037, 15039-15042, 15045, 15047-15050, 15052-15056, 15058-15065, 15068, 15070-15072, 15075, 15077-15079, 15081-15084, 15086-15089, 15091, 15093-15094, 15096-15113, 15115-15118, 15120-15123, 15125-15126, 15128-15130, 15132-15133, 15135, 15137-15139, 15141-15142, 15144, 15146, 15148-15151, 15153-15155, 15157-15159, 15161-15163, 15165-15169, 15171-15174, 15176-15177, 15179, 15182-15183, 15185-15187, 15189-15190, 15193-15194, 15196-15197, 15199, 15201-15203, 15205-15210, 15213, 15215-15216, 15219-15225, 15227-15230, 15232-15239, 15241, 15243, 15246-15247, 15249-15259, 15261, 15263-15271, 15273, 15275-15280, 15282-15287, 15289, 15291-15292, 15295-15298, 15301-15306, 15310-15312, 15314, 15316, 15318-15319, 15322-15329, 15331-15334, 15336, 15339-15341, 15346, 15348-15352, 15354-15355, 15357-15358, 15360-15362, 15364, 15367-15368, 15371, 15373, 15375-15380, 15382, 15386-15392, 15394-15395, 15397, 15399-15402, 15404-15405, 15409-15410, 15414-15419, 15424, 15427-15428, 15433-15439, 15441, 15444-15445, 15447, 15449, 15451-15453, 15455-15456, 15459-15461, 15463-15467, 15469-15470, 15472-15474, 15476, 15478-15480, 15482-15485, 15487, 15490-15491, 15493, 15495-15499, 15501, 15503-15505, 15507, 15510-15513, 15515-15521, 15523-15528, 15531-15533, 15535, 15537, 15539-15547, 15552-15555, 15557, 15560-15564, 15567-15568, 15570, 15573-15576, 15580, 15582-15585, 15587-15588, 15590-15591, 15595-15598, 15600-15601, 15604-15606, 15608, 15611, 15613-15626, 15629, 15631-15633, 15635-15636, 15638, 15640-15641, 15644-15647, 15650-15655, 15657-15658, 15660, 15663-15666, 15670-15671, 15674, 15676, 15678-15680, 15682-15688, 15690, 15692-15695, 15697-15698, 15701-15704, 15706-15710, 15712, 15714, 15716, 15719-15723, 15725, 15727-15728, 15730, 15732, 15734-15738, 15740-15743, 15745-15748, 15750-15751, 15753-15758, 15761, 15763-15776, 15778, 15780-15782, 15784-15785, 15787-15788, 15791-15796, 15801-15804, 15808-15812, 15814-15823, 15825-15828, 15831-15832, 15834, 15836-15839, 15841, 15843-15848, 15851-15853, 15857-15858, 15860-15861, 15863-15868, 15870-15878, 15880-15882, 15884-15889, 15892-15893, 15896, 15899, 15902-15908, 15911-15913, 15915, 15917, 15920-15922, 15925, 15927-15931, 15933-15935, 15937-15938, 15940-15942, 15944-15946, 15948-15949, 15951-15952, 15954-15958, 15960-15966, 15968-15970, 15972-15976, 15978-15979, 15981-15983, 15985-15989, 15991-15994, 15997-15999, 16001-16007, 16009-16010, 16012-16013, 16015-16020, 16022-16023, 16025-16026, 16028-16030, 16033-16035, 16038, 16040-16041, 16043-16045, 16048-16052, 16054-16056, 16058, 16060-16061, 16065-16066, 16068-16069, 16072-16077, 16079-16083, 16085-16087, 16089-16091, 16095-16100, 16102-16107, 16109-16110, 16112-16114, 16116-16122, 16124, 16127-16132, 16134-16135, 16137, 16139, 16143-16147, 16149-16150, 16155-16157, 16160-16162, 16164, 16167-16168, 16170-16171, 16173-16177, 16180-16183, 16186-16191, 16193-16194, 16197, 16199-16203, 16206-16209, 16211, 16214-16217, 16220-16221, 16223-16228, 16230-16232, 16234-16235, 16237-16242, 16244-16245, 16247, 16249-16251, 16255-16256, 16260-16265, 16268, 16270-16273, 16275-16276, 16278-16279, 16282, 16284-16286, 16288-16290, 16293, 16295-16297, 16300-16305, 16309-16311, 16313-16320, 16322, 16324-16325, 16327-16328, 16331, 16333-16337, 16340-16351, 16353, 16355-16360, 16362-16363, 16365-16366, 16369, 16372, 16378-16379, 16381-16383, 16385-16386, 16388-16389, 16391-16393, 16396, 16398-16399, 16401-16402, 16405-16406, 16408, 16411-16415, 16417-16423, 16425, 16427-16429, 16431-16432, 16436-16439, 16441-16442, 16445, 16449-16451, 16453-16459, 16463-16464, 16466-16469, 16472-16484, 16488-16490, 16492, 16494-16496, 16498, 16500-16511, 16513-16514, 16516, 16520-16523, 16525-16528, 16531, 16534, 16537-16538, 16540, 16542-16543, 16545, 16547-16548, 16550, 16552-16554, 16557-16559, 16561-16563, 16565-16567, 16573-16580, 16583-16584, 16587-16588, 16591-16600, 16602-16606, 16608-16609, 16613, 16615-16619, 16621-16623, 16625-16627, 16629, 16631, 16633, 16635, 16638-16639, 16643-16644, 16647-16648, 16651, 16653-16655, 16658-16660, 16662, 16668-16673, 16675-16677, 16679-16680, 16682-16686, 16691-16700, 16702-16704, 16706-16708, 16710-16711, 16719-16720, 16723-16725, 16728-16729, 16731-16737, 16739-16741, 16743, 16745-16746, 16750-16760, 16762-16768, 16770, 16772-16775, 16777-16785, 16787-16789, 16791-16798, 16800-16802, 16807, 16809-16810, 16813-16817, 16819, 16821-16822, 16825-16830, 16832, 16834-16837, 16841-16843, 16846-16852, 16854, 16856-16857, 16860, 16863, 16865-16867, 16869-16872, 16874-16878, 16880, 16882-16886, 16888-16895, 16897, 16899-16900, 16902-16906, 16908, 16910-16912, 16915-16920, 16925, 16928, 16930-16931, 16933-16934, 16936, 16939-16943, 16945-16947, 16949-16956, 16958-16959, 16961-16962, 16964, 16966, 16969, 16971, 16973-16977, 16979-16980, 16988-16991, 16993, 16995-16998, 17000, 17002-17004, 17006, 17008-17010, 17012-17015, 17018-17020, 17022, 17024-17026, 17028-17029, 17031-17032]

Appendix D

Selected Molecules in ChEMBL-35-based Datasets

The instruction of obtaining original ChEMBL-35 database is given in Chapter 7. Here, all used molecules in ChEMBL-35 database are listed in SMILES.

D.1 Testing Set

Corresponding Datasets: LM_Mulliken_Test, LM_Hirshfeld_Test, LM_CM5_Test and LM_Struct_Test.

ChEMBL_3: CCCCC[C@H]1CCCCCCCCCC(=O)O[C@@H]2[C@@H](O[C@@H]3O[C@H](C)[C@@H](OC(=O)C(C)CC)[C@H](O)[C@H]3O)[C@H](C)O[C@@H](O[C@H]3[C@H](O[C@H]4[C@H](O1)O[C@H](C)[C@@H](O)[C@@H]4O)O[C@H](CO)[C@@H](O)[C@@H]3O)[C@@H]2OC(=O)C(C)CC

ChEMBL_166: NC[C@@H]1O[C@H](O[C@H]2[C@@H](O)[C@H](O[C@@H]3[C@@H](O)[C@H](N)C[C@H](N)[C@H]3O[C@H]3O[C@H](CN)[C@@H](O)[C@H](O)[C@H]3N)O[C@@H]2Cn2cc(C3CCCCN3)nn2)[C@H](N)[C@@H](O)[C@@H]1O

ChEMBL_59: CNC(=O)NC(=N)NCCC[C@@H]1NC(=O)[C@@H](CC(C)C)NC(=O)C[C@@H](C(=O)O)NC(=O)C[C@@H](C(=O)O)NC(=O)[C@H](Cc2ccccc2)N(C)C1=O

ChEMBL_71: CC(C)[C@H](NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCCCN)NC(=O)[C@H](CCCCN)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](C)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCC(=O)O)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H]1CCCN1C(=O)[C@@H](N)[C@@H](C)O)C(=O)O

ChEMBL_64: CO[C@@H]1[C@@H](O)[C@H](O[C@@H]2CO[C@@H](O[C@@H]3CO[C@@H](OCC[C@@H](/C=C/[C@@H](C)[C@H]4[C@@H](O)[C@@H](O)[C@]5(C)[C@]4(C)CC[C@@H]4[C@@]6(C)CC[C@H](O)[C@H](O)[C@@H]6[C@@H](O)C[C@]45O)C(C)C)[C@H](O)[C@H]3O)[C@H](O)[C@H]2OC)OC[C@H]1O

ChEMBL_58: CN(C)CCCNC(=O)CCNC(=O)c1cc(NC(=O)c2cc(NC(=O)c3cc(NC(=O)[C@H](N)CCNC(=O)c4cc(NC(=O)c5nc(NC(=O)c6nccn6C)cn5C)cn4C)cn3C)cn2C)cn1C

ChEMBL_167: C=CC(C)(CC/C=C(\CO)C(=O)O[C@H]1C[C@]2(C(=O)O)[C@H](O)C[C@]3(C)C(=CC[C@@H]4[C@@]5(C)CC[C@H](O[C@@H]6O[C@H](CO[C@@H]7O[C@H](C)[C@H](O)[C@H](O)[C@H]7O[C@@H]7OC[C@@H](O)[C@H](O)[C@H]7O)[C@@H](O)[C@H](O)[C@H]6O[C@@H]6O[C@H](CO)[C@@H](O)[C@H](O)[C@H]6O)C(C)(C)[C@@H]5CC[C@]43C)[C@@H]2CC1(C)C)O[C@@H]1O[C@@H](C)[C@H](O)[C@@H](O)[C@@H]1O

ChEMBL_159: COc1ccc(-c2coc3cc(O[C@@H]4O[C@H](CO[C@@H]5O[C@@H](C)[C@H](O)[C@@H](O)[C@H]5O)[C@@H](O)[C@H](O[C@@H]5O[C@H](CO[C@@H]6O[C@@H](C)[C@H](O)[C@@H](O)[C@H]6O)[C@@H](O)[C@H]6O)[C@@H](O)[C@H](O)[C@H]5O)[C@H]4O)c(OC)c(OC)c3c2=O)cc1

ChEMBL_171: COc1cc(O)c2c(O[C@@H]3O[C@H](CO)[C@@H](O)[C@H](O)[C@H]3O)c3c(c4c2c1[C@@]1(C4)C(C)=CCCC1(C)C)[C@H](O)[C@@]1(O)CC(O)=C(C(N)=O)C(=O)[C@@]1(O)C3=O

ChEMBL_165: O=C(O[C@H]1C(O)O[C@@H]2COC(=O)c3cc(O)c(O)c(O)c3-c3c(O)c(O)c4oc(=O)c5c(c(O)c(O)c6oc(=O)c3c4c65)-c3c(cc(O)c(O)c3O)C(=O)O[C@H]2[C@@H]1O)c1cc(O)c(O)c(O)c1

ChEMBL_99: CCCN(NC(=O)[C@@H]1[C@@H]2[C@H](CN1C(=O)[C@@H](NC(=O)N[C@H](CN1C(=O)C3CCC(C3)C1=O)C(C)(C)C)C(C)(C)C)C2(C)C)C(=O)c1ccccc1

ChEMBL_72: CC[C@H](C)[C@H]1NC(=O)[C@@H](Cc2ccoc2)N(C)C(=O)[C@H](C)N(C)C(=O)[C@H](CC(C)C)NC(=O)[C@H](Cc2ccoc2)N(C)C(=O)[C@H](C(C)C)NC(=O)[C@@H](C(C)C)NC1=O

ChEMBL_73: NCCCC[C@H](NC(=O)[C@@H]1C[C@@H](OCC2CCCCC2)CN1C(=O)[C@@H](CCc1ccccc1)NC(=O)OCc1ccccc1)C(=O)c1nc2ccccc2o1

ChEMBL_98: CC(C)[C@H](NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCCCN)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCC(=O)O)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H]1CCCN1C(=O)[C@@H](N)[C@@H](C)O)C(=O)O

ChEMBL_164: COc1ccc(OC)c([C@@H]2O[C@H](CNC(=O)c3ccccc3C(=O)O)[C@@H](OC(=O)c3ccccc3)[C@H](OC(=O)c3ccccc3)[C@H]2OC(=O)c2ccccc2)c1

