



DISSERTATION | DOCTORAL THESIS

Titel | Title

Machine learning for chemical data

verfasst von | submitted by

Beryl Ramadhian Aribowo

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of
Doktor der Naturwissenschaften (Dr.rer.nat.)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree
programme code as it appears on the
student record sheet:

UA 796 605 405

Dissertationsgebiet lt. Studienblatt | Field of
study as it appears on the student record
sheet:

Mathematik

Betreut von | Supervisor:

emer. o. Univ.-Prof. Dr. Arnold Neumaier

Abstract

The most important chemical property of molecules is the equilibrium energy and that of their molecular configurations. This thesis deals with the prediction of these energies from given data using statistical and machine learning approaches.

The first part of the thesis deals with the construction of highly accurate potential energy surfaces for diatomic molecules using traditional functional approaches.

The second part deals with more complex potential energy functions for small molecules H_xO_y and their training with reverse mode automatic differentiation, which offers better efficiency compared to finite differences or symbolic differentiation. In addition, a novel semi-supervised learning technique based on Shepard interpolation is presented, which is useful for training chemical data with minimal information on the target.

The third part of the thesis discusses the QM9 challenge, the problem of predicting all 130K equilibrium energies of the QM9 dataset with chemical accuracy from the equilibrium energies for a tiny amount (less than 1%) of training geometries. By combining delta machine learning, hyperparameter tuning, and data selection techniques involving combinatorial optimization, an accuracy of 3.25 kcal/mol is achieved with 100 training geometries. Using the training parameters found and a less expensive data selection strategy, chemical accuracy (1 kcal/mol) was achieved with 3750 training geometries.

Zusammenfassung

Die wichtigste chemische Eigenschaft von Molekülen ist die Energie im Gleichgewicht und in ihren molekularen Konfigurationen. Diese Arbeit befasst sich mit der Vorhersage dieser Energien aus gegebenen Daten unter Verwendung statistischer und maschineller Lernansätze.

Der erste Teil der Arbeit befasst sich mit der Konstruktion hochgenauer potenzieller Energieoberflächen von zweiatomigen Molekülen unter Verwendung traditioneller funktionaler Ansätze.

Der zweite Teil befasst sich mit komplexeren potenziellen Energiefunktionen für kleine Moleküle H_xO_y und deren Training mit automatischer Differenzierung im umgekehrten Modus, die im Vergleich zu finiten Differenzen oder symbolischer Differenzierung eine bessere Effizienz bietet. Darüber hinaus wird eine neuartige halbüberwachte Lerntechnik auf der Grundlage der Shepard-Interpolation vorgestellt, die für das Training chemischer Daten mit minimalen Informationen über das Ziel nützlich ist.

Der dritte Teil der Arbeit behandelt die QM9-Herausforderung, das Problem der Vorhersage aller 130K Gleichgewichts-Energien des QM9-Datensatzes mit chemischer Genauigkeit aus den Gleichgewichts-Energien für eine winzige Menge (weniger als 1%) von Trainingsgeometrien. Durch die Kombination von maschinellem Delta-Lernen, Hyperparameter-Abstimmung und Datenauswahltechniken, die kombinatorische Optimierung beinhalten, wird mit 100 Trainingsgeometrien eine Genauigkeit von 3.25 kcal/mol erreicht. Mit den gefundenen Parametern und einer weniger aufwendigen Datenauswahlstrategie wurde chemische Genauigkeit (1 kcal/mol) mit 3750 Trainingsgeometrien erreicht.

Acknowledgement

First and foremost, I am deeply grateful to Allah the Almighty for the greatest blessing, that is the blessing of life, so that I can exist in this world, and experience the journey of PhD study.

I would like to express my deep gratitude and appreciation to Prof. Arnold Neumaier as my supervisor during my PhD study, for his patience, encouragement, and guidance. I learned a great deal from him, not only in technicalities and sciences, but more importantly, invaluable insights and wisdom that incite inspiration.

I would like to acknowledge the Vienna Graduate School of Computational Optimization (VGSCO) for the resources, support, and funding, under the FWF-Project No. W1260-N35. I would like to thank Prof. Radu Ioan Boț as an exemplary leader of the VGSCO and for his generosity. I would like to also thank Bettina Hiebl as the project coordinator of the VGSCO, who have helped me many times with administration tasks.

I would like to thank Prof. O. Anatole von Lilienfeld for his hospitality and for the fruitful discussions during my research stay in the University of Toronto and the last stages of my PhD study. I am grateful of Prof. Kuntjoro Adji Sidarto and Prof. Hidemi Nagao, my supervisors during master's study, who encouraged and supported me to pursue PhD study.

Thanks to my colleagues, friends, and acquaintances in Vienna, Bandung, Toronto, and many other places; for making life during PhD study more vibrant, giving me some pieces of shards of life outside of research and study.

My deepest gratitude is for my family. For my mother, father, and sister, who have given me continuous support, love, and understanding even when I am seas and continents away.

All in all, my sincere appreciation for all that have contributed directly or indirectly to the writing of this thesis.

Contents

Abstract	i
Zusammenfassung	ii
Acknowledgement	iii
1 Introduction	1
1.1 Overview	1
1.2 Atoms and molecules	3
1.3 Spectroscopic notation	3
1.4 Potential energy surface	4
1.5 The Born-Oppenheimer approximation	4
1.6 Quality assessment	5
1.7 Unit conversion factors	5
2 Machine learning tools for chemical data	6
2.1 The features of chemical data	6
2.1.1 Dressed atoms, bonds, angles, and torsion (DA, DB, DN, DT)	6
2.1.2 SMILES string	7
2.1.2.1 General molecular subgraph and bond extraction . .	9
2.1.2.2 Extracting angles & torsions	10
2.1.3 Higher resolution features	11
2.1.3.1 Many Body Density Functional (MBDF)	12
2.1.3.2 Atom-centered Symmetry Functions (ACSF)	12
2.1.3.3 Smooth Overlap of Atomic Positions (SOAP)	12
2.1.3.4 Faber–Christensen–Huang–Lilienfeld (FCHL)	13
2.1.4 Feature selection	13
2.1.4.1 Atomic feature selection	13
2.1.4.2 Molecular feature selection	16
2.1.5 Alternative way to scale the features	19
2.1.6 Feature transformation by message passing	19
2.1.7 Completeness of molecular features	20
2.2 Models for general chemical data fitting	22
2.2.1 Standard linear model	22
2.2.2 Kernel models	23
2.2.3 Neural network (NN)	24
2.3 Hyperparameter optimization	26
2.3.1 Handling special-typed variables	28
2.3.1.1 Treating categorical variables	28
2.3.1.2 Treating ordered integer variables	29
3 Functional forms of potential energy surfaces	31
3.1 Permutational-invariant PES (BRAAMS & BOWMAN 2009)	31
3.2 CHIPR (ROCHA & VARANDAS 2021)	32
3.2.1 General representation	32
3.2.2 One-body term	33
3.2.3 Two-body terms	33
3.2.4 Three-body terms	34
3.2.5 Four-body terms	34