ChEMBL_158: COc1cc2c(O[C@@H]3OC[C@]([O])(CO[C@@H]4OC[C@@H](O)[C@H](O)[C@H]4O)[C@H]3O[C@H]3OC[C@@H](O)[C@@H](O)[C@@H]3O)c3c(c(-c4ccc5c(c4)OCO5)c2cc1OC)C(=O)OC3

ChEMBL_1: CO[C@H]1C[C@H](COC[C@H]2[C@@H](OC)C[C@H](O[C@H]3CC[C@@]4(C)C(=CC[C@]5(O)[C@@H]4C[C@@H](OC(=O)/C=C/c4ccccc4)[C@@]4(C)[C@]5(O)CC[C@@]4(O)C(C)=O)C3)O[C@@H]2C)O[C@@H](C)[C@H]1COC[C@H]1C[C@H](OC)[C@H](O[C@H]2C[C@@H](OC)[C@H](O[C@H]3O[C@@H](CO)[C@H](O)[C@@H](O)[C@@H]3O)[C@H](C)O2)[C@@H](C)O1

ChEMBL_5: COc1cc(C(O)C(COC(=O)/C=C/c2ccc(O)cc2)Oc2c(OC)cc(/C=C/COC(=O)/C=C/c3ccc(O)cc3)cc2OC)ccc1O

ChEMBL_174: CC(C)[C@H](NC(=O)[C@@H](Cc1c[nH]c2ccccc12)NC(=O)[C@H](Cc1ccc(O)cc1)NC(=O)[C@@H](N)CC(=O)O)C(=O)N[C@H](Cc1c[nH]c2ccccc12)C(=O)N[C@H](Cc1c[nH]c2ccccc12)C(=O)N[C@@H](CCCCN)C(N)=O

ChEMBL_160: C=C(C)[C@@H]1CC[C@]2(C(=O)O[C@@H]3O[C@H](CO)[C@@H](O)[C@H](O)[C@H]3O)CC[C@]3(C)[C@H](CC[C@@H]4[C@@]5(C)CC[C@H](O[C@@H]6OC[C@H](O)[C@H](O)[C@H]6O)C(C)(C)[C@@H]5CC[C@]43C)[C@@H]12

ChEMBL_148: COc1cc(O)c2c(=O)c(O[C@@H]3O[C@H](CO[C@@H]4O[C@@H](C)[C@H](O)[C@@H](O[C@@H]5O[C@@H](C)[C@H](O)[C@@H](O)[C@H]5O)[C@H]4O)[C@@H](O)[C@H](O)[C@H]3O)c(-c3ccc(O)c(O)c3)oc2c1

ChEMBL_88: CO[C@H]1[C@H](O[C@H]2[C@H](OCC[C@@H](/C=C/[C@@H](C)[C@H]3C[C@@H](O)[C@H]4[C@@H]5C[C@H](O)[C@H]6C[C@@H](O)CC[C@]6(C)[C@H]5CC[C@@]43C)C(C)C)O[C@@H](CO)[C@@H]2O)OC[C@@H](OC)[C@@H]1O
ChEMBL_77: CC(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCCN)C(=O)N1CCC[C@H]1C(=O)N[C@@H](Cc1c[nH]c2ccccc12)C(=O)N[C@H](C(=O)N[C@@H](CC(C)C)C(=O)O)C(C)(C)C
ChEMBL_63: C[C@@H]1O[C@@H](O[C@H]2[C@H](O)[C@@H](O)[C@H](OC[C@H]3O[C@@H](OC(=O)[C@]45CCC(C)(C)C[C@H]4C4=CC[C@@H]6[C@@]7(C)CC[C@H](O[C@@H]8O[C@H](C(=O)O)[C@@H](O)[C@H](O)[C@H]8O[C@@H]8O[C@H](CO)[C@@H](O)[C@H](O)[C@H]8O)C(C)(C)[C@@H]7CC[C@@]6(C)[C@]4(C)CC5)[C@H](O)[C@@H](O)[C@@H]3O)O[C@@H]2CO)[C@H](O)[C@H](O)[C@H]1O
ChEMBL_76: COc1cc(O)c2c(c1C(=O)/C=C/c1ccc(O)cc1)O[C@H](c1ccc(O)cc1)[C@@H]1[C@@H](c3ccc(O)cc3)O[C@@H](CCc3ccc(O)cc3)C[C@@H]21
ChEMBL_89: COc1cc(/C=C/C(=O)OC[C@H]2O[C@@](COC(=O)/C=C/c3ccc(O)cc3)(O[C@H]3O[C@H](COC(C)=O)[C@@H](O)[C@H](O)[C@H]3O)[C@@H](OC(=O)/C=C/c3ccc(O)c(OC)c3)[C@@H]2O)ccc1O
ChEMBL_149: CC[C@H]1OC(=O)[C@H](C)[C@@H](O[C@H]2C[C@@](C)(OC)[C@@H](O)[C@H](C)O2)[C@H](C)[C@@H](O[C@@H]2O[C@H](C)C[C@H](N(C)C)[C@H]2OC(=O)CCNC(=O)OC(C)(C)C)[C@@](C)(O)C[C@@H](C)C(=O)[C@H](C)[C@@H](OC(=O)CCNC(=O)OC(C)(C)C)[C@@]1(C)O
ChEMBL_161: CCCCCCCCCC[C@H](CCCCCCCC(C)=O)O[C@@H]1O[C@H](COC(C)=O)[C@@H](OC(C)=O)[C@H](OC(C)=O)[C@H]1O[C@@H]1O[C@H](CO[C@@H]2O[C@H](C)[C@H](OC(C)=O)[C@H](OC(C)=O)[C@@H]2OC(C)=O)[C@@H](OC(C)=O)[C@H](OC(=O)CCC)[C@H]1O[C@@H]1O[C@@H](C)[C@H](OC(C)=O)[C@@H](OC(C)=O)[C@@H]1OC(C)=O
ChEMBL_175: CC(=O)OC[C@]12Oc3ccc(-c4ccc5c(c4O)C(=O)C4=C(O)C[C@@H](C)[C@@H](OC(C)=O)[C@@]4(COC(C)=O)O5)c(O)c3C(=O)C1=C(O)C[C@@H](C)[C@H]2OC(C)=O
ChEMBL_4: COC1=C2C[C@@H](O)CC[C@]2(C)[C@H]2CC[C@@]3(C)[C@@H](CC(=O)[C@]3(O)[C@H](C)[C@H](CCC(C)C)O[C@@H]3OC[C@H](O)[C@H](O[C@@H]4OC[C@@H](O)[C@H](O)[C@H]4OC(=O)c4ccc(OC)cc4)[C@H]3OC(C)=O)[C@@H]2C1
ChEMBL_188: C=CCNC(=O)C(=O)C(CCCC)NC(=O)[C@@H]1[C@@H]2[C@H](CN1C(=O)[C@@H](NC(=O)N[C@@H](CN1C(=O)CCN(C)C1=O)C(C)(C)C)C(C)(C)C)C2(C)C
ChEMBL_177: CCN(CCCCCCn1c(=O)c2cc3c(=O)n(CCCCCCN(CC)Cc4ccccc4OC)c(=O)c3cc2c1=O)Cc1ccccc1OC
ChEMBL_60: CCCC(=O)O[C@@H]1[C@H](C(C)C)[C@@H]2[C@H]3C=C(C)[C@H](O)[C@H](OC(=O)CC)[C@@H](OC(=O)CCC)[C@]3(C)C[C@H](OC(=O)CCC)[C@]2(C)[C@H]1OC(=O)CCC
ChEMBL_74: CC(=O)OC[C@]12[C@H](OC(C)=O)[C@H](OC(=O)c3cccnc3)[C@@H]3[C@@H](OC(C)=O)[C@@]14O[C@@]3(C)COC(=O)c1cccnc1C(C)C(C)C(=O)O[C@@H]([C@H](OC(C)=O)[C@@H]2OC(C)=O)[C@]4(C)O
ChEMBL_49: CC(=O)O[C@@]12CO[C@@H]1C[C@H](O)[C@@]1(C)C(=O)[C@H](OC(=O)CCc3ccccc3)C3=C(C)[C@@H](OC(=O)[C@H](O)[C@H](C=C(C)C)NC(=O)OC(C)(C)C)C[C@@](O)([C@@H](OC(=O)c4ccccc4)[C@H]21)C3(C)C
ChEMBL_200: COc1cccc(C(=O)O[C@H]2[C@@H]3[C@]4(OC(C)=O)CO[C@@H]4C[C@H](OC(=O)[C@H](C)NC(=O)c4ccc(-c5c6ccc(=O)cc-6oc6cc(O)ccc56)c(C(=O)O)c4)[C@@]3(C)C(=O)[C@H](OC(C)=O)C3=C(C)[C@@H](OC(=O)[C@H](O)[C@@H](NC(=O)c4ccccc4)c4ccccc4)C[C@]2(O)C3(C)C)c1
ChEMBL_61: CC1=C(CC[C@@H](C)CO[C@@H]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O)O[C@H]2C[C@H]3[C@@H]4CC=C5C[C@@H](O[C@@H]6O[C@H](CO)[C@H](O[C@@H]7O[C@H](CO)[C@@H](O)[C@H](O[C@@H]8OC[C@@H](O)[C@H](O)[C@H]8O)[C@H]7O[C@@H]7O[C@H](CO)[C@@H](O)[C@H](O[C@@H]8O[C@H](CO)[C@@H](O)[C@H](O)[C@H]8O)[C@H]7O)[C@H](O)[C@H](O)[C@H]6O)[C@H](O)C[C@]5(C)[C@H]4CC[C@]3(C)[C@@H]12
ChEMBL_176: Oc1ccc(-c2c3nc(c(-c4ccc(O)cc4)c4ccc([nH]4)c(-c4ccc(O)cc4)c4nc(c(-c5ccc(O)cc5)c5ccc2[nH]5)C=C4)C=C3)cc1
ChEMBL_189: CC1(C)CC[C@]2(C(=O)O[C@@H]3OC[C@H](O)[C@H](O)[C@H]3O)CC[C@]3(C)C(=CC[C@@H]4[C@@]5(C)CC(=O)[C@H](O[C@@H]6O[C@H](CO)[C@H](O)[C@H]6O)[C@@](C)(CO)[C@@H]5[C@H](O)C[C@]43C)[C@@H]2C1
ChEMBL_7: C[C@@H]1O[C@@H](O[C@H]2[C@H](O[C@@H]3C[C@H](O)CC4=CC[C@H]5[C@@H]6C[C@@H]7O[C@]8(C[C@H](O)[C@@H](C)CO8)[C@@H](C)[C@@H]7[C@@]6(C)CC[C@@H]5[C@]43C)OC[C@H](O)[C@@H]2O[C@@H]2OC[C@@H](O)[C@H](O)[C@H]2O)[C@H](O)[C@H](O)[C@@H]1O
ChEMBL_105: COC(=O)N[C@H](C(=O)N[C@@H](Cc1ccc(-c2ccccn2)cc1)[C@@H](O)C[C@H](Cc1ccccc1)NC(=O)[C@@H](N1CCN(Cc2cccc(C(C)(C)O)n2)C1=O)C(C)(C)C)C(C)(C)C
ChEMBL_138: C=C(C)[C@@H]1CC[C@]2(C(=O)O[C@@H]3O[C@H](CO)[C@@H](O)[C@H](O)[C@H]3O)CC[C@]3(C)[C@H](CC[C@@H]4[C@@]5(C)CC[C@H](O[C@@H]6OC[C@H](O)[C@H](O)[C@H]6O[C@@H]6O[C@@H](C)[C@@H](O)[C@H](O)[C@H]6O)C(C)(C)[C@@H]5CC[C@]43C)[C@@H]12
ChEMBL_112: O=C1O[C@H]2[C@@H]3OC(=O)c4cc(O)c(O)c(O)c4-c4c(cc(O)c(O)c4O)C(=O)OC[C@H]2O[C@@H](OC(=O)c2cc(O)c(O)c(O)c2)[C@@H]3OC(=O)c2cc(O)c(O)c3c2[C@@H]2C1=CC(=O)[C@]([O-])(O3)C2(O)O
ChEMBL_106: CC(=O)O[C@H]1[C@@H](OC(C)=O)[C@@H](Cn2cc(COC(=O)c3cccc(C(=O)OCc4cn(C[C@H]5O[C@@H](C6=CC(=O)c7ccccc7C6=O)[C@H](OC(C)=O)[C@@H](OC(C)=O)[C@H]5OC(C)=O)nn4)c3)nn2)O[C@@H](C2=CC(=O)c3ccccc3C2=O)[C@@H]1OC(C)=O
ChEMBL_39: CCC(=O)OC[C@@]1(O)[C@H](OC(C)=O)[C@]2(CC)C=CCN3CC[C@@]4(c5cc([C@@]6(C(=O)OC)C[C@@H]7C=C(CC)CN(CCc8c6[nH]c6ccccc86)C7)c(OC)cc5N(C)[C@@H]14)[C@@H]32
ChEMBL_10: CN1C(=O)[C@H](NC(=O)Nc2cccc(OCCNC(=O)COCC(=O)NNC(=O)COCC(=O)N[C@H]3CC[C@@]4(O)[C@H]5Cc6cc(O)cc7c6[C@@]4(CCN5C)[C@H]3O7)c2)N=C(c2ccccc2)c2ccccc21
ChEMBL_38: CC(C)(C)c1cc2c(OCCCCNC(=N)N)c(c1)Cc1cc(C(C)(C)C)cc(c1OCCCCNC(=N)N)Cc1cc(C(C)(C)C)cc(c1OCCCCNC(=N)N)Cc1cc(C(C)(C)C)cc(c1OCCCCNC(=N)N)C2
ChEMBL_107: C=CCNC(=O)C(=O)[C@@H]1CCCCCCCCC[C@H](NC(=O)N[C@H](CN2C(=O)c3ccccc3C2=O)C(C)(C)C)C(=O)N2C[C@@H]3[C@H]([C@H]2C(=O)N1)C3(C)C
ChEMBL_113: C[C@H]1O[C@@H](O[C@H]2[C@H](O[C@H]3CO[C@@H](O[C@H]4CC[C@]5(C)[C@H]6CC[C@]78OC[C@@]9(CCC(C)(C)C[C@H]97)[C@H](O)C[C@@]8(C)[C@]6(C)CC[C@H]5C4(C)C)[C@H](O[C@@H]4O[C@H](CO)[C@@H](O)[C@H](O)[C@H]4O)[C@H]3O)O[C@H](CO)[C@@H](O)[C@@H]2O)[C@@H](O)[C@@H](O)[C@@H]1O
ChEMBL_14: CO[C@]1(CC[C@H](C)CO[C@@H]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O)O[C@H]2C[C@H]3[C@@H]4CC=C5C[C@@H](O[C@@H]6O[C@H](CO)[C@H](O)[C@H]6O[C@H]6O[C@@H](O)[C@H](O[C@@H]7O[C@@H](C)[C@H](O[C@@H]8O[C@H](CO)[C@@H](O)[C@H](O)[C@H]8O)[C@@H](O)[C@H]7O)[C@H]6O[C@@H]6O[C@@H](C)[C@H](O)[C@@H](O)[C@]6(C)O)CC[C@]5(C)[C@H]4CC[C@]3(C)[C@H]2[C@@H]1C
ChEMBL_28: CCCCCCCCCCCC(=O)O[C@H]1[C@H](O[C@@H]2[C@@H](O)[C@H]3OC(=O)CCCCCCCCCC(CCCCC)O[C@@H]4O[C@H](CO)[C@@H](O)[C@H](O)[C@H]4O[C@H]3O[C@H]2C)O[C@@H](C)[C@H](O[C@@H]2O[C@@H](C)[C@H](OC(=O)CC)[C@@H](OC(=O)C(C)C(C)C)O)[C@H]2O[C@H]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O)[C@H]1O
ChEMBL_29: CC1=CC[C@@H]([C@@H](C)[C@H]2CC[C@@]3(C)[C@@H]4CC[C@H]5[C@](C)(C(=O)O)[C@@H](O[C@@H]6O[C@H](CO)[C@H](O)[C@H]6O[C@H]6O[C@@H](O)[C@H](O)[C@H]6O)CC[C@]5(C)[C@H]4CC[C@]3(C)[C@H]2[C@@H]1C)C(C)C
ChEMBL_15: NCC(=O)N[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O)O)C(=O)N[C@@H](Cc1ccccc1)C(=O)NCC(=O)Nc1ccccc1C(=O)NCC(=O)N[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O)O)C(=O)N[C@@H](Cc1ccccc1)C(=O)O
ChEMBL_102: CCOC(=O)[C@H]1[C@H](C)O[C@@H](O[C@H]2[C@H](N(C)C)[C@@H](O)[C@H](O[C@H]3[C@@H](CC=O)C[C@@H](C)[C@H](O)CN(C)CCC[C@H](CCCc4ccnc5ccccc45)OC(=O)C[C@@H](OC(=O)CC)[C@@H]3OC)O[C@@H]2C)C[C@@]1(C)O
ChEMBL_116: C[C@H](NC(=O)[C@H](CCCCN)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCC(=O)O)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H]1CCCN1C(=O)[C@@H](N)[C@@H](C)O)C(=O)N[C@@H](CCCNC(=N)N)C(=O)NCC(=O)O
ChEMBL_114: C=C(C)[C@@H]1CC[C@]2(C(=O)O)CC[C@]3(C)[C@H](CC[C@@H]4[C@@]5(C)CC[C@H](OC(=O)CCC/C=C\C/C=C\C/C=C\C/C=C\C/C=C\CC)C(C)(C)[C@@H]5CC[C@]43C)[C@@H]12
ChEMBL_128: C[C@@H]1O[C@@H](O[C@H]2[C@H](O[C@H]3CC[C@]4(C)[C@H]5CC[C@@H]6[C@@H]([C@]7(C)CC(O)C(O)/C=C/C(C)(C)O)CC[C@@]6(C)[C@]5(C)C[C@@H](O)C[C@H]4C3(C)C)O[C@H](CO)[C@@H](O)[C@@H]2O)[C@H](O)[C@H](O)[C@H]1O
ChEMBL_17: COc1cc2c(O[C@@H]3OC[C@](O)(CO[C@@H]4OC[C@@H](O)[C@H](O)[C@H]4O)[C@H](O)[C@H]3O)[C@H]3O)c3c(c(-c4ccc5c(c4)OCO5)c2cc1OC)C(=O)OC3
ChEMBL_16: NCCCC[C@@H](C=O)NC(=O)CC(CCCN)NC(=O)CC(CCCN)NC(=O)CC(CCCN)NC(=O)CC(CCCN)NC(=O)CC(CCCN)NC(=O)CC(CCCN)NC(=O)CC(CCCN)NC(=O)CC(N)CCCN
ChEMBL_115: CCCCCCCCO[C@@H]1O[C@@H](C)[C@H](OC(=O)CCCCC)[C@@H](O[C@@H]2O[C@@H](C)[C@H](OC(C)=O)[C@@H](O[C@@H]3O[C@@H](C)[C@H](OC(C)=O)[C@@H](O)[C@H]3O)[C@H]2OC(C)=O)[C@H]1O
ChEMBL_101: C[C@H](NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCC(=O)O)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H]1CCCN1C(=O)[C@@H](N)[C@@H](C)O)C(=O)N[C@@H](CCCNC(=N)N)C(=O)NCC(=O)O