3.3	CHIPR for HO_2^+ (XAVIER et al 2019)	35
3.4	CHIPR fitting	36
3.5	2 x 2 diabatic matrix for HO_2^+ dissociation (XAVIER & VARANDAS 2021)	37
3.6	3 x 3 diabatic matrix for H_2O_2 dissociation (COELHO & BRANDÃO 2017)	38
3.7	EHF and long range term for $\text{H}_2\text{O}(^1A_1)$ (BRANDÃO & RIO 2003)	40
3.7.1	Elements of the matrix	40
3.7.2	Diatomic terms	40
3.7.3	Triatomic terms	41
3.8	V_R potentials for HO_2 (BRANDÃO et al 2009)	43
3.9	MBE/DC PES for H_2O (GALVÃO & RODRIGUES 2008)	44
3.10	Neural network for H_3O (CHEN et al 2013)	44
3.11	Lennard-Jones-like pair potential (DEITERS & NEUMAIER 2016)	44
3.12	Pair potential for noble gases (DEITERS & SADUS 2019)	45
4	Datasets and proposed functions of pair potentials containing H and O atoms	47
4.1	H_2 data	47
4.1.1	H_2^+ data	47
4.2	O_2 data	48
4.2.1	O_2^+ data	48
4.3	OH data	49
4.3.1	OH^+ data	51
4.3.2	OH^- data	51
4.4	Proposed pair potential ansatz	52
4.4.1	Polynomial fit	52
4.4.2	RATPOT1	53
4.4.3	RATPOT2	54
4.4.4	RATPOT3	54
4.4.5	RATPOTL	54
4.5	Fitting method	56
4.5.1	Fitting for general pair potential ansatz	56
4.5.2	Fitting dissociated energy	56
4.6	Pair potentials benchmark	57
5	Potential energy surface with bonding features	60
5.1	Bonding features	60
5.1.1	Bonding function	60
5.1.2	Reference energy	61
5.1.3	Coordination vector	62
5.1.4	Orientation vector	62
5.1.4.1	Converting distances to coordinates	62
5.1.5	Gram matrix	64
5.1.6	Neighbourhood matrix	64
5.2	Models	64
5.2.1	Model with trainable reference pair potential	65
5.2.2	Model with fixed diatomic partial model	65
5.3	Objective function	65
5.4	Extension: primitive bonding features	66

5.4.1	Coordination function	66
5.4.2	Bump functions	67
5.4.3	Bump functions for linear system	68
5.5	Notes on implementation	70
5.6	Numerical experiments	73
5.6.1	Feature statistics	73
5.6.2	Fitting H ₃ , H ₄ , and H ₅ molecules	74
6	Semi supervised model based on Shepard interpolation	76
6.1	Robust Shepard interpolation	76
6.1.1	Generalized Shepard model	76
6.1.2	Interpolating equilibrium geometries	77
6.1.3	Interpolation properties	78
6.2	Computational details	79
6.2.1	Linear model form	79
6.2.2	B-spline basis function	79
6.2.3	Precomputed values	81
6.2.4	Fitting the model	82
6.3	Numerical experiments on pair potential H _x O _y dataset	83
6.4	Numerical experiments on QM9 dataset	84
6.4.1	Memory usage	84
6.4.2	Hyperparameter space	85
6.4.3	Fitting results	87
6.4.3.1	Null model and reduced energies	87
6.4.3.2	Choice of training sets	87
6.4.3.3	Fitting reduced energies	88
6.4.3.4	Hyperparameter optimization (Hyperopt)	89
7	Delta machine learning (ΔML) and optimal selection of molecules	94
7.1	Delta machine learning (Δ ML)	94
7.2	Data selection technique	95
7.2.1	Unsupervised selection technique	95
7.2.2	Supervised selection technique	95
7.3	Numerical experiments	97
7.3.1	Setup	97
7.3.1.1	Dataset	97
7.3.1.2	Hyperparameter space	97
7.3.1.3	Training set selection schemes	98
7.3.2	Preliminary experiments of baselines, training-set selection, and features	99
7.3.3	Baselines with feature selection	102
7.3.4	Dressed atom hybridization	103
7.3.5	Benchmark: baseline learning curves	104
7.3.6	Benchmark: learning curves of higher resolution features . . .	105
7.3.7	Hyperparameter tuning intensification for <code>ntrain</code> = 100 . .	108
7.3.8	Optimal training set selection	110
7.3.9	Optimal level of delta learning	113
7.4	Explainable AI (X-AI): principal components of kernel feature	113

8	High-performance code in Julia	117
8.1	Minimizing memory allocation	117
8.2	Multi-threading	119
8.3	Automatic differentiation (AD)	120
8.3.1	Background and simple example	120
8.3.2	Implementation example in Julia	122
	Conclusion	124
	Appendices	125
A	General implementation information	125
B	Semi supervised model (Chapter 6)	125
B.1	Program structure for experiments in Section 6.4	125
B.1.1	Feature extraction	125
B.1.2	Data setup subprogram	126
B.1.3	Fitter subprogram	126
B.2	Additional results for experiments in Section 6.3	128
B.3	Additional results for experiments in Section B.1	128
C	ΔML (Chapter 7)	130
C.1	Additional results involving baseline features	130
C.2	Hyperparameter tuning detailed results (Section 7.3.7)	136

1 Introduction

1.1 Overview

Chemical data contains useful properties (quantities) for various applications. One such property is the energy of the chemical system that can be determined given the structure of the system. Traditionally, the energy can be predicted by *ab initio* methods accurately. However, *ab initio* methods have a massive drawback with regards to the computational cost, such that it is intractable for large chemical systems [1]. On the other side of the spectrum, force field methods exist, cheap but less accurate and often system specific [2].

With this regard, machine learning brings the promise of the best of both worlds. In hindsight, the advancement of machine learning for the predictions of energies can be categorized into two major approaches:

(i). Designing a set of atomic or molecular **features** (or also often referred as atomic or molecular representation), i.e., chemical data feature engineering. Chemical data feature engineering mostly requires specific domain knowledge in physics and chemistry to explain the interaction of atoms rather than generic modelling abstraction, although arguably some components can be promoted into generic abstractions to some degree. Several ready to use prominent atomic features as software libraries including but not limited to [3, 4, 5, 6]. These features incorporate symmetry such that they are invariant when the molecular coordinates are translated, rotated, or when atoms with equal type are permuted [7].

(ii). Designing the fitting model; in contrast to the features, this is more of a generic approach. Well known fitting models for chemical quantity regressions ranging from linear models to the neural networks [8, 9, 10, 11]. Often, specific domain knowledge of the problem to be solved is not required to design such models. Usually the techniques are abstracted in more generic forms that are common in mathematics and computer science disciplines. There can also be a case where a fitting model were devised from the necessity of solving domain specific problem, one of the most recent well known model of this case is the message passing neural network (MPNN) [12]. For energy prediction, the models for fitting can be further loosely categorized as global potential energy surfaces for smaller molecules and general fitting model for large equilibrium geometries.

Global potential energy surface captures the whole energy curve from the short range interatomic interactions up to the long range interactions. From the perspective of pair potential (potential curve of molecules containing two atoms (diatomic) only), the functional forms for global potential energy surface mimic the shape of curve produced by the Lennard-Jones (LJ) potential [13], with the mix of tuning parameters to account for data points flexibility, such as in [14, 15]. For global potentials of smaller molecules larger than diatomics, additionally it is common to construct the functional forms using n -body sum, it is a sum of nonlinear functions where the functions are ordered based on the dependency of the number of atoms' interactions. One such functional form is CHIPR [16].

Models for larger equilibrium molecules are commonly in the form of general machine learning models, such as employed in [11, 17, 6]. The models take in pre-computed atomic features that encode the invariants properties. The models and the features are often disjoint, giving opportunities to mix and match the construction of the energy prediction pipeline, in fact, this can be wrapped as a hyperparameter optimization problem. Some models have more interconnected relationship between the feature and the fitting model, e.g., [10] devised the symmetric atomic feature

alongside a neural network that is symmetric with regard to the atomic energy, and [18] incorporated symmetries in the structure of the network without the need of any prior feature extraction (analogous to deep neural networks for general problems, they do not need intensively engineered features as input).

All of the aforementioned techniques provide a zoo of feature-model combinations that can be utilized and tailored according to problem’s requirement. Something to note is that the term "model" here can also refer to the combined feature and fitting model themselves, depending on the context; as it is also a common practice in mathematical modelling to distinguish the ansatz and formulas which is aggregated as the "model," from the algorithms or ways to find the solutions which is often dubbed as the "solver".

The quality of machine learning techniques are practically judged by the prediction errors. In expectation, the prediction errors decay as the number of data used for training is larger. While machine learning techniques themselves are known to be fast, the accurate data from *ab initio* techniques that are used for the training process do not come for cheap. This suggests that machine learning methods with efficient data usage for training are desired. Furthermore, aside from the obvious speed up in computation due to lower amount of data to be processed for training process, accurate data efficient techniques would also imply better generalization, model transferability would not be out of the realm of possibility.

A machine learning for quantum chemistry challenge, dubbed the **QM9 challenge**, was proposed by [19] and then reiterated in [20, 21, 2]. The challenge uses the QM9 dataset [22] for the benchmark. The QM9 dataset contains the geometries of 134k equilibrium molecules containing up to 9 heavy atoms (C, N, O, F) excluding H. This challenge addresses the goal of reaching chemical accuracy: 1 kcal/mol, with only 100 molecules for training, i.e., less than 0.1% of the total number of molecules in the QM9 dataset (and the rest of the 130k molecules are for testing). This challenge is also set as a motivation for this thesis.

We started by the benchmark of global potential functions on high accuracy datasets of small molecules, specifically pair potential functions for molecules containing Hydrogen and Oxygen atoms, and Hydrogen molecules of size 3 up to 6. For pair potential functions benchmark in Chapter 4, we developed rational potential (RATPOT) functions. Our method reached chemical accuracy (1 kcal/mol) for most of the datasets, and is more efficient in computation time when compared to CHIPR. For the global potential benchmark on Hydrogen clusters in Chapter 5, we developed atomic features that mimics the bonding strength of atoms, and continuous model that combines the features which is compatible for automatic differentiation. The model reached chemical accuracy for H₄ dataset, and is more efficient computationally due to reverse mode automatic differentiation in comparison to finite differences.

For the benchmark of large equilibrium molecules, we used the QM9 dataset with the constraints of QM9 challenge. First we developed a semi supervised model based on Shepard interpolation in Chapter 6. The model can make use of extra unlabelled data, hence does not violate the QM9 challenge constraints even if the total molecules used are over 100. Chemical accuracy was not yet reached within the constraints of the QM9 challenge. The model was also tested for the pair potential datasets and reached chemical accuracy for some of the datasets, despite was not specifically designed for global pair potential datasets. On the next attempts of QM9 challenge in Chapter 7, we combined delta machine learning, data selection techniques, and hyperparameter tuning. For only 100 training molecules (the QM9

challenge constraints), our method is at 3.25 kcal/mol; the method reached chemical accuracy with 3750 training molecules using lower resolution hyperparameter tuning.

1.2 Atoms and molecules

Throughout the discussion of the functional forms of potential energy surfaces, several relevant concepts pertaining to the description of atoms are:

- The **nuclear geometry** defines that molecules consist of nuclei at the center of the atom, which the nuclei itself is surrounded by the **electron cloud**. The electron cloud describes the region which has a high probability of electrons to reside.
- The **atom type** a is an element from the periodic table of the elements, $a \in \{\text{H, He, Li, Be, ..., O, ...}\}$.
- The **nuclear charge** Z_i of the i th atom is equal to the number of the protons within atom i , which is the atomic number within the periodic table of the elements. The example of the nuclear charges is shown in Table 1.
- The **position** $\mathbf{x}_i$ of the atom i th is the location of the nucleus of atom i within the cartesian coordinates of three-dimensional space.
- The **interatomic distance** r_{ij} is the Euclidean distance between atom i and j ,

$$r_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2 \quad (1)$$

where $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2}$. Occasionally, the notation R_k ($k = 1, 2, 3...$) is preferred, it denotes the k th interatomic distances, this is relevant when the indices of the formula requires the enumeration of the distances in correlation to the data points, or when the indices are more convenient to enumerate in conjunction to other factors or terms which do not directly correlate to atomic combinations. Other possibilities would be writing d_{ij} or d_k as the distance between i th and j th atom and k th distance respectively.

Table 1: Nuclear charges (or the atomic numbers) of nine atom types.

a	H	He	Li	Be	B	C	N	O	F	...
Z	1	2	3	4	5	6	7	8	9	...

1.3 Spectroscopic notation

The following description of the spectroscopic notation is obtained from [23]. The molecular spectroscopic notation $I(T)$ describes the state T of molecule I . T is composed of three parts: a letter, a spectroscopic symbol, and molecular orbital designation. The ground state is labeled X and excited states with the same multiplicity as the ground state are labelled in increasing order of energy with capital letters: $A, B, C, ...$; excited states with different multiplicity are labelled with lower-case letters: $a, b, c, ...$. The spectroscopic symbol which shows the spin multiplicity $2S + 1$ is represented by the superscripted number. The quantum number that represents angular momentum is Λ ; the Greek letters $\Sigma, \Pi, \Delta...$ (follows $s, p, d...$) signify $\Lambda = 0, 1, 2, ...$ respectively. The subscript g or u , indicates even or odd symmetry

respectively under inversion of wavefunction through the center of the molecule. One denotes the even or odd symmetry under the reflection of wavefunction at any plane passing through both nuclei by the superscripted + or − respectively. For example, $I(T) = \text{H}_2(X^1\Sigma_g^+)$ is a ground state hydrogen with $2S + 1 = 1$, $\Lambda = 1$, even symmetry under inversion of wavefunction, and even symmetry under the reflection of the wavefunction.

For convenience, consider the proposed rules by [24] to determine the possible symbols for the diatomic molecular states given a pair of atomic states. For example, two atoms in identical ^3S states can form a diatomic molecule with $^1\Sigma_g^+$, $^3\Sigma_u^+$, or $^5\Sigma_g^+$ states; ^1S_g and ^1P_u possibly will result in $^1\Sigma_g^+$, $^1\Sigma_u^+$, $^1\Pi_g$, or $^1\Pi_u$. The symmetry is g or u if the sum of the angular momentum is even or odd respectively. The simplified correlation rules are shown in Table 2.

Table 2: Possible diatomic terms resulted from atomic terms.