ChEMBL_124: CCC(=O)O[C@@H]1CC(=O)O[C@H](C)C[C@H](O)[C@H](N(C)CCCc2ccncc2)/C=C/[C@H](OC(C)=O)[C@H](C)C[C@H](CC=O)[C@H](O[C@@H]2O[C@H](C)[C@@H](O)[C@H](N(C)C)[C@H]2O)[C@H]1OC
ChEMBL_130: CC(C)(O)[C@@H]1CC[C@@](C)([C@H]2CC[C@]3(C)[C@@H]2[C@H](O)C[C@@H]2[C@@]4(C)CC[C@H](O)C(C)(C)[C@@H]4[C@@H](O[C@@H]4O[C@H](CO)[C@@H](O)[C@H](O)[C@H]4O[C@@H]4OC[C@@H](O)[C@H](O)[C@H]4O)C[C@]23C)O1
ChEMBL_33: C[C@@H]1CC[C@@]2(OC1)O[C@H]1C[C@H]3[C@@H]4CC=C5C[C@@H](O[C@@H]6O[C@H](CO)[C@@H](O[C@@H]7OC[C@H](O)[C@@H](O[C@@H]8O[C@@H](C)[C@H](O[C@@H]9O[C@@H](C)[C@H](O)[C@@H](O)[C@H]9O)[C@@H](O)[C@H]8O)[C@@H]7O)[C@H](O)[C@H]6O[C@@H]6O[C@@H](C)[C@H](O)[C@@H](O)[C@H]6O)CC[C@]5(C)[C@H]4CC[C@]3(C)[C@H]1[C@@H]2C
ChEMBL_32: CC(=O)OC[C@H]1O[C@@H](O[C@H]2CC[C@@]3(C)C(=C[C@H](O)[C@]4(C)[C@@H]3C[C@@H](O)[C@@H]3[C@@H]([C@](C)(CCC=C(C)C)O[C@@H]5O[C@H](CO[C@@H]6OC[C@@H](O)[C@H](O)[C@H]6O)[C@@H](O)[C@H](O)[C@H]5O)CC[C@]34C)C2(C)C)[C@H](O[C@@H]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O)[C@@H](O)[C@@H]1O
ChEMBL_131: CCC(=O)[C@H]1C[C@@H](C)[C@]2(CC(=O)[C@@]3(C)C4=C(CC[C@@]32C)[C@@]2(C)CC[C@H](O[C@@H]3O[C@H](CO[C@@H]5OC[C@H](O)[C@H](O)[C@H]5O[C@@H](CO)[C@@H](O)[C@H](O[C@@H]6OC[C@H](O)[C@H](O)[C@H]6O[C@@H]6OC[C@@H](O)[C@H](O)[C@H]6O)[C@H]5O[C@@H]5O[C@@H](C)[C@H](O)[C@@H](O)[C@H]5O)[C@@H](O)[C@H]3O)[C@]4(C)CO)[C@@H]2CC4)O1
ChEMBL_119: CCC(=O)O[C@H]1C(=O)[C@@]2(C)[C@H]([C@H](OC(=O)c3ccccc3)[C@]3(O)C[C@H](OC(=O)[C@H](O)[C@H](C=C(C)C)NC(=O)OC4CCCC4)C(C)=C1C3(C)C)[C@]1(OC(C)=O)CO[C@@H]1C[C@@H]2O
ChEMBL_133: C=C(C)[C@@H]1CC[C@]2(C)CC[C@]3(C)[C@H](CC[C@@H]4[C@@]5(C)CC[C@H](O[C@@H]6O[C@H](CO)[C@@H](O[C@@H]7O[C@@H](C)[C@H](O)[C@H](O)[C@H]7O)[C@H](O)[C@H]6O[C@@H]6O[C@@H](C)[C@H](O)[C@@H](O)[C@H]6O)C(C)(C)[C@@H]5CC[C@]43C)[C@@H]12
ChEMBL_127: COC(=O)CCCCC(=O)NCC(=O)NC(COCOCCOCCOCCOCCO[C@@H]1O[C@H](CO)[C@H](O)[C@H]1O)(COCOCCOCCOCCOCCO[C@@H]1O[C@H](CO)[C@H](O)[C@H]1O)COCOCCOCCOCCOCCO[C@@H]1O[C@H](CO)[C@H](O)[C@H]1O
ChEMBL_30: C[C@@H]1C(=O)O[C@H]2C[C@H]3[C@@H]4CC=C5C[C@@H](O[C@@H]6O[C@H](CO)[C@H](O[C@@H]7O[C@H](CO)[C@@H](O)[C@H](O[C@@H]8OC[C@@H](O)[C@H](O)[C@H]8O)[C@H]7O[C@@H]7O[C@H](CO)[C@@H](O)[C@H](O[C@@H]8O[C@H](CO)[C@@H](O)[C@H](O)[C@H]8O)[C@H]7O)[C@H](O)[C@H]6O)[C@H](O)C[C@]5(C)[C@H]4CC[C@]3(C)[C@H]21
ChEMBL_24: C=C(C)[C@@H]1CC[C@]2(CO[C@@H]3O[C@H](CO)[C@@H](O)[C@H](O)[C@H]3O)CC[C@]3(C)[C@H](CC[C@@H]4[C@@]5(C)CC[C@H](O[C@@H]6O[C@@H](C)[C@H](O)[C@@H](O)[C@H]6O)C(C)[C@@H](O)[C@@H]6O[C@@H](C)[C@@H]5CC[C@]43C)[C@@H]12
ChEMBL_19: CC(C)(C)OC(=O)N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H](Cc1c[nH]c2ccccc12)C(=O)N[C@@H](CCCCNC(=O)OCc1ccccc1)C(=O)O
ChEMBL_126: CC(=O)O[C@@]12CO[C@@H]1C[C@H](OC(=O)CN1CCN(Cc3ccc4c(c3)OCO4)CC1)[C@@]1(C)C(=O)[C@H](O)C3=C(C)[C@@H](OC(=O)[C@H](O)[C@@H](NC(=O)c4ccccc4)c4ccccc4)C[C@@](O)([C@@H](OC(=O)c4ccccc4)[C@H]21)C3(C)C
ChEMBL_132: O=C(O)CCCCCCCCCCNC(=O)COc1c(-c2cc3ccccc3n2Cc2ccccc2)n(Cc2ccccc2)c2ccccc12
ChEMBL_136: CC(=O)N[C@@H](CC(C)C)C(=O)N1CCC[C@H]1CC(=O)N[C@@H](Cc1ccccc1)C(=O)N[C@@H](Cc1ccccc1)C(=O)N[C@@H](CC(=O)O)C(N)=O
ChEMBL_21: C[C@H]1[C@@H](OCC(CO[C@H]2O[C@@H]3O[C@@]4(C)CC[C@H]5[C@H](C)CC[C@@H]([C@H]2C)[C@@]35OO4)OC(=O)CCC(=O)O)O[C@@H]2O[C@@]3(C)CC[C@H]4[C@H](C)CC[C@@H]1[C@@]24OO3
ChEMBL_20: C[C@H](NC(=O)CNC(=O)[C@@H](NC(=O)[C@@H](Cc1ccccc1)NC(=O)CNC(=O)CNC(=O)[C@@H](N)Cc1ccccc1)[C@@H](C)O)C(=O)N[C@@H](CCCN=C(N)N)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CO)C(=O)N[C@@H](C)C(=O)N[C@@H](CCCN=C(N)N)C(=O)N[C@@H](CCCCN)C(N)=O
ChEMBL_34: C[C@H](NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H]1CCCN1C(=O)[C@@H](N)[C@@H](C)O)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCNC(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@H](C(N)=O)[C@@H](C)O
ChEMBL_123: CC(=O)N[C@@H](Cc1cnc[nH]1)C(=O)N[C@H](Cc1ccc(C)cc1)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](Cc1c[nH]c2ccccc12)C(N)=O
ChEMBL_22: O=C(c1cccc2c(NCCCCCCNc3c4ccccc4nc4c(C(=O)N5CCOCC5)cccc34)c3ccccc3nc12)N1CCOCC1
ChEMBL_37: CC(C)C[C@H](NC(=O)[C@](C)(CCCC(=O)O)NC(=O)OCC1c2ccccc2-c2ccccc21)C(=O)N[C@@H](Cc1ccc(O)c(N)c1)C(=O)NC1(C(=O)N[C@@H](CC(N)=O)C(N)=O)CCCCC1
ChEMBL_23: C[C@H]1[C@H](C)CC[C@]2(C(=O)O)CC[C@]3(C(=O)O)C(=CC[C@@H]4[C@@]5(C)CC[C@H](O[C@@H]6O[C@H](CO)[C@@H](O)[C@H](O[C@@H]7O[C@H](CO)[C@@H](O)[C@H](O)[C@H]7O)[C@H]6O)C(C)(C)[C@@H]5CC[C@]43C)[C@H]12
ChEMBL_108: CC[C@H]1OC(=O)[C@H](C)[C@@H](O[C@H]2C[C@@](C)(OC)[C@@H](O)[C@H](C)O2)[C@H](C)[C@@H](O[C@@H]2O[C@H](C)C[C@H]3[C@H]2OC(=O)N3C)[C@](C)(O)C[C@@H](C)CN[C@H](C)[C@@H]2OC(=O)O[C@]12C
ChEMBL_134: O=c1c(O[C@@H]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O[C@@H]2OC[C@@H](O)[C@H](O)[C@H]2O)c(-c2ccc(O[C@@H]3O[C@H](CO)[C@@H](O)[C@H]3O)cc2)oc2cc(O)cc(O)c12
ChEMBL_184: CC[C@H](C)[C@@H]([C@@H](CC(=O)N1CCC[C@H]1[C@H](OC)[C@@H](C)C(=O)Nc1cc(/C=C\c2cc(OC)c3c(c2)OCO3)ccc1OC)OC)N(C)C(=O)[C@@H](NC(=O)[C@H](C(C)C)N(C)C)C(C)C
ChEMBL_147: CC(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CC(=O)O)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CC(=O)O)C(=O)N[C@@H](CCCNC(=N)N)C(=O)O
ChEMBL_93: CC(C)[C@H](NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H](N)CCCNC(=N)N)C(=O)N[C@@H](CCC(N)=O)C(=O)N[C@@H](CCCNC(=N)N)C(=O)NCC(=O)N[C@@H](CCCCN)C(=O)N[C@@H](Cc1c[nH]c2ccccc12)C(=O)N[C@@H](Cc1c[nH]c2ccccc12)C(=O)O
ChEMBL_78: C[C@@H]1O[C@@H](O[C@H]2[C@H](O)[C@@H](O)[C@H](OC[C@H]3O[C@@H](OC(=O)[C@]45CCC(C)(C)C[C@H]4C4=CC[C@@H]6[C@@]7(C)CC[C@H](O)[C@@](C)(CO)[C@@H]7CC[C@@]6(C)[C@]4(C)CC5)[C@H](O)[C@@H](O)[C@@H]3O)O[C@@H]2CO)[C@H](O)[C@H](O)[C@H]1O
ChEMBL_50: CCCc1ccc(C(=O)N[C@@H](c2ccccc2)[C@@H](O)C(=O)O[C@H]2C[C@@]3(O)[C@@H](OC(=O)c4ccccc4)[C@@H]4[C@]5(OC(C)=O)CO[C@@H]5C[C@H](O)[C@@]4(C)C(=O)[C@H](OC(C)=O)C(=C2C)C3(C)C)cc1
ChEMBL_79: CC[C@H](C)[C@@H](OC(=O)[C@@H](NC(=O)[C@H](C)OCc1ccccc1)C(C)C)C(=O)N[C@@H](C(=O)O[C@@H](C)C(=O)N[C@H](C(=O)O[C@@H](C(=O)N[C@@H](C(=O)O[C@@H](C)C(=O)N[C@@H](C(=O)O[C@H](CC(C)C)C(=O)N[C@@H](C(=O)OC(C)(C)C)C(C)C)C(C)C)C(C)C)[C@@H](C)CC)C(C)C)C(C)C
ChEMBL_92: C/C=C(\C)C(=O)O[C@H]1[C@H](O)[C@@H](O)[C@H](O[C@@H]2[C@@H](O)[C@@H](OC(=O)C(C)CC)[C@H](O[C@H]3[C@H](O[C@H]4[C@H](O[C@H](CCCCC)CCCCCCCCC(=O)O)O[C@H](C)[C@@H](O)[C@@H]4O)O[C@H](CO)[C@@H](O)[C@@H]3O)O[C@H]2C)O[C@@H]1C
ChEMBL_152: C/C=C(/C)C(=O)O[C@H]1[C@H](OC(=O)C(C)C)[C@]2(CO)[C@H](O)[C@H](O)[C@]3(C)C=CC[C@@H]4[C@@]5(C)CC[C@H](O[C@@H]6O[C@H](C(=O)O)[C@@H](O)[C@H](O[C@@H]7O[C@@H](CO)[C@H](O)[C@H]7O)[C@H]6O[C@@H]6O[C@H](CO)[C@H](O)[C@H]6O)[C@@H](C)[C@@H]5CC[C@]43C)[C@@H]2CC1(C)C
ChEMBL_191: CO[C@]1(CC[C@@H](C)CO)O[C@H]2C[C@H]3[C@@H]4CC=C5C[C@@H](O[C@H]6O[C@H](CO)[C@@H](O)[C@H](O[C@@H]7O[C@H](CO)[C@@H](O)[C@H](O)[C@H]7O)[C@H]6O[C@@H]6O[C@@H](C)[C@H](O)[C@@H](O)[C@H]6O)CC[C@]5(C)[C@H]4CC[C@]3(C)[C@H]2[C@@H]1C
ChEMBL_185: CN1CC[C@]2(c3cccc4c3C[C@H]3N(C)CC[C@@]45[C@@H]4Nc6c([C@]78CCN(C)[C@H]7Nc7ccccc78)cccc6[C@]35CCN4C)c3ccccc3N[C@H]12
ChEMBL_193: CC(C)[C@H](NC(=O)[C@H](C)NC(=O)[C@H](Cc1ccccc1)NC(=O)[C@H](Cc1ccccc1)NC(=O)/C=C/C(=O)NCC(=O)NCC(=O)N[C@@H](Cc1ccccc1)C(=O)O)C(N)=O
ChEMBL_187: CO[C@H]1[C@H](OC[C@@H]2O[C@@H](O[C@@H](CC[C@@H](C)[C@H]3C[C@@H](O)[C@@H]4[C@]3(C)CC[C@H](O)C4=CC[C@@H]5[C@]6(C)CC[C@H](O)[C@@](C)(CO)[C@@H]6CC[C@@]5(C)C=O)C(C)C)O)[C@H](O)[C@H]2O)[C@H](O)[C@H]1O
ChEMBL_150: CO[C@]1(CC[C@H](C)CO[C@@H]2O[C@H](CO[C@@H]3O[C@H](CO)[C@@H](O)[C@H](O)[C@H]3O)[C@@H](O)[C@H](O)[C@H]2O)O[C@H]2C[C@H]3[C@@H]4CC=C5C[C@@H](O[C@@H]6O[C@H](CO)[C@@H](O)[C@H](O[C@@H]7O[C@@H](C)[C@H](O[C@@H]8O[C@H](CO)[C@@H](O)[C@H](O)[C@H]8O)[C@@H](O)[C@H]7O)[C@H]6O[C@@H]6O[C@@H](C)[C@H](O)[C@@H](O)[C@H]6O)CC[C@]5(C)[C@H]4CC[C@]3(C)[C@H]2[C@@H]1C
ChEMBL_144: CC(C)C[C@H](NC(=O)[C@H](CC(C)C)NC(=O)[C@@H](Cc1c[nH]c2ccccc12)NC(=O)[C@H](Cc1ccccc1)NC(=O)[C@@H](Cc1c[nH]c2ccccc12)NC(=O)[C@@H](N)Cc1cnc[nH]1)C(N)=O
ChEMBL_90: CCCCCCCCOc1c2cc(C(=O)O)cc1Cc1cc(C(=O)O)cc(c1OCCCCCCCC)Cc1cc(C(=O)O)cc(c1OCCCCCCCC)Cc1cc(cc(C(=O)O)c1)C2
ChEMBL_84: C=C1CO[C@@]2(O[C@H]3C[C@H]4[C@@H]5CC=C6C[C@@H](O)C[C@@H](O[C@@H]7OC[C@H](O)[C@H](O)[C@H]7O[C@@H]7O[C@@H](C)[C@H](OC(C)=O)[C@@H](O)[C@H]7O)[C@@]6(C)C[C@H](OC(C)=O)[C@H]5[C@@]5(C)C(=O)[C@H](O)[C@H](O)[C@H]1O
ChEMBL_53: C[C@@H]1O[C@@H](O[C@H]2[C@H](OC(=O)[C@]34CCC(C)(C)C[C@H]3C3=CC[C@@H]5[C@@]6(C)C[C@H](O)[C@H](O)[C@@](C)(CO)[C@@H]6[C@H](O)C[C@@]5(C)[C@]3(C)CC4)OC[C@H](O)[C@@H]2O)[C@H](O)[C@H]1O
ChEMBL_47: CC(C)[C@H](NC(=O)[C@H](Cc1c[nH]c2ccccc12)NC(=O)[C@@H](NC(=O)[C@H](CCCCN)NC(=O)[C@H](Cc1ccc(O)cc1)NC(=O)[C@H](Cc1ccccc1)NC(=O)[C@H](CCCCN)NC(=O)CN)C(C)C)C(=O)N[C@@H](CCCNC(=N)N)C(=O)O
ChEMBL_46: C=C(C)[C@@H]1CC[C@]2(C(=O)O)CC[C@]3(C)[C@H](CC[C@@H]4[C@@]5(C)CC[C@H](O[C@@H]6OC[C@H](O[C@@H]7O[C@H](CO)[C@@H](O)[C@H](O)[C@H]7O)[C@H](O)[C@H]6O[C@@H]6O[C@H](CO)[C@@H](O)[C@H](O)[C@H]6O)C(C)(C)[C@@H]5CC[C@]43C)[C@@H]12