Atomic terms	Diatomic terms
$S_g + S_g$ or $S_u + S_u$	Σ^+
$S_g + S_u$	Σ^-
$S_g + P_g$ or $S_u + P_u$	Σ^-, Π
$S_g + P_u$ or $S_u + P_g$	Σ^+, Π
$P_g + P_g$ or $P_u + P_u$	$\Sigma^+(2), \Sigma^-, \Pi(2), \Delta$
$P_g + P_u$	$\Sigma^+, \Sigma^-(2), \Pi(2), \Delta$

1.4 Potential energy surface

The **potential energy surface** (PES) describes the potential energy of a set of nuclear geometries of a particular molecule. By changing the nuclear geometry in a small amount in a desirable direction and evaluating it through the potential energy function, the PES is formed. In the simplest form, the PES is a system composed of a function f which computes the value of energy V relative to the intermolecular distances $\mathbf{R}$:

$$V := f(\mathbf{R}). \quad (2)$$

1.5 The Born-Oppenheimer approximation

The *ab initio* methods aim to find the solution of the non-relativistic solution of the electronic Schrödinger equation. Mainly this is done by the **Born-Oppenheimer (BO) approximation** [25]. The motivation of BO approximation is to alleviate the difficulty of computing the energy and wave function of molecules, for example the benzene molecule C_6H_6 consists of 12 nuclei and 42 electrons, in a three-dimensional coordinates, results in a Schrodinger equation which is a partial differential eigenvalue equation with 162 variables ((3×12) nuclear + (3×42) electronic). The BO approximation is used to separate the (quantum mechanical) motion of the electrons from the motion of the nucleus, due to the fact that the ratio of the mass of electrons and nucleus are very large, which implies that given the same amount of kinetic energy, the nuclei move much more slowly than the electrons.

The main interest lies in the first step of the BO approximation, which is solving the electronic Schrödinger equation, yielding a wave function depending on electrons, formally

$$H_{\text{elec}}\chi(\mathbf{R}) = E_{\text{elec}}\chi(\mathbf{R}), \quad (3)$$

where H_{elec} is the electronic **Hamiltonian** which is the sum of electronic {kinetic energies, electronic repulsions, internuclear repulsions, electron–nuclear attractions}; $\chi(\mathbf{R})$ is the eigenfunction; $\mathbf{R}$ is the list of electronic coordinates; the eigenvalue E_{elec} is the PES. By varying $\mathbf{R}$, i.e., changing the nuclear geometry in small steps one can measure different values of the PES, this is referred as the **adiabatic approximation**.

On the other hand, **diabatic approximation** is used when the PES is multi-sheeted. The diabatic approach approximates the multi-valued energies through eigenvalues of a matrix (where each element is formed by polynomial functions), where higher eigenvalue represents higher level of energy (e.g., the energy of an excited state molecule). The lowest eigenvalue represents the energy of the ground state.

1.6 Quality assessment

To measure the performance of the PES model, the root mean squared error (RMSE) is used. The formal definition of RMSE pertaining to the PES is

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (V_{i,\text{fit}} - V_{i,\text{ab initio}})^2}, \quad (4)$$

where n is the number of the data points (of the whole dataset),

$$V_{i,\text{fit}} := f(R_i) \quad (5)$$

is the energy value from the fitted function f , with the i th molecular distance from the dataset as the input parameter; $V_{i,\text{ab initio}}$ is the corresponding i th *ab initio* energy value. In a fitting scenario, the RMSE is used for both in training and validation scheme, where within the training scheme, RMSE acts as the objective function, i.e., it is used to tune the parameters of the function f , so that the RMSE of the whole dataset satisfies

$$\text{RMSE} \leq \delta, \quad 0 < \delta \ll 1, \quad (6)$$

on a scaled dataset where

$$R_i, V_i \in [0, 1], \text{ for } i = 1, 2, \dots, n. \quad (7)$$

1.7 Unit conversion factors

The data available here are represented in $(R, V) = (\text{Bohr}, \text{Hartree})$ units, where R is the atomic distance value and V is the energy value. The energy unit conversion factors are shown in Table 3, which is obtained from National Chiao Tung University’s website [26], for example, 1 Hartree = 27.2107 eV; the atomic distance conversion factor is shown in Eq.(8).

$$1 \text{ atomic unit (a.u.)} = 1 \text{ Bohr} = 0.529177249 \text{ Angstrom (A)}. \quad (8)$$

Table 3: Atomic energy conversion factors.

	Hartree	eV	kcal/mol	kJ/mol	cm ⁻¹
Hartree	1	27.2107	627.503	2625.5	219474.63
eV	0.0367502	1	23.0609	96.486 9	8065.73
kcal/mol	0.00159362	0.0433634	1	4.18400	349.757
kJ/mol	0.00038088	0.01036410	0.239001	1	83.593
cm ⁻¹	4.55633×10^{-6}	1.23981×10^{-4}	0.00285911	0.0119627	1

2 Machine learning tools for chemical data

Throughout this book, the main basis of approaches used can be abstracted into a collection of machine learning approaches, this suggests at least high-level descriptions of tools commonly used for machine learning problems are necessary; such as features, models, and optimization techniques that are relevant for molecular machine learning.

2.1 The features of chemical data

Analogous to general machine learning problems, machine learning that addresses chemical problems based on the geometry of molecules require **feature extraction** process to ensure sensible accuracy for prediction. Features essentially are the transformed numerical data from raw data that encode the pattern of the data well enough (depending on the subjective goal). Typically features that are neither too complicated nor too simple in terms of mathematical operations and the size of the features are desirable, i.e., balancing the trade-off between accuracy and efficiency (achieving both would be ideal). Additional set of requirements for features based on molecular geometries, is that the features need to be invariant with respect to rotation, translation, and permutation (of atoms with the same type).

Another similar term is **molecular fingerprint**. Molecular fingerprint is almost synonymous to molecular fingerprint, it describes unique representation of molecules in numerical form. The difference from molecular features is that feature is used for more general machine learning applications not limited to chemical data.

2.1.1 Dressed atoms, bonds, angles, and torsion (DA, DB, DN, DT)

We start from the dressed n -body graph-based feature. These features explains the fundamental of designing atomic features in particular, as usually they encapsulate the n -body interactions incrementally (so far, usually 4-body or dihedral features are the stopping point of the increments).

Dressed atoms (DA) is expressed as the sum of atomic-energies, i.e.,

$$E^{\text{DA}} = \sum_{a \in S^{(da)}} n_a E_a^{\text{atom}}, \text{ for } S^{(da)} := \{\text{H, C, N, O, F, ...}\}. \quad (9)$$

where A is the set of atom-types (unique set if atom-types queried from the whole dataset); E^{atom} is obtained by fitting against the true energy E . It is a common practice to subtract E^{DA} from the total energy E . Therefore the target for fitting of the first level becomes

$$E_t^1 = E_t - E^{\text{DA}}, \quad (10)$$

where E_t is the target energy.

Apart from the atomic energies, molecules also inherently contain the bonding-energies, which depend on the bond-types and the bond-level. The dressed bonds (DB) energy is defined by

$$E^{\text{DB}} = \sum_{b \in S^{(db)}} n_b E_b^{\text{bond}}. \quad (11)$$

where B is the set of bond types. Here E_b^{bond} is learned in a similar manner as the atomic energies in (9). We can have several variations of sets of bond types based on how we classify the covalent bonds. Here we distinguish the orders of the bonds (1, 2, 3, etc), and the types of bonds (C-C, C-H, C-O, etc), therefore, with

$$\text{bond_orders} := \{1, 2, 3\},$$

and the set $S^{(da)}$, we can construct

$$S^{(db)} := C^R(S^{(da)}, 2) \times \text{bond_orders},$$

where $C^R(S, r)$ is the combination with replacement given a set S and number of sample r . Therefore explicitly we have

$$S^{(db)} := \{1\text{CC}, 2\text{CC}, \dots, 1\text{CH}, \dots, 1\text{FF}, \dots, 2\text{OO}\}, \quad |S^{(db)}| = 21.$$

If this level of energy will also be eliminated from the total energy of the molecular system, then the energy target of the second level becomes

$$E_t^2 = E - E^{\text{DA}} - E^{\text{DB}}, \quad (12)$$

For consideration, a common practice for molecular data processing, in particular when directly related to the structure of the molecules, is to pre-process the molecules such that the Hydrogens (united atom representation) are removed therefore $|S^{(db)}|$ is smaller.

We can extend the definitions of DA and DB into dressed angles (DN) and dressed torsions (DT) in a similar fashion in terms of the target energy and the number of elementary structures involved. More detailed technical explanation of DN and DT will be contained within the technical description on how to extract the substructures in sections 2.1.2.1 and 2.1.2.2.

2.1.2 SMILES string

Typically, the graph structure of a molecule is encoded in a line of string called the **simplified molecular-input-line-entry system (SMILES)** string [27]. In the context of extracting the information of atomic covalent bonds, the only rules that we are interested in from the SMILES encoding are the ones that dictate the atom types; and the rules that determine the bond-types and bond-levels, simply we restrict to parsing:

$$(-, =, \#) \implies (1, 2, 3)$$

where the left hand side triplet is the encoded SMILES tokens of the corresponding right hand side triplet of bond-levels. It can be noticed that in this case the single bond "-" is omitted, (in fact, it is the general convention).

Let us see an example of the SMILES string of `dsgdb9nsd_066222.xyz` geometry from QM9 dataset:

$$\text{CC1(CNCC\#N)CN1} \implies \text{C}_6\text{H}_{11}\text{N}_3, \quad (13)$$

this means given a SMILES string on the left hand side, a correct parser will output the molecular formula on the right side. However we are interested in the graph $G = (V, E)$ structure rather than molecular formula, where we represent an atom as a node $v \in V$ and a covalent bond as an edge $e \in E$. In order to extract the graph structure, we parse the SMILES string in a depth-first-search (DFS) way, with some backtrack mechanism spawned whenever:

- A branch exists. A branch starts by "(" token and ends on ")" token (without quotes). In the above SMILES, a branch starts from the third C and ends on the first N, this implies a bond between the second C and the third C.
- A cyclical structure exists. A cycle starts from a numbered atom and ends on an atom with the same number. In the above SMILES, a cycle starts from "C1" and ends on "N1", this implies a bond between the corresponding C and N.

A [Julia](#) package named [MolecularGraph.jl](#) [28] is useful for having many tools relating to the operations of SMILES string and graph representation of molecules, where one of them is generating a graph structure given the SMILES string representation of a molecule. It uses the [Graph.jl](#) [29] abstraction for the data structure organization. Although for extracting the subgraphs, e.g., if one want to extract the n -body fragments (such as diatomic and triatomic subgraphs), one would still need to implement the graph traversal manually.

Given the subgraphs, we count the occurrence of each unique subgraph composition and store them into a feature vector for each molecule, then we dub this as the dressed features: **Dressed [Bond, Angle, Torsion]**. The illustration of the whole process can be seen in figure 1.

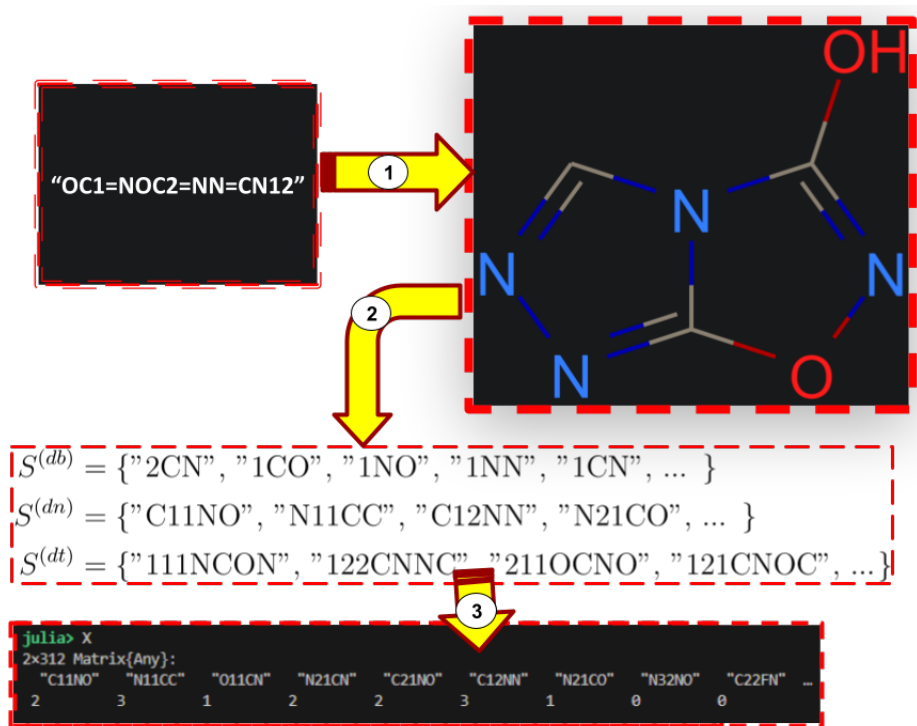


Figure 1: The dressed feature extraction process: starting from the SMILES string $\rightarrow$ graph representation $\rightarrow$ symmetrized subgraphs $\rightarrow$ dressed feature matrix.

2.1.2.1 General molecular subgraph and bond extraction In general, there are many ways to extract any type of the molecular subgraph (bond, angle, and torsion) $S \subseteq G$. Our approach here is we start from a *central* structure (node or edge) and then traverse all of the combinations of its *left* and *right* neighbour structures. The main difference of the traversal approach used between the molecular subgraph types is only the *level* of traversal, e.g., for bond, from the central structure we only need to traverse its immediate neighbours, while for angle we need to traverse the *neighbours of neighbours*, and so on.

A bond is a subgraph containing three structures, represented by a tuple of (v_{-1}, e_0, v_1) , $v_{-1} \in S_{-1}$, $v_1 \in S_1$, where S_{-1} is the collection of the left side structures that are one step away from $e_0 \in E$, and S_1 is the right side's. A molecule may contain many bonds. For extracting a bond, we pick any edge $e_0 := e \in E$ (typically we start from the edge with the earliest index) as the central structure, and then we traverse the combinations of all of the left and right structures $S_{-1} \times S_1$. Each pair $(v_{-1}, v_1) \in S_{-1} \times S_1$ would result in a bond when combined with the central structure e_0 . Then we pick again a new edge to be set as the central structure and repeat the whole process, we do this $\forall e \in E$.

As mentioned before that for different types of subgraph, the main difference is the level of traversal from the center. From the bond tuple example, we can generalize an " n -body" subgraph $G^{(n)}$ that contains $m = 2n - 1$ structures:

$$G^{(n)} := (s_{-n+1}, s_{-n+2}, \dots, s_0, \dots, s_{n-2}, s_{n-1}), \text{ for } n > 1, \quad (14)$$

where s_0 is the center structure. For odd n then s_0 would be a node $v \in V$; if n is even, then s_0 would be an edge $e \in E$. The structure type will always be alternating between node and edge where the pattern starts from the center, for example, the bond subgraph described earlier consists of (node, edge, node). The combination of the neighbours becomes $S_{-n+1} \times S_{-n+2} \times \dots \times S_{-1} \times S_1 \times \dots \times S_{n-2} \times S_{n-1}$ for $n > 1$ and $m = 2n+1$. We can infer that the order of the cost of the traversal for extracting a maximum size of n -body subgraphs for a molecule containing N atoms is $\mathcal{O}(Nn)$.