ChEMBL_52: O=c1cc(-c2ccc(OCCOCCOCCOCCOCCOCCOCCOCCOCCOCCOCCOCCOCCOc3ccc(-c4cc(=O)c5ccccc5o4)cc3)cc2)oc2ccccc12

ChEMBL_85: CCC(=O)O[C@H]1C(=O)[C@@]2(C)[C@H]([C@H](OC(=O)c3cccc(OC)c3)[C@]3(O)C[C@H](OC(=O)[C@H](O)[C@@H](NC(=O)OC(C)(C)C)[C@H]4CC4(C)C)C(C)=C1C3(C)C)[C@]1(OC(C)=O)CO[C@@H]1C[C@@H]2O

ChEMBL_91: COc1ccc(C[C@H]2C(=O)N[C@@H](C)C(=O)N(C)[C@H]3Cc4ccc(c(O)c4)Oc4cc(ccc4OC)C[C@@H](C(=O)N[C@H](C)C(=O)N[C@@H](C)C(=O)N2C)N(C)C3=O)cc1

ChEMBL_145: CCCCCCCCCC(=O)O[C@@H]1[C@@H](O)[C@@]2(CCCC(OC(C)=O)C(C)C/C=C/c3ccccc3)O[C@H](C(=O)O)[C@@](O)(C(=O)O)[C@]1(C(=O)O)O2

ChEMBL_179: Cc1nc2c(=O)[nH]c1CCC(=O)N[C@@H](Cc1ccccc1)C(=O)N[C@H](Cc1c[nH]c3ccccc13)C(=O)N[C@@H](CCCCN)C(=O)NCCCC2

ChEMBL_186: C[C@H](NC(=O)[C@H](C)NC(=O)[C@H](CCC(=O)O)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H]1CCCN1C(=O)[C@@H](N)[C@@H](C)O)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCNC(=N)N)C(=O)NCC(=O)O

ChEMBL_8: CCCCCC[C@H](N)C(=O)N[C@@H](Cc1ccc(O)cc1)C(=O)N1CCC[C@H]1C(=O)N[C@@H](Cc1c[nH]c2ccccc12)C(=O)N[C@@H](Cc1ccccc1)C(N)=O

ChEMBL_192: CN(C)c1ccc(NC(=O)c2cc(NC(=O)c3ccc(C(=O)c4ccc(C(=O)Nc5cc(C(=O)Nc6ccc(N(C)C)cc6)n(C)c5)cc4)cc3)cn2C)cc1

ChEMBL_196: O=C1C=C(c2ccc(OCCOCCOCCOCCOCCOCCOc3ccc(-c4cc(=O)c5ccccc5o4)cc3)cc2)Cc2ccccc21

ChEMBL_182: C/C=C/C(=O)O[C@H]1C(=O)[C@]2(C)[C@@H](OC(=O)c3ccccc3)C[C@H]3OC[C@@]3(OC(C)=O)[C@H]2[C@H](OC(=O)c2ccccc2)[C@]2(O)C[C@H](OC(=O)[C@H](O)[C@@H](NC(=O)c3ccccc3)c3ccccc3)C(C)=C1C2(C)C

ChEMBL_141: CC/C=C\C/C=C\C/C=C\CCCCCCCC(=O)OC[C@H](CO[C@@H]1O[C@H](CO)[C@H](O)[C@H](O)[C@H]1O)OC(=O)CCCCCCC/C=C\C/C=C\C/C=C\CC

ChEMBL_169: N=C(N)NCCC[C@H](NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](Cc1c[nH]c2ccccc12)NC(=O)[C@H](CCCCN)NC(=O)[C@@H](N)CCCNC(=N)N)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](Cc1c[nH]c2ccccc12)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCCN)C(N)=O

ChEMBL_56: CN(C)CCNC(=O)c1cccc2c(NCCN3CCN(CCNc4c5ccccc5nc5c(C(=O)NCCN(C)C)cccc45)CC3)c3ccccc3nc12

ChEMBL_42: CN[C@H](CC(C)C)C(=O)N[C@H]1C(=O)N[C@@H](CC(N)=O)C(=O)N[C@H]2C(=O)N[C@H]3C(=O)N[C@H](C(=O)N[C@H](C(=O)NCCCCCCCCCCN)c4cc(O)cc(O)c4-c4cc3ccc4O)[C@H](O)c3ccc(cc3)Oc3cc2cc(c3O)Oc2ccc(cc2)[C@H]1O

ChEMBL_43: COc1cc([C@H]2[C@H](C(=O)O[C@@H]3C[C@@H]4[C@@H](CN(C)C[C@@H]4C)[C@@H]3C)[C@H](c3ccc(O)c(OC)c3)[C@H]2C(=O)O[C@@H]2C[C@@H]3[C@@H](CN(C)C[C@@H]3C)[C@@H]2C)ccc1O

ChEMBL_57: C[C@H](NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H]1CCCN1C(=O)[C@@H](N)[C@@H](C)O)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CO)C(=O)O

ChEMBL_94: CC[C@@]1(O)C[C@H](O[C@H]2C[C@H](N(C)C)[C@H](O[C@H]3C[C@H](O)[C@H](O[C@@H]4C[C@H](O)[C@H](O)[C@H](C)O4)[C@H](C)O3)[C@@H](C)O2)c2c(cc3c(c2O)C(=O)c2c(O)ccc(O)c2C3=O)[C@@H]1C(=O)OC

ChEMBL_140: CC[C@]12C=CCN3CC[C@@]4(c5ccc(OC)cc5N(C)[C@H]4[C@@](O)(C(=O)OC)[C@@H]1OC(=O)CCCCCCCCC(=O)O[C@H]1c4cc5c(cc4[C@@H](c4cc(OC)c(OC)c(OC)c4)[C@H]4C(=O)OC[C@@H]41)OCO5)[C@@H]32

ChEMBL_154: CC(C)[C@H](NC(=O)[C@H](CCCCN)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CC(N)=O)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H](N)Cc1c[nH]c2ccccc12)C(=O)N[C@H](C(=O)N[C@@H](Cc1c[nH]c2ccccc12)C(=O)N[C@@H](CCCNC(=N)N)C(=O)O)C(C)C

ChEMBL_183: CN1CCC(NC(=O)c2cccc3c(NCCCCCCCCNc4c5ccccc5nc5c(C(=O)NC6CCN(C)CC6)cccc45)c4ccccc4nc23)CC1

ChEMBL_197: Nc1ccccc1NC(=O)c1ccc(CNC(=O)CCCCC(=O)NCCCNCCCCNCCC(c2ccccc2)c2ccccc2)cc1

ChEMBL_181: Cc1c2oc3c(C)ccc(C(=O)N[C@@H]4C(=O)N[C@H](C(C)C)C(=O)N5[C@H](C(=O)N(C)CC(=O)N(C)[C@@H](C(C)C)C(=O)O[C@@H]4C)[C@@H](O)C[C@@H]5C)c3nc-2c(C(=O)N[C@@H]2C(=O)N[C@H](C(C)C)C(=O)N3CCC[C@H]3C(=O)N(C)CC(=O)N(C)[C@@H](C)C(=O)O[C@@H]2C)c(N)c1=O

ChEMBL_195: C[C@H](NC(=O)[C@H](CCCCN)NC(=O)[C@H](CCCCN)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@H](CCC(=O)O)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H]1CCCN1C(=O)[C@@H](N)[C@@H](C)O)C(=O)NCC(=O)O

ChEMBL_142: C=CC1CN(C)[C@H]2Cc3c([nH]c4ccccc34)[C@H](c3c(OC)ccc4c5c([nH]c34)[C@]3(C(=O)OC)C[C@@H]4C[C@H](CC)[C@@H]3N(CC5)C4O)C[C@H]1C2C(=O)OC

ChEMBL_82: CC(C)CCC(=O)[C@@H](C)[C@@]1(O)[C@@H](O[C@@H]2OC[C@H](O)[C@H](O)[C@H]2O)C[C@H]2[C@@H]3CC=C4C[C@@H](O[C@@H]5O[C@H](CO)[C@@H](O)[C@H](O)[C@H]5O)CC[C@]4(C)[C@H]3CC[C@@]21C

ChEMBL_96: C[C@]12CC[C@@H]3c4ccc(OCC(=O)N[C@@H](CCCNC(=N)N)C(=O)NCC(=O)N[C@@H](CC(=O)O)C(=O)N[C@@H](Cc5ccccc5)C(=O)N[C@@H](CCCNC(=N)N)C(=O)NCC(=O)N[C@@H](CC(=O)O)C(=O)N[C@@H](Cc5ccccc5)C(=O)O)cc4CC[C@H]3[C@@H]1CC[C@@H]2O

ChEMBL_41: CC(=O)O[C@@]12CO[C@@H]1C[C@H](O)[C@@]1(C)C(=O)[C@H](OC(=O)COc3ccccc3)C3=C(C)[C@@H](OC(=O)C(O)C(NC(=O)c4ccccc4)c4ccccc4)C[C@@](O)([C@@H](OC(=O)c4ccccc4)C12)C3(C)C

ChEMBL_55: CC1=C(CC[C@H](C)CO[C@@H]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O)O[C@H]2C[C@H]3[C@@H]4CC[C@@H]5C[C@@H](O[C@@H]6O[C@H](CO)[C@H](O)[C@H](O)[C@H]6O[C@@H]6O[C@H](CO)[C@@H](O)[C@H](O)[C@H]6O)CC[C@]5(C)[C@H]4CC[C@]3(C)[C@@H]12

ChEMBL_69: CC(C)CC[C@H](O)[C@@H](C)[C@H]1[C@@H](O[C@@H]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O)C[C@H]2[C@@H]3CC=C4C[C@@H](O)C[C@@H](O[C@@H]5O[C@@H](C)[C@H](O)[C@@H](O)[C@H]5O)[C@]4(C)[C@H]3CC[C@]12C

ChEMBL_68: O=C(O)C[C@@H](C(=O)O)n1cc(CCCC(=O)OCOc2n/c(=C\c3cccc(C(=O)c4ccccc4)c3)c(=O)[nH]/c2=C\c2ccccn2)nn1

ChEMBL_54: C[C@H](NC(=O)[C@H](CCCCNC(N)=O)NC(=O)CNC(=O)CNC(=O)[C@@H](NC(=O)[C@H](CO)NC(=O)[C@@H](N)CCCCN)[C@@H](C)O)C(=O)N1CCC[C@H]1C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCNC(=N)N)C(=O)NCC(=O)O

ChEMBL_97: C[C@H](NC(=O)[C@H](C)NC(=O)[C@H](CCCNC(=N)N)NC(=O)[C@@H]1CCCN1C(=O)[C@@H](N)[C@@H](C)O)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCNC(=N)N)C(=O)O

ChEMBL_83: CC(=O)N(C)[C@@H](CC1CCCC1)C(=O)N(C)[C@@H](Cc1ccccc1)C(=O)N(C)[C@@H](Cc1ccccc1)C(N)=O

ChEMBL_143: C=C(C)[C@@H]1CC[C@]2(C(=O)N[C@H]3CCCc4ccccc43)CC[C@]3(C)[C@H](CC[C@@H]4[C@@]5(C)CC[C@H](OC(=O)CC(C)(C)C(=O)O)C(C)(C)[C@@H]5CC[C@]43C)[C@@H]12

ChEMBL_194: CC(C)C[C@H](NC(=O)[C@H](Cc1c[nH]c2ccccc12)NC(=O)[C@@H](N)CCCNC(=N)N)C(=O)N[C@H](C(=O)N[C@@H](CC(C)C)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](CCCCN)C(=O)N[C@@H](CCCNC(=N)N)C(=O)N[C@@H](Cc1c[nH]c2ccccc12)C(=O)O)C(C)C

ChEMBL_180: CO[C@H]1C[C@H](O[C@H]2CC[C@@]3(C)C(=CC[C@]4(O)[C@@H]3C[C@@H](OC(=O)/C=C/c3ccccc3)[C@@]3(C)[C@]4(O)CC[C@]3(O)[C@H](C)O)C2)O[C@H](C)[C@H]1O[C@H]1C[C@H](OC)[C@H](O[C@H]2C[C@H](OC)[C@H](O[C@@H]3O[C@H](CO[C@@H]4O[C@H](CO)[C@@H](O)[C@H](O)[C@H]4O)[C@@H](O)[C@H](O)[C@H]3O)[C@@H](C)O2)[C@@H](C)O1

Appendix E

Selected Molecules in ZINC20-based Datasets

The instruction of obtaining original ZINC20 database is given in Chapter 7. Here, we list the name of files that we used for different datasets. Please be noted that, a filtering operation has been applied to these molecules for limiting the element and dropping molecules with error (see Section 3.3.1).

E.1 Training Set

Corresponding datasets: SE_Train.

"CAAAMN.xaa", "CAAARN.xaa", "CAABMN.xaa", "CAABRN.xaa", "CAACRN.xaa", "CAADMN.xaa", "CAADRN.xaa", "CABAMN.xaa", "CABARN.xaa", "CABBRN.xaa", "CABDMN.xaa", "CABDRN.xaa", "CACAMN.xaa", "CACARN.xaa", "CACBMN.xaa", "CACBRN.xaa", "CACCRN.xaa", "CACDMN.xaa", "CACDRN.xaa", "CAEAMN.xaa", "CAEARN.xaa", "CAEBMN.xaa", "CAEBRN.xaa", "CAEDMN.xaa", "CAEDRN.xaa", "CBAAMN.xaa", "CBAARN.xaa", "CBABMN.xaa", "CBABRN.xaa", "CBADMN.xaa", "CBADMN.xab", "CBADMN.xac", "CBADMN.xad", "CBADMN.xae", "CBADRN.xaa" and "CBADRN.xab".

E.2 Testing Set

Corresponding datasets: SE_Test.