One additional important process is to preserve the symmetricity of the subgraph representation, due to the fact that the traversal direction of the graph is relative depending on the order of the branch visited. For each of the subgraph types: bond, angle, and torsion, the symmetrization procedure will be different.

Consider a concrete example, given the molecular graph structure of the SMILES string in (13), then for the extraction of bonds, the parser would output:

`bonds := {1CC, 1CC, 1CC, 1CN, 1CN, 1NC, 1CC, 3CN, 1CN}`,

where for each entry, the format is

`concat(bond_level, left_atom, right_atom)`,

where

`concat("A", "B", ..., "Z") = "AB...Z"`.

Notice that the implicit H atoms are not parsed; also notice the occurrence of "1CN" and "1NC", we treat those two as the same bond-type (symmetric), i.e., we encode the data structure as undirected graph, which is the (desired) consequence of choosing combination with replacement to sample the bond-types in the previous section. In practice the symmetrization is achieved by simply sorting the `[source_atom,`

`destination_atom]` lexicographically. After symmetrization, then list of bonds becomes

`bonds` := {1CC, 1CC, 1CC, 1CN, 1CN, 1CN, 1CC, 3CN, 1CN}.

2.1.2.2 Extracting angles & torsions From the general definition in (14), angle subgraph has the form of $G^{(3)} := (v_{-2}, e_{-1}, v_0, e_1, v_2)$, meanwhile torsion subgraph is $G^{(4)} := (v_{-3}, e_{-2}, v_{-1}, e_0, v_1, e_2, v_3)$. The traversal procedure itself simply extends bond subgraph’s procedure in terms of the collections of neighbours to be explored. As mentioned before, the symmetrization process is crucial and it is different for each subgraph type. Angle subgraph analogy is two bonds joined together with a central node, therefore we only need to sort the *endpoint* nodes and the edges will follow, the full algorithm is presented in algorithm 1. Meanwhile torsion subgraph analogizes to two bond subgraphs joined by an edge, therefore the sorting rule involves the *inner* and *outer* nodes, as seen in algorithm 2.

Algorithm 1 Angle symmetrization

Input: A string s of 5 characters $s_1, \dots, s_5$ describing an angle, where:

- s_1 := center atom type
- s_2 := left bond order
- s_3 := right bond order
- s_4 := left atom type
- s_5 := right atom type

Output: An ordered sequence of characters s'

if $s_4 = s_5$ **then**

Sort the order of bonds $[s_2, s_3]$ lexicographically and store them in s'_{bonds}

else if $s_4 \neq s_5$ **then**

Sort the atoms $[s_4, s_5]$ lexicographically, and store them in s'_{atoms}

Bond orders $[s_2, s_3]$ should follow the indexing order of the sorted atoms, i.e., if $[s_4, s_5]$ swap positions then $[s_2, s_3]$ should do the same; then store them in s'_{bonds} .

end if

$s' \leftarrow [s_1, s'_{\text{bonds}}, s'_{\text{atoms}}]$

Examples:

- C11HC $\rightarrow$ C11CH,
- C21OC $\rightarrow$ C12CO,
- C21CH $\rightarrow$ C21CH.

Algorithm 2 Torsion symmetrization

Input: A string s of 7 characters $s_1, \dots, s_7$ describing a torsion, where:

- s_1 := center bond order
- s_2 := left bond order
- s_3 := right bond order
- s_4 := outer left atom type
- s_5 := inner left atom type
- s_6 := inner right atom type
- s_7 := outer right atom type

Output: An ordered sequence of characters s'

```
Initialize swap_bonds := false and sort_bonds := false
# Sorting atoms:
if  $s_4 > s_7$  lexicographically then
    Swap the atoms: swap  $s_4$  and  $s_7$ , then swap  $s_5$  and  $s_6$ 
    Set swap_bonds := true
else if  $s_4 = s_7$  then
    if  $s_5 = s_6$  then
        Set sort_bonds := true
    else if  $s_5 > s_6$  lexicographically then
        Swap inner atoms: swap  $s_5$  and  $s_6$ 
        Set swap_bonds := true
    end if
end if
# Sorting bonds:
if swap_bonds = true then
    Swap the bonds: swap  $s_2$  and  $s_3$ 
else if sort_bonds = true then
    Sort  $[s_2, s_3]$  lexicographically
end if
 $s' \leftarrow s$  after applying the sorting mechanism above
```

Examples:

- 121CCCC $\rightarrow$ 112CCCC,
- 121COCO $\rightarrow$ 121COCO,
- 121OCCO $\rightarrow$ 112OCCO,
- 121OCOC $\rightarrow$ 112COCO.

2.1.3 Higher resolution features

The n -body features to include more precision, in particular, by features in real numbers, there are many such features across the literature. The basic ideas of

these features will be explained in this section, in higher level of computational description in which we are interested in using the programs.

In general, each set of features results in an array data structure with $(n_{\text{mol}}, n_{\text{atom}}^l, n_{\text{af}})$, where n_{mol} is the number of molecules. Typically this is stored in a vector of matrices, where each matrix has different row size, due to the possibility of each molecule having different number of atoms n_{atom}^l ; meanwhile the number of atomic features n_{af} is always uniform across the array, it is possible that n_{af} itself indexes a matrix, i.e., $n_{\text{af}} := n_{\text{row}} \times n_{\text{col}}$, where n_{row} is the number of rows, and n_{col} is the number of columns. These continuous features encapsulate radial distributions (2-body) and angular distributions (3-body).

2.1.3.1 Many Body Density Functional (MBDF) Many body density functional (MBDF) (and its variant, convolutional MBDF (CMBDF)) [6, 30], computes the area of the radial distributions and the area of the angular distributions to capture the 2 and 3-body features. The whole form of 2 and 3 body functions for atom i is summarized as

$$\begin{aligned} F_j^{(2)}[i] &= \int_0^{r_c} dr \omega_j^{(3)}(r) \partial_r^j \rho_i^{(2)}(r), \text{ for } j = 0, 1, 2, \\ F_j^{(3)}[i] &= \int_0^\pi d\theta \omega_j^{(3)}(\theta) \partial_\theta^j \rho_i^{(3)}(\theta), \text{ for } j = 0, 1, \end{aligned} \quad (15)$$

where $\omega_j^{(n)}(\cdot)$ is a heuristic weighting function of n -body values and $\rho_i^{(n)}(\cdot)$ is a function that computes the distribution of n -body values; r_c and π are cutoff constants for distance and angle respectively.

MBDF produces $n_{\text{af}} = 5$ for each atom. For each molecule l , MBDF returns a matrix with $(n_{\text{atom}}^l, n_{\text{af}})$ size. The difference with the convolutional version is that the computation of the feature set is much faster by making use of convolution operator.

2.1.3.2 Atom-centered Symmetry Functions (ACSF) Atom-centered symmetry functions (ACSF) [10, 3], computes 4 symmetry functions for each atom, they are composed of radial and angular functions. ACSF only considers the set of atomic coordinates as the primary input. For each molecule l , ACSF returns a matrix with $(n_{\text{atom}}^l, n_{\text{af}})$ size. The hyperparameters include (but not limited to) [31]:

- cut-off radius r_{cut} (in Angstrom),
- width of the Gaussian,
- damping scalar.

Given the default settings, the output gives $n_{\text{af}} = 51$. ACSF is computed using `DescriptorZoo.jl` [31, 32] library package

2.1.3.3 Smooth Overlap of Atomic Positions (SOAP) Smooth overlap of atomic positions (SOAP) [4] encodes the atomic representation by expansion of Gaussian atomic density, based on spherical harmonics and radial basis functions. Overall, SOAP has similar idea and structure to ACSF. The hyperparameters include, but not limited to:

- cut-off radius r_{cut} (in Angstrom),

- the number of radial basis functions n_{rbf} ,
- maximum degree of spherical harmonics n_{sphere} .

The initial hyperparameters were set to $(r_{\text{cut}}, n_{\text{rbf}}, n_{\text{sphere}}) = (8.0, 2, 2)$, this gives $n_{\text{af}} = 165$. SOAP is extracted using `D2Scribe` [33] program library.

2.1.3.4 Faber–Christensen–Huang–Lilienfeld (FCHL) Faber–Christensen–Huang–Lilienfeld (FCHL) [17]. Essentially FCHL encodes radial distribution by the two-body term, and encodes the mean distances and angular information by three-body term. FCHL considers atomic coordinates and the nuclear charges as the primary inputs. For each molecule, FCHL returns an array of size $(\max_l n_{\text{atom}}^l, 5, n_{\text{neigh}})$, where n_{neigh} is the number of atomic neighbours; almost in all cases $n_{\text{neigh}} = \max_l n_{\text{atom}}^l$, the only exception is when periodic boundary cell mode is used (for crystals). The hyperparameters are:

- cut-off radius (in Angstrom), the default is $r_{\text{cutoff}} = 5.0$,
- number of neighbours, the default is $n_{\text{neigh}} = 23$,
- periodic cells boundaries, the default is `pbcs = None`.

For the QM9 dataset, FCHL returns an array with size $(133885, 29, 5, 29)$. FCHL is extracted using `QML` [34] library package.

2.1.4 Feature selection

Given the complete features of a dataset. In some cases it is desirable to select the most relevant features for computational efficiency and accuracy. Fitting by only the selected features is almost always faster than the default features, due to the reduced overall size of the features and the process of the selection being independent of the fitting itself (despite sometimes the feature selection process takes considerable amount of time depending on the size of the dataset). Quite often the feature selection also leads to higher fitting accuracy since if some components of the features are noises, they would be more likely to be discarded. In this section, the procedures of chemical data feature selection are explained.

2.1.4.1 Atomic feature selection

- Given an array f containing the features with the size $(N, n_{\text{atom}}^l, n_{\text{af}})$, to select the most relevant components of features, we do the **principal component analysis** (PCA). First, compute the mean vectors and the matrix S by

$$s = \frac{1}{N} \sum_l \frac{1}{n_{\text{atom}}^l} \sum_i f_{i:}^l,$$

$$S = \frac{1}{N} \sum_l \frac{1}{n_{\text{atom}}^l} \sum_i f_{i:}^l (f_{i:}^l)^\top, \quad (16)$$

for $l = 1, \dots, N, \quad i = 1, 2, \dots, n_{\text{atom}}^l$,

where n_{atom}^l is the number of atom in molecule l ; then compute the covariance matrix

$$C = S - ss^\top. \quad (17)$$

The correlation matrix can be formed by

$$C' := DCD, \quad D = \text{Diag}(1/\sqrt{C_{ii}}); \quad (18)$$

alternatively, the sensitivity of the atomic features is viable as the scaling factor

$$D = \text{Diag}(1/\sigma_j), \quad (19)$$

where the sensitivity vector's entries are

$$\sigma_j = \frac{1}{\sum_l n_{\text{atom}}^l} \sum_{li} d_{ij}^l, \quad (20)$$

where

$$d_{ij}^l := \max_k |f_{ij}^l - (\hat{f}_k)_{ij}^l| \quad \text{for } k = 1, 2, \dots, \quad (21)$$

where $\hat{f}_k$ is the set of perturbed atomic features, obtained by perturbing the atomic coordinates randomly

$$x \leftarrow x \pm 0.01, \quad (22)$$

and using the perturbed coordinates to extract the atomic features $\hat{f}_k$.

Select the eigenvectors such that they correspond to the n_{af} largest eigenvalues. They are obtained by first doing the spectral decomposition of the correlation matrix

$$C' = Q\Lambda Q^\top, \quad (23)$$

where Λ is the diagonal matrix containing the k th eigenvalue Λ_{kk} and Q is the matrix containing the k th eigenvector $Q_{:,k}$, then we permute the columns of Q and the diagonal entries λ_i of Λ such that

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{n_f^{\text{atom}}}, \quad (24)$$

and select

$$\hat{Q} = Q_{:,1:n_{af}}. \quad (25)$$

Finally the transformed feature can be obtained by

$$f_{i:}^l \leftarrow \hat{Q}^\top (f_{i:}^l - s), \quad \text{for } l = 1, 2, \dots, N. \quad (26)$$

- Scale

$$f_{ij}^l \in [0, 1] \quad (27)$$

by

$$\underline{f}_j := \min_{li} f_{ij}^l, \quad (28)$$

$$\overline{f}_j := \max_{li} f_{ij}^l, \quad (29)$$

$$f_{ij}^l \leftarrow (f_{ij}^l - \underline{f}_j) / (\overline{f}_j - \underline{f}_j), \quad \text{for } j = 1, \dots, n_f^{\text{select}}. \quad (30)$$

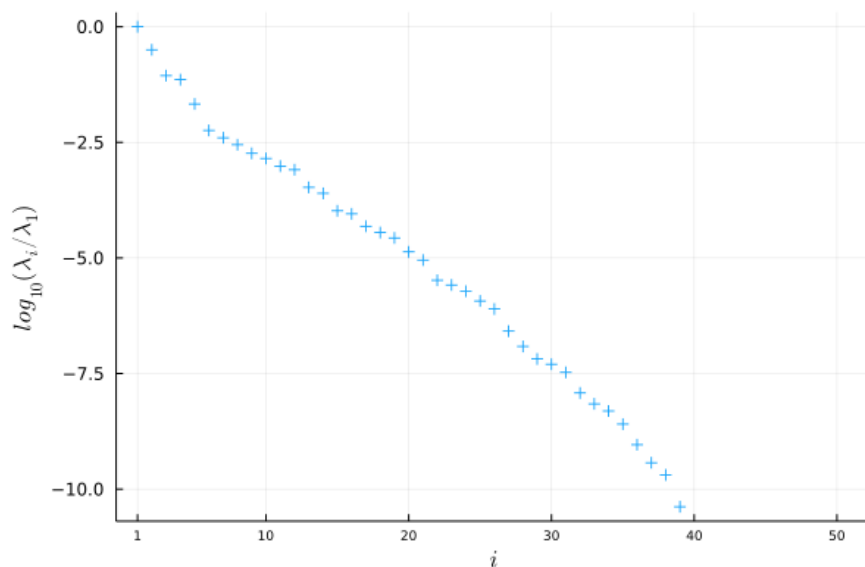


Figure 2: The ratio of each eigenvalue against the largest eigenvalue, obtained from the PCA of ACSF on the atomic level computed on the QM9 dataset.