"CBADRN.xac"

Appendix F

Comparison of Computational Cost of Def2SVP and 6-31G(d) Basis Sets

Molecule	CPU time (hh:mm:ss)		RCC
	6-31G(d)	Def2SVP	
ChEMBL_41	118:47:13	256:14:46	115.72%
ChEMBL_42	358:05:10	525:57:39	46.88%
ChEMBL_43	48:18:08	71:24:35	47.84%
ChEMBL_55	75:34:03	149:43:40	98.14%
ChEMBL_56	58:50:47	75:28:03	28.24%
ChEMBL_57	191:40:17	737:10:23	284.60%
ChEMBL_68	43:18:14	90:03:39	107.97%
ChEMBL_69	66:42:36	159:31:41	139.14%
ChEMBL_79	232:06:12	350:52:57	51.18%
ChEMBL_82	33:12:29	102:50:58	209.71%
ChEMBL_83	168:39:42	238:10:56	41.22%
ChEMBL_94	45:04:31	67:15:53	49.23%
ChEMBL_97	187:31:53	393:29:25	109.83%
ChEMBL_140	117:08:54	233:08:12	99.01%
ChEMBL_142	83:39:28	234:01:06	179.73%
ChEMBL_143	112:58:18	109:34:33	-3.01%
ChEMBL_169	632:32:40	309:34:30	-51.06%
ChEMBL_180	127:11:39	271:03:03	113.10%
ChEMBL_183	83:24:32	82:31:11	-1.07%
ChEMBL_194	171:42:06	352:16:18	105.16%
ChEMBL_195	325:59:58	558:23:59	71.29%
ChEMBL_196	13:15:58	38:19:59	188.95%
ChEMBL_197	11:21:42	24:20:12	114.20%
Average	143:47:14	236:09:02	93.31%

Appendix G

Python Script for Filtering ChEMBL-35 Database

```python
from rdkit import Chem
from rdkit.Chem import Descriptors
from rdkit.Chem import SDMolSupplier

# set allowed atoms
allowed_atoms = {'C', 'H', 'N', 'O'}
# set path to ChEMBL dataset
supplier = Chem.SDMolSupplier('./0_ChEMBL/chembl_35.sdf')
# set path to save smiles
save_path = './1_smiles/'
# set the number of molecules to save
num_molecules = 200

def is_gdb13_cno_like(mol):
    if mol is None:
        return False
    atoms = [atom.GetSymbol() for atom in mol.GetAtoms()]

    # 1. only C, H, N, O
    if any(atom not in allowed_atoms for atom in atoms):
        return False

    # 2. number of atoms in [50, 100] (inclusive)
    if (len(atoms) < 50) or (len(atoms) > 100):
        return False

    # 3. no charged molecules
    for atom in mol.GetAtoms():
        if atom.GetFormalCharge() != 0:
            return False

```

```
    # 4. must be a single connected component
    if '.' in Chem.MolToSmiles(mol):
        return False

    return True

smiles_list = []
count = 0
for mol in supplier:
    if mol is not None and is_gdb13_cno_like(mol):
        smiles = Chem.MolToSmiles(mol)
        smiles_list.append(smiles)
        count += 1
    if count == num_molecules:
        break

count = 1
for smi in smiles_list:
    mol_name = save_path + "ChEMBL_" + str(count) +".smi"
    with open(mol_name, 'w') as f:
        f.write(smi)
    count += 1
```

Appendix H

Program "prj_acdm" Example Input Files for the Operations of Mulliken-based Models in Project ACP

H.1 Training

File "training.in"

```
set_mt 10
set_py_version python3.10

# read training set and Mulliken charge
cppgd {
    # read charge
    new training_charge
    training_charge read_gaussian_log_dir /Users/xieqin/Desktop/PhD_Thesis/
        chpt3_dataset/database/2_gaussian_log_file_analysis/training_set
        Mulliken
    # read topology
    new training_topology
    training_topology read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/
        chpt3_dataset/database/6_mol2_conversion/train_mol2

    # delete parts of data, prepare for merge
    training_charge delete structure
    training_charge delete topology
    training_charge delete dipole_moment

    training_topology delete structure
    training_topology delete charge
```

```
    training_topology delete dipole_moment

    # merge for dataset
    merge training_charge training_topology training_set
}

# train the models and save
predac {
    new model_1 training_set
    # normalisation set
    model_1 set_norm true
    model_1 set_norm_style std
    # number of rounds of update
    model_1 set_cutoff 3
    # set training parameters
    # batch_size=16, epochs=100, layers=5, nodes= 50
    model_1 set_train_param 16 100 5 50
    # training
    model_1 train /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/model_Mulliken/
        model_1

    new model_2 training_set
    # normalisation set
    model_2 set_norm true
    model_2 set_norm_style std
    # number of rounds of update
    model_2 set_cutoff 3
    # set training parameters
    # batch_size=16, epochs=100, layers=5, nodes= 50
    model_2 set_train_param 16 100 5 50
    # training
    model_2 train /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/model_Mulliken/
        model_2

    new model_3 training_set
    # normalisation set
    model_3 set_norm true
    model_3 set_norm_style std
    # number of rounds of update
    model_3 set_cutoff 3
    # set training parameters
    # batch_size=16, epochs=100, layers=5, nodes= 50
    model_3 set_train_param 16 100 5 50
    # training
    model_3 train /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/model_Mulliken/
        model_3
}
```

H.2 Testing

File “testing.in”

```
set_mt 10
set_py_version python3.10

# read testing set and Mulliken charge
cppgd {
    # read charge
    new testing_charge
    testing_charge read_gaussian_log_dir /Users/xieqin/Desktop/PhD_Thesis/
        chpt3_dataset/database/2_gaussian_log_file_analysis/testing_set
        Mulliken
    # read topology
    new testing_topology
    testing_topology read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/
        chpt3_dataset/database/6_mol2_conversion/test_mol2

    # delete parts of data, prepare for merge
    testing_charge delete structure
    testing_charge delete topology
    testing_charge delete dipole_moment

    testing_topology delete structure
    testing_topology delete charge
    testing_topology delete dipole_moment

    # merge for dataset
    merge testing_charge testing_topology testing_set
}

# predict with trained model
predac {
    new model_1 testing_set
    model_1 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/model_1/model_1
    model_1 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/predicts

    new model_2 testing_set
    model_2 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/model_2/model_2
    model_2 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/predicts

    new model_3 testing_set
    model_3 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/model_3/model_3
```

```
    model_3 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/predicts

    model_1 save_original_charge /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/predicts
}
```

H.3 Larger Molecule Testing

File "LM_testing.in"

```
set_mt 10
set_py_version python3.10

# read testing set and Mulliken charge
cppgd {
    # read charge
    new testing_charge
    testing_charge read_gaussian_log_dir /Users/xieqin/Desktop/PhD_Thesis/
        chpt3_dataset/database/7_large_molecule_dataset/4_gaussian_log
        Mulliken
    # read topology
    new testing_topology
    testing_topology read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/
        chpt3_dataset/database/7_large_molecule_dataset/5
        _optimised_mol2_format

    # delete parts of data, prepare for merge
    testing_charge delete structure
    testing_charge delete topology
    testing_charge delete dipole_moment

    testing_topology delete structure
    testing_topology delete charge
    testing_topology delete dipole_moment

    # merge for dataset
    merge testing_charge testing_topology testing_set
}

# predict with trained model
predac {
    new model_1 testing_set
    model_1 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/model_1/model_1
    model_1 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/LM_predicts

```

```
32      new model_2 testing_set
33      model_2 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
            model_Mulliken/model_2/model_2
34      model_2 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
            model_Mulliken/LM_predicts
35
36      new model_3 testing_set
37      model_3 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
            model_Mulliken/model_3/model_3
38      model_3 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
            model_Mulliken/LM_predicts
39
40      model_1 save_original_charge /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
            model_Mulliken/LM_predicts
41  }
```

Appendix I

Charge Value (True Value) Inspection in Project ACP

I.1 Mulliken Charge

Dataset		**H**	**C**	**N**	**O**
"Mulliken_Test"	**Min Value** (e)	0.0476	-1.0517	-0.7952	-0.6741
	Max Value (e)	0.4541	1.0420	0.4037	-0.1752
"Mulliken_LM_Test"	**Min Value** (e)	0.0747	-0.5977	-0.8378	-0.7255
	Max Value (e)	0.4820	0.8408	-0.0691	-0.2957

I.2 Hirshfeld Charge

Dataset		**H**	**C**	**N**	**O**
"Hirshfeld_Test"	**Min Value** (e)	0.0099	-0.1427	-0.3147	-0.3538
	Max Value (e)	0.2038	0.2545	0.2582	0.0148
"Hirshfeld_LM_Test"	**Min Value** (e)	-0.0003	-0.1583	-0.3409	-0.3473
	Max Value (e)	0.1946	0.2578	0.0395	-0.0707

I.3 CM5 Charge

Dataset		**H**	**C**	**N**	**O**
"CM5_Test"	**Min Value** (e)	0.0728	-0.2386	-0.5911	-0.4743
	Max Value (e)	0.3961	0.3744	0.1064	0.0728
"CM5_LM_Test"	**Min Value** (e)	0.0673	-0.2567	-0.6837	-0.4692
	Max Value (e)	0.3681	0.3614	-0.1114	-0.1354

I.4 DDEC6 Charge

	H	C	N	O
Min Value (e)	-0.0363	-0.5090	-0.7688	-0.5881
Max Value (e)	0.4273	0.7544	0.6013	0.0723

I.5 SE Charge

Due to limitations in the data source, the precision here is limited to two decimal places.

	H	C	N	O	F	S	Cl	Br
Min Value (e)	-0.01	-0.89	-1.54	-1.07	-0.22	-0.66	-0.18	-0.10
Max Value (e)	0.48	0.76	0.13	0.39	-0.06	2.82	0.07	0.08

Appendix J

The Python Script of the Test Run in Project ACP for Cost of Time of MPF-based Models in Predicting Charge in a Large Single Molecule

```
import os

tf_model =
    "/Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/model_Mulliken/model_1/model_1"

def generate_alkane_mol2(n_carbons, output_file):
    smiles = 'C' * n_carbons
    cmd = f'obabel -:"{smiles}" -O {output_file} -h'
    os.system(cmd)
    print(f"{output_file} generated")

# genearate the mol2 files
n_list = [1, 500, 1000, 5000, 10000, 20000]
file_list = []
for n in n_list:
    output_file = f"C{n}.mol2"
    file_list.append(output_file)
    generate_alkane_mol2(n, output_file)

# loop and run prj_acdm
for file in file_list:
    in_file = open("temp.in", "w")
    in_file.write(
    "set_py_version python3.10\n"\
    "set_mt 1\n"\
    "cppgd {\n"\
```

```
    "\tnew data\n"\
    f"\tdata read_mol2_file {file}\n"\
    "\tdata print_info\n"\
    "}\n"
    "predac {\n"\
    "\tnew test data\n"\
    f"\ttest load_tf_model {tf_model}\n"\
    "\ttest predict 1 ./\n"\
    "}\n"\
    )
    in_file.close()
    os.system("prj_acdm temp.in")
    os.remove("temp.in")
```

Appendix K

The "prj_acdm" Input File of the Test Run in Project ACP for Cost of Time of MPF-based Models in Predicting Charge in a Large Dataset with Multiple Molecules

```
set_mt 10
set_py_version python3.10

cppgd {
    new zinc
    zinc read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/chpt3_dataset/
        database/ZINC_dataset/training_set
    zinc print_info
}

predac {
    new speed_test zinc
    speed_test load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        model_Mulliken/model_1/model_1
    speed_test predict 768 /Users/xieqin/Desktop/PhD_Thesis/chpt4_ACP/
        efficiency
    speed_test save_original_charge /Users/xieqin/Desktop/PhD_Thesis/
        chpt4_ACP/efficiency
}
```

Appendix L

Python Code of GLaCE Example for Ethylene

```python
from typing import Callable
import numpy as np

# Basic classes
class Node:
    def __init__(self, feature: int|np.ndarray):
        # here the feature list is defined as a list of atom ids
        self.feature = np.asarray(feature, dtype=int)

    def add_id (self, id: int):
        # the id here is used to identify the node in the graph
        self.id = id

    def __eq__(self, other):
        if isinstance(other, Node):
            return np.array_equal(self.feature, other.feature)
        return False

class Edge:
    def __init__(self, node_i: Node, node_j: Node, feature: int):
        self.node_i = node_i
        self.node_j = node_j
        self.feature = feature

    def __eq__(self, other):
        if isinstance(other, Edge):
            return (self.node_i == other.node_i and
                    self.node_j == other.node_j and
                    self.feature == other.feature)
        return False

```

```
class Graph:
    def __init__(self):
        self.__node_list = np.empty(0, dtype=Node)
        self.__edge_list = np.empty(0, dtype=Edge)

    def add_node(self, node: Node):
        id = self.__node_list.shape[0]
        node.add_id(id)
        self.__node_list = np.append(self.__node_list, node)

    def add_edge(self, edge: Edge):
        self.__edge_list = np.append(self.__edge_list, edge)

    def print(self):
        print("node feature:")
        for node in self.__node_list:
            print(node.id, node.feature)
        print("topology matrix:")
        print(self.topology)

    @property
    def topology(self):
        T = np.zeros((self.__node_list.size,
           self.__node_list.size), dtype=int)
        for edge in self.__edge_list:
            T[edge.node_i.id, edge.node_j.id] = edge.feature
            T[edge.node_j.id, edge.node_i.id] = edge.feature
        return T

    @property
    def Nodes(self):
        return self.__node_list

    @property
    def Edges(self):
        return self.__edge_list

# X: Graph; Phi: node formation function; I: interaction function
def glace(X: Graph, Phi: Callable[[Node, Node], Node], Xi:
    Callable[[Node, Node], int]) -> Graph:
    # get the topology matrix of Graph G
    T = X.topology
    T_size = T.shape[0]
    # Xp stands for the graph in next level
    Xp = Graph()

    # iterate over all topology matrix in X
    for i in range(T_size):
```

```
78          for j in range(i, T_size):
79              # if node_i and node_j are ajacent in Graph X
80              # then check and form a new node in Graph Xp
81              if T[i, j] != 0:
82                  new_node = Phi(X.Nodes[i], X.Nodes[j])
83                  if new_node is not None:
84                      # check if the new node is already in Xp
85                      if not np.any(Xp.Nodes == new_node):
86                          Xp.add_node(new_node)
87
88      # iterate over all possible pairs of nodes in Xp
89      Tp_size = Xp.Nodes.shape[0]
90      for i in range(Tp_size):
91          for j in range(i, Tp_size):
92              # check adjacency in Xp according to interaction
                    function
93              interaction = Xi(Xp.Nodes[i], Xp.Nodes[j])
94              if interaction:
95                  new_edge = Edge(Xp.Nodes[i], Xp.Nodes[j],
                        interaction)
96                  Xp.add_edge(new_edge)
97
98      return Xp
99
100 # Customized functions
101 # node formation function: define how a new node is formed
102 def node_formation(node_i: Node, node_j: Node) -> Node:
103     new_feature = np.unique(np.append(node_i.feature,
            node_j.feature))
104     if new_feature.size == node_i.feature.size + 1:
105         return Node(new_feature)
106     else:
107         return None
108
109 # interaction function: define how two nodes interact
110 # here the feature are always defined as share atom number
111 def interaction(node_i: Node, node_j: Node) -> int:
112     if (node_i.id == node_j.id):
113         return 0
114     sum_size = node_i.feature.shape[0] + node_j.feature.shape[0]
115     unique_size = np.unique(np.append(node_i.feature,
            node_j.feature)).shape[0]
116     return sum_size - unique_size
117
118 # generate example graph for C2H4
119 l1 = Graph()
120 # generate 6 atoms in the molecule
121 for i in range (6):
```

```
    l1.add_node(Node(i))

# generate bonds between atoms
# in C2H4, there are 4 single bonds and 1 double bonds
l1.add_edge(Edge(l1.Nodes[0], l1.Nodes[2], 1))
l1.add_edge(Edge(l1.Nodes[0], l1.Nodes[3], 1))
l1.add_edge(Edge(l1.Nodes[1], l1.Nodes[4], 1))
l1.add_edge(Edge(l1.Nodes[1], l1.Nodes[5], 1))
l1.add_edge(Edge(l1.Nodes[0], l1.Nodes[1], 2))

print("Graph L1:")
l1.print()

l2 = glace(l1, node_formation, interaction)
print("Graph L2:")
l2.print()

l3 = glace(l2, node_formation, interaction)
print("Graph L3:")
l3.print()

l4 = glace(l3, node_formation, interaction)
print("Graph L4:")
l4.print()
```

Appendix M

Program "prj_acdm" Example Input Files for the Operations of "Model_Struct" in Project MSP

M.1 Training

File "training.in"

```
set_mt 10
set_py_version python3.10

# read training set
cppgd {
    # read structure
    new training_set
    training_set read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/chpt3_dataset
        /database/6_mol2_conversion/train_mol2
}

# train the models and save
predco {
    new pdco_train training_set
        pdco_train set_cutoff 3 3 3
        pdco_train set_bg_train_param 256 100 5 100
        pdco_train set_ag_train_param 256 100 5 100
        pdco_train set_tg_train_param 256 100 5 100

    # model_1
    pdco_train train /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/model_struct/
        model_1
    # model_2
    pdco_train train /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/model_struct/
        model_2
```

```
    # model_3
    pdco_train train /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/model_struct/
        model_3

    pdco_train save_original_y_feature /Users/xieqin/Desktop/PhD_Thesis/
        chpt5_MSP/model_struct
}
```

M.2 Testing

File "testing.in"

```
set_mt 10
set_py_version python3.10

# read testing set and
cppgd {
    # read structure
    new testing_set
    testing_set read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/chpt3_dataset/
        database/6_mol2_conversion/test_mol2
}

# predict with trained model
predco {
    new model_1 testing_set
    model_1 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/model_1
    model_1 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/predicts

    model_1 save_original_y_feature /Users/xieqin/Desktop/PhD_Thesis/
        chpt5_MSP/model_struct/predicts
}

clear_session

cppgd {
    # read structure
    new testing_set
    testing_set read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/chpt3_dataset/
        database/6_mol2_conversion/test_mol2
}

predco {
    new model_2 testing_set
    model_2 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/model_2
```

```
    model_2 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/predicts
}

clear_session

cppgd {
    # read structure
    new testing_set
    testing_set read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/chpt3_dataset/
        database/6_mol2_conversion/test_mol2
}

predco {
    new model_3 testing_set
    model_3 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/model_3
    model_3 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/predicts
}
```

M.3 Larger Molecule Testing

File "LM_testing.in"

```
set_mt 10
set_py_version python3.10

# read testing set and
cppgd {
    # read structure
    new testing_set
    testing_set read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/chpt3_dataset/
        database/7_large_molecule_dataset/5_optimised_mol2_format
}

# predict with trained model
predco {
    new model_1 testing_set
    model_1 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/model_1
    model_1 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/LM_predicts

    model_1 save_original_y_feature /Users/xieqin/Desktop/PhD_Thesis/
        chpt5_MSP/model_struct/LM_predicts
}

```

```
clear_session

cppgd {
    # read structure
    new testing_set
    testing_set read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/chpt3_dataset/
        database/7_large_molecule_dataset/5_optimised_mol2_format
}

predco {
    new model_2 testing_set
    model_2 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/model_2
    model_2 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/LM_predicts
}

clear_session

cppgd {
    # read structure
    new testing_set
    testing_set read_mol2_dir /Users/xieqin/Desktop/PhD_Thesis/chpt3_dataset/
        database/7_large_molecule_dataset/5_optimised_mol2_format
}

predco {
    new model_3 testing_set
    model_3 load_tf_model /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/model_3
    model_3 predict 16 /Users/xieqin/Desktop/PhD_Thesis/chpt5_MSP/
        model_struct/LM_predicts
}
```

Appendix N

Python Script for Filling the Gaussian ZMatrix Template

```
import numpy as np
import os
from os.path import join
from tqdm import tqdm

default_torsion = 0.0

VERBOSE = False
# directories
ORIGINAL_ZMAT_DIR =
    "/Users/xieqin/Desktop/PhD_Thesis_meta/chpt3_dataset/database/
    8_zmat_conversion/test_zmat"
GENERATED_ZMAT_DIR =
    "/Users/xieqin/Desktop/PhD_Thesis_meta/chpt5_MSP/visualise/
    zmat_pred"
PREDICTED_DATA_DIR =
    "/Users/xieqin/Desktop/PhD_Thesis_meta/chpt5_MSP/model_struct/
    predicts"
TRACK_DATA_DIR =
    "/Users/xieqin/Desktop/PhD_Thesis_meta/chpt5_MSP/model_struct/
    track"
TRACK_AG = "ag_tracking_info.csv"
TRACK_BG = "bg_tracking_info.csv"
TRACK_TG = "tg_tracking_info.csv"

class Bond:
    def __init__(self, atom1, atom2, length=None):
        self.atom1 = atom1
        self.atom2 = atom2
        self.length = length

```

```
class Angle:
    def __init__(self, atom1, atom2, atom3, angle=None):
        self.atom1 = atom1
        self.atom2 = atom2
        self.atom3 = atom3
        self.angle = angle

class Torsion:
    def __init__(self, atom1, atom2, atom3, atom4, angle=None):
        self.atom1 = atom1
        self.atom2 = atom2
        self.atom3 = atom3
        self.atom4 = atom4
        self.angle = angle

class Variable:
    def __init__(self, name, value=None):
        self.name = name
        self.value = value

 # Load track data
track_ag = np.loadtxt(join(TRACK_DATA_DIR, TRACK_AG),
   delimiter=",", skiprows=1, dtype=str)
track_bg = np.loadtxt(join(TRACK_DATA_DIR, TRACK_BG),
   delimiter=",", skiprows=1, dtype=str)
track_tg = np.loadtxt(join(TRACK_DATA_DIR, TRACK_TG),
   delimiter=",", skiprows=1, dtype=str)

 # Read zmat file list
zmat_list = os.listdir(ORIGINAL_ZMAT_DIR)
zmat_list = [zmat for zmat in zmat_list if zmat.endswith(".gzmat")]

for model_idx in [1, 2, 3]:
    # Predicted file names
    PREDICTED_AG = f"model_{model_idx}_ag_predict.npy"
    PREDICTED_BG = f"model_{model_idx}_bg_predict.npy"
    PREDICTED_TG = f"model_{model_idx}_tg_predict.npy"
    # Load predicted data
    predicted_ag = np.load(join(PREDICTED_DATA_DIR, PREDICTED_AG))
    predicted_bg = np.load(join(PREDICTED_DATA_DIR, PREDICTED_BG))
    predicted_tg = np.load(join(PREDICTED_DATA_DIR, PREDICTED_TG))
    # Output folder
    model_out_dir = join(GENERATED_ZMAT_DIR, f"model_{model_idx}")
    os.makedirs(model_out_dir, exist_ok=True)

    for input_zmat_file in tqdm(zmat_list, desc=f"Model
       {model_idx}"):
        molecule_name = input_zmat_file.split(".")[0]
```

```
68          with open(join(ORIGINAL_ZMAT_DIR, input_zmat_file), "r")
               as f:
69              lines = f.readlines()
70
71          bonds = []
72          angles = []
73          torsions = []
74
75          ag_mask = np.char.strip(track_ag[:, 0]) == molecule_name
76          bg_mask = np.char.strip(track_bg[:, 0]) == molecule_name
77          tg_mask = np.char.strip(track_tg[:, 0]) == molecule_name
78
79          track_ag_molecule = track_ag[ag_mask]
80          track_bg_molecule = track_bg[bg_mask]
81          track_tg_molecule = track_tg[tg_mask]
82          predicted_ag_molecule = predicted_ag[ag_mask]
83          predicted_bg_molecule = predicted_bg[bg_mask]
84          predicted_tg_molecule = predicted_tg[tg_mask]
85
86          for i in range(len(predicted_ag_molecule)):
87              atom1 = int(track_ag_molecule[i][2])
88              atom2 = int(track_ag_molecule[i][3])
89              atom3 = int(track_ag_molecule[i][4])
90              degree = float(predicted_ag_molecule[i])
91              angle = Angle(atom1, atom2, atom3, degree)
92              angles.append(angle)
93          for i in range(len(predicted_bg_molecule)):
94              atom1 = int(track_bg_molecule[i][2])
95              atom2 = int(track_bg_molecule[i][3])
96              length = float(predicted_bg_molecule[i])
97              bond = Bond(atom1, atom2, length)
98              bonds.append(bond)
99          for i in range(len(predicted_tg_molecule)):
100             atom1 = int(track_tg_molecule[i][2])
101             atom2 = int(track_tg_molecule[i][3])
102             atom3 = int(track_tg_molecule[i][4])
103             atom4 = int(track_tg_molecule[i][5])
104             degree = float(predicted_tg_molecule[i])
105             torsion = Torsion(atom1, atom2, atom3, atom4, degree)
106             torsions.append(torsion)
107
108         output_lines = []
109         variable_list = []
110         start_mark = False
111         atom_id = 0
112         error_skip = False
113         for line in lines:
114             if line == "Variables:\n":
```

```
                    break
                if line == "0  1\n":
                    start_mark = True
                    output_lines.append(line)
                    continue
                if not start_mark:
                    output_lines.append(line)
                    continue

                output_lines.append(line)
                fragments = line.split()
                frag_count = 0
                bond_atom = None
                bond_variable_name = None
                angle_atom = None
                angle_variable_name = None
                torsion_atom = None
                torsion_variable_name = None
                for frag in fragments:
                    if frag_count == 0:
                        frag_count += 1
                        continue
                    elif frag_count == 1:
                        bond_atom = int(frag) - 1
                    elif frag_count == 2:
                        bond_variable_name = frag
                    elif frag_count == 3:
                        angle_atom = int(frag) - 1
                    elif frag_count == 4:
                        angle_variable_name = frag
                    elif frag_count == 5:
                        torsion_atom = int(frag) - 1
                    elif frag_count == 6:
                        torsion_variable_name = frag
                    frag_count += 1
                if bond_atom is not None:
                    length = None
                    for bond in bonds:
                        if (bond.atom1 == atom_id and bond.atom2 ==
                            bond_atom) or (bond.atom1 == bond_atom and
                            bond.atom2 == atom_id):
                            length = bond.length
                            break
                    if bond_variable_name is not None and length is
                        not None:
                        variable_list.append(Variable(bond_variable_name,
                            length))
                    else:
```

```
159                     if VERBOSE:
160                         print(f"Failed to find bond variable
                                {bond_variable_name} or length for
                                atom_id {atom_id} and bond_atom
                                {bond_atom} in molecule
                                {molecule_name}")
161                     error_skip = True
162                 if angle_atom is not None:
163                     degree = None
164                     for angle in angles:
165                         if (bond_atom == angle.atom2) and
                            ((angle.atom1 == atom_id and angle.atom3 ==
                            angle_atom) or (angle.atom1 == angle_atom
                            and angle.atom3 == atom_id)):
166                             degree = angle.angle
167                             break
168                     if angle_variable_name is not None and degree is
                        not None:
169                         variable_list.append(Variable(angle_variable_name,
                            degree))
170                     else:
171                         if VERBOSE:
172                             print(f"Failed to find angle variable
                                {angle_variable_name} or angle for
                                atom_id {atom_id}, bond_atom
                                {bond_atom} and angle_atom {angle_atom}
                                in molecule {molecule_name}")
173                         error_skip = True
174                 if torsion_atom is not None:
175                     degree = None
176                     for torsion in torsions:
177                         if (
178                             ((torsion.atom1 == atom_id and
                                torsion.atom4 == torsion_atom) or
                                (torsion.atom1 == torsion_atom and
                                torsion.atom4 == atom_id)) and
179                             ((torsion.atom2 == bond_atom and
                                torsion.atom3 == angle_atom) or
                                (torsion.atom2 == angle_atom and
                                torsion.atom3 == bond_atom))
180                         ):
181                             degree = torsion.angle
182                             break
183                     if torsion_variable_name is not None and degree is
                        not None:
184                         variable_list.append(Variable(torsion_variable_name,
                            degree))
185                     else:
```

```
186                     if VERBOSE:
187                         print(f"Failed to find torsion variable
                               {torsion_variable_name} or angle for
                               atom_id {atom_id}, bond_atom
                               {bond_atom}, angle_atom {angle_atom}
                               and torsion_atom {torsion_atom} in
                               molecule {molecule_name}. Defaulting to
                               {default_torsion}")
188                     variable_list.append(Variable(torsion_variable_name,
                           default_torsion))
189                 atom_id += 1
190
191             if error_skip:
192                 if VERBOSE:
193                     print(f"Skipping molecule {molecule_name} due to
                           error in variable extraction")  # Skip
                           molecules with errors
194                 continue
195
196             output_lines.append("Variables:\n")
197             for variable in variable_list:
198                 output_lines.append(f"{variable.name} =
                       {variable.value}\n")
199             with open(join(model_out_dir, f"{molecule_name}.gzmat"),
                   "w") as fout:
200                 fout.writelines(output_lines)  # Write output file
```

Appendix O

Python Script for RMSD Calculation in Project MSP

```python
from rdkit import Chem
from rdkit.Chem import rdMolAlign
from os.path import join
import os
from tqdm import tqdm

CURRENT_DIR = "/Users/xieqin/Desktop/PhD_Thesis_meta/chpt5_MSP/visualise/"

TRUE_STRUCT_DIR = "/Users/xieqin/Desktop/PhD_Thesis_meta/chpt3_dataset/database/3_modified_xyz_structure/testing_set"
PRED_STRUCT_DIR = "/Users/xieqin/Desktop/PhD_Thesis_meta/chpt5_MSP/visualise/xyz_pred"

OUTPUT_CSV_FILE = join(CURRENT_DIR, "rmsd_results.csv")

import concurrent.futures

# list dir from PRED_STRUCT_DIR
files_names = os.listdir(PRED_STRUCT_DIR)
files_names = [f for f in files_names if f.endswith('.xyz')]

def compute_rmsd(file):
    true_file = join(TRUE_STRUCT_DIR, file)
    pred_file = join(PRED_STRUCT_DIR, file)
    molecule_name = file.split('.')[0]
    true_mol = Chem.MolFromXYZFile(true_file)
    pred_mol = Chem.MolFromXYZFile(pred_file)
    rmsd = rdMolAlign.GetBestRMS(true_mol, pred_mol)
```

```
    return molecule_name, rmsd

results = []
with concurrent.futures.ThreadPoolExecutor() as executor:
    for molecule_name, rmsd in tqdm(executor.map(compute_rmsd,
        files_names), total=len(files_names)):
        results.append((molecule_name, rmsd))

with open(OUTPUT_CSV_FILE, 'w') as f:
    f.write("molecule,RMSD\n")
    for molecule_name, rmsd in results:
        f.write(f"{molecule_name},{rmsd}\n")
```

Appendix P

Coordinate of Visualised Molecules in XYZ format

P.1 Molecule `cno_15016`

DFT Reference Structure

12

O	1.716939	1.415824	0.000273
C	0.689006	0.766341	0.000020
N	−0.565641	1.354595	−0.000352
C	−1.779105	0.671352	0.000000
C	−1.779101	−0.671357	0.000134
N	−0.565634	−1.354597	0.000042
C	0.689010	−0.766338	−0.000026
O	1.716940	−1.415820	−0.000119
H	−0.554636	2.367899	−0.000316
H	−2.680856	1.267370	0.000222
H	−2.680849	−1.267378	0.000619
H	−0.554624	−2.367899	−0.000367

ML-predicted Structure

12

O	1.747150	1.408433	0.001249
C	0.710340	0.759739	0.000924
N	−0.534728	1.394294	0.001383
C	−1.743608	0.727768	0.001057
C	−1.772219	−0.653113	0.000229

N	−0.591980	−1.369138	−0.000275
C	0.678298	−0.786699	−0.000004
O	1.687346	−1.477787	−0.000482
H	−0.688048	2.390551	−0.004409
H	−2.678393	1.264151	0.001438
H	−2.646095	−1.283889	−0.000096
H	−0.516614	−2.374317	−0.000884

P.2 Molecule `cno_16580`

DFT Reference Structure

13

O	2.694904	0.188385	0.112074
N	1.667382	−0.734021	−0.131307
C	0.514771	−0.171654	−0.106733
C	0.120268	1.233097	0.092337
C	−1.210473	1.361688	−0.073108
C	−1.837326	0.024879	−0.339580
O	−1.670643	−0.907835	0.756339
C	−0.740643	−0.959680	−0.344518
H	3.487302	−0.369139	0.064711
H	0.821775	2.014256	0.355283
H	−1.774493	2.284204	0.012738
H	−2.754673	−0.084731	−0.911879
H	−0.725260	−1.880818	−0.919401

ML-predicted Structure

13

O	2.903781	−0.662002	0.124853
N	2.166378	0.431400	0.621736
C	0.969912	0.437048	0.152711
C	0.023554	1.507091	0.550126
C	−1.156157	1.313240	−0.056134
C	−1.042575	0.064862	−0.911437
O	−0.174261	−0.925436	−0.333449
C	−1.591967	−1.069975	−0.133098
H	2.254490	−1.100710	−0.448241
H	0.095215	2.528216	0.182166
H	−1.971209	2.008396	0.122867
H	−1.834967	−0.634657	−0.628582

H	−2.049303	−1.898842	−0.676564

P.3 Molecule cno_16399

DFT Reference Structure

14

O	−2.166632	−1.068876	0.040657
C	−1.531523	0.125191	0.012424
O	−2.131590	1.176922	0.026394
C	−0.052971	0.048021	−0.023292
N	0.546775	−1.096087	−0.029469
N	1.896690	−0.900377	−0.164144
C	2.230410	0.504967	0.175527
C	0.894818	1.220386	−0.120028
H	−1.486385	−1.770189	0.010176
H	2.427022	−1.634989	0.291470
H	3.064210	0.856795	−0.436178
H	2.505292	0.598008	1.236618
H	0.870971	1.646446	−1.131656
H	0.656003	2.023422	0.580668

ML-predicted Structure

14

O	−2.415290	−0.521283	0.018959
C	−1.620461	0.258559	0.783775
O	−0.424635	0.321914	0.632021
C	−0.169047	0.372819	0.643431
N	0.461453	−0.303581	−0.269495
N	1.744169	0.195737	−0.393238
C	1.977894	1.195649	0.648705
C	0.583137	−0.401484	−0.407386
H	−1.848701	−0.977732	−0.623783
H	2.375579	−0.572210	−0.598998
H	2.783789	0.847299	1.305127
H	2.783789	0.847299	1.305127
H	0.745707	0.233327	−1.287539
H	0.745707	0.233327	−1.287539

P.4 Molecule cno_16261

DFT Reference Structure

19

O	2.756632	−0.077020	−0.998757
C	2.518499	0.019022	0.193637
N	1.277623	0.066597	0.754243
C	0.040220	0.006449	−0.006964
C	−0.885187	1.223143	0.194659
C	−2.270431	0.748579	−0.306284
C	−2.221179	−0.813019	−0.296534
C	−0.869930	−1.184919	0.344771
H	3.318035	0.079356	0.959237
H	1.217417	0.145900	1.761529
H	0.347671	−0.060874	−1.054628
H	−0.517256	2.115952	−0.320534
H	−0.927474	1.463446	1.266527
H	−2.464815	1.118027	−1.318648
H	−3.075247	1.135596	0.326801
H	−2.264891	−1.196707	−1.322096
H	−3.066863	−1.254126	0.240379
H	−0.466991	−2.140307	−0.006137
H	−0.967956	−1.251806	1.438213

ML-predicted Structure

19

O	3.509032	0.559295	−0.412253
C	2.328090	0.520990	−0.103408
N	1.720578	−0.530411	0.526178
C	0.291920	−0.547430	0.886429
C	−0.483964	0.718645	0.500219
C	−1.456491	0.224496	−0.585545
C	−1.741562	−1.230727	−0.175381
C	−0.483964	0.718645	0.500219
H	1.874964	−0.348275	0.404988
H	2.303548	−1.175014	1.045160
H	−0.235361	0.369010	0.598402
H	−1.449403	0.495584	0.041512
H	−1.449403	0.495584	0.041512
H	−2.354033	0.852450	−0.615024

H	−2.354033	0.852450	−0.615024
H	−2.821616	−1.411585	−0.135798
H	−2.821616	−1.411585	−0.135798
H	−1.449403	0.495584	0.041512
H	−1.449403	0.495584	0.041512

P.5 Molecule `cno_15931`

DFT Reference Structure

19

O	4.105229	−0.313717	−0.000157
C	2.951105	0.519971	−0.000044
C	1.724220	−0.383130	0.000040
C	0.409391	0.405996	−0.000020
C	−0.807265	−0.509252	0.000295
O	−1.969054	0.329455	0.000287
N	−3.117804	−0.463570	−0.000215
C	−4.172996	0.251375	−0.000180
H	4.886920	0.258879	−0.000145
H	2.932218	1.174123	−0.887931
H	2.932371	1.174091	0.887870
H	1.777567	−1.036557	0.880491
H	1.777565	−1.036683	−0.880317
H	0.358615	1.059382	−0.880971
H	0.358773	1.059776	0.880653
H	−0.819836	−1.154211	0.888716
H	−0.820093	−1.154551	−0.887863
H	−5.121407	−0.277220	−0.000789
H	−4.154201	1.342302	0.000196

ML-predicted Structure

19

O	1.360867	−0.728874	0.000353
C	1.507688	0.690010	−0.000318
C	0.138853	1.380447	−0.000624
C	−1.024036	0.372527	−0.000132
C	−0.536522	−1.079433	0.000546
O	0.897527	−1.154213	0.000560
N	1.433666	0.140018	−0.000058
C	0.503243	1.017222	−0.000458

H	0.408112	−0.918185	0.000456
H	0.524224	1.177725	−0.000534
H	0.524224	1.177725	−0.000534
H	−0.655898	0.625348	−0.000256
H	−0.655898	0.625348	−0.000256
H	−0.616813	−0.645283	0.000342
H	−0.616813	−0.645283	0.000342
H	0.559614	−1.104840	0.000542
H	0.559614	−1.104840	0.000542
H	−0.540167	0.710519	−0.000298
H	−0.540167	0.710519	−0.000298