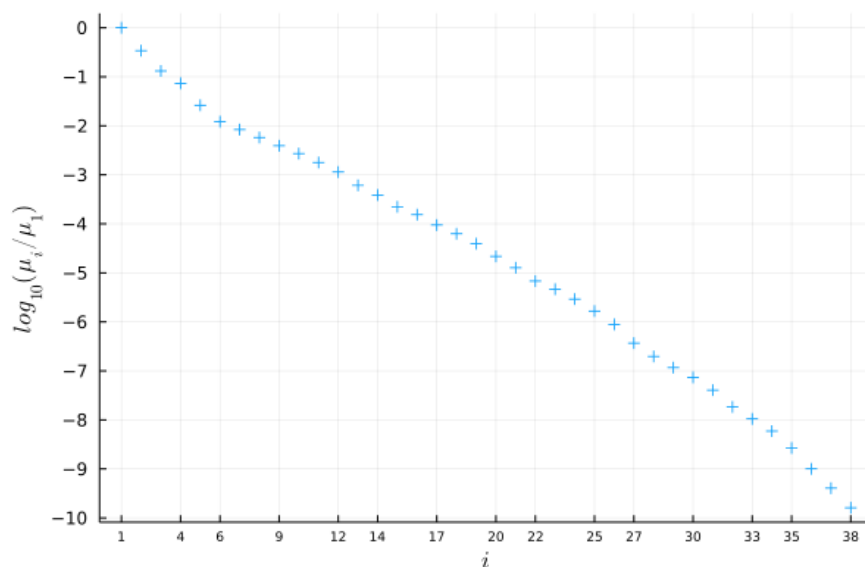


Figure 3: $\mu_i := \sum_{j \geq i} \lambda_j$ for $k = 1, 2, \dots, n_f$, obtained from the PCA of ACSF on the atomic level computed on the QM9 dataset. This shows the distribution of the eigenvalues when i amount of the (sorted) eigenvalues are dropped.

Timing and storage Features based on atomic symmetry functions in general has $\mathcal{O}(n_{\text{atom}} n_{\text{cutoff}} c)$ time complexity, where c is a factor referring to the feature extractor’s method of computation (e.g., ACSF requires c amount of radial symmetry function computations), and n_{cutoff} is the number of atoms included within the cutoff radius, as it is required to sweep through the atom and its neighbours within the cutoff radius each time an atomic symmetry quantity is queried. The resulting features of each atom is a vector with length n_{af} , therefore a molecule has an order of $\mathcal{O}(n_{\text{atom}} n_{af})$ storage complexity. In total for N molecules, each time an atomic feature extraction is queried, it has $\mathcal{O}(N n_{\text{atom}} n_{\text{cutoff}} c)$ time complexity and $\mathcal{O}(N n_{\text{atom}} n_{af})$ storage complexity.

In general, each query of PCA needs to sweep all entries of the feature matrix multiple times. Specific for atomic features, it can be broken down into:

- Computing mean vector s requires $\mathcal{O}(Nn_{\text{atom}}n_{af})$ time and $\mathcal{O}(n_{af})$ storage.
- Symmetric matrix S requires $\mathcal{O}(Nn_{\text{atom}}n_{af}^2)$ time and $\mathcal{O}(n_{af}^2)$ storage.
- The covariance matrix can be computed with $\mathcal{O}(n_{af}^2)$ operations given the mean vector s and the symmetric matrix S , with also $\mathcal{O}(n_{af}^2)$ storage complexity.
- The correlation matrix can be computed by $\mathcal{O}(n_{af}^2)$ operations given the covariance matrix C and the scaling factors D , requires $\mathcal{O}(n_{af}^2)$ storage.
- Spectral decomposition of the symmetric matrix requires $\mathcal{O}(n_{af}^3)$ time and $\mathcal{O}(n_{af}^2)$ storage.
- Projecting the atomic features into lower dimensional space requires $\mathcal{O}(n_{af}^2)$ time for each atom, hence for all molecules, it requires $\mathcal{O}(Nn_{\text{atom}}n_{af}^2)$; the storage complexity is $\mathcal{O}(Nn_{\text{atom}}n_{af})$.

Overall, in terms of the timing and storage, the raw atomic feature extraction itself should be the upper bound of the whole atomic feature processing block, since the factor c itself could also be a composition of many factors depending on the method of the feature extractor.

2.1.4.2 Molecular feature selection The molecular feature matrix $F \in \mathbb{R}^{N \times n_f}$ where N is the number of molecules in the dataset and n_f is the number of features, needs to be precomputed and stored in the non-volatile storage in order to avoid re-computation. The molecular feature matrix often can be obtained through processing the array of atomic features. The atomic features is often in the form of row vector f_i^l with length n_{af} for each atom i in a molecule l .

- l th molecule contains n_{atom}^l number of atoms and a set of atom types $P = \{\text{H}, \text{C}, \text{O}, \dots\}$ with cardinality n_{type}^l , the molecular features are computed by

$$s_j^l = \sum_{m \in M_j} f_m^l, \quad (31)$$

where M_j is the set of index which contains the row positions of atom type $j \in P$ in the f^l matrix. Furthermore we add

$$N_j = |M_j|, \quad (32)$$

combined into

$$\hat{N} = [N_{\text{H}}, N_{\text{C}}, N_{\text{O}}, \dots, 1/n_{\text{atom}}^l]. \quad (33)$$

The feature vector of the l th molecule becomes

$$F_l := [s_j^l, \hat{N}] \in \mathbb{R}^{1 \times n_f^{\text{mol}}}, \forall j \in P, \quad (34)$$

where

$$n_f^{\text{mol}} := 5n_f + 6, \quad (35)$$

in most cases this is a sufficiently good feature vector for fitting.

Additional features can be introduced, such as the sum of squares

$$\hat{s}_j^l = \sum_{m \in M_j} (f_{m:}^l)^2; \quad (36)$$

the binomial feature

$$S_j^l = \sum_{m \in M_j} f_{m:}^l (f_{m:}^l)^\top; \quad (37)$$

since S_j^l is a symmetric matrix, only the upper triangular and the diagonal entries are needed. Hence if we include both the sum of squares and the binomial features

$$F_l := [s_j^l, \hat{s}_j^l, S_j^l, \hat{N}] \in \mathbb{R}^{1 \times n_f^{\text{mol}}}, \forall j \in P, \quad (38)$$

where

$$n_f^{\text{mol}} := \frac{5}{2}(n_f^2 + n_f) + 6, \quad (39)$$

this defines the fingerprint matrix $F \in \mathbb{R}^{N \times n_f}$.

- Do feature selection by using PCA in a similar manner as (17) – (26) to take the subset of the columns of W , this can be done by the following:

$$\begin{aligned} s &= \frac{1}{N} \sum_l F_l, \\ S &= \frac{1}{N} \sum_l F_l (F_l)^\top, \\ \text{for } l &= 1, 2, \dots, N, \end{aligned} \quad (40)$$

compute the covariance matrix

$$C = S - ss^\top, \quad (41)$$

compute the correlation matrix

$$C' := DCD, \quad D = \text{Diag}(1/\sqrt{C_{ii}}), \quad (42)$$

and its spectral decomposition

$$C' = Q\Lambda Q^\top, \quad (43)$$

permute the columns of Q and the entries of Λ such that

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{n_f^{\text{mol}}}, \quad (44)$$

and select

$$\hat{Q} = Q_{:,1:n_{mf}}. \quad (45)$$

Obtain the transformed feature by

$$F_{l:} \leftarrow \hat{Q}^\top (F_{l:} - s), \text{ for } l = 1, 2, \dots, N. \quad (46)$$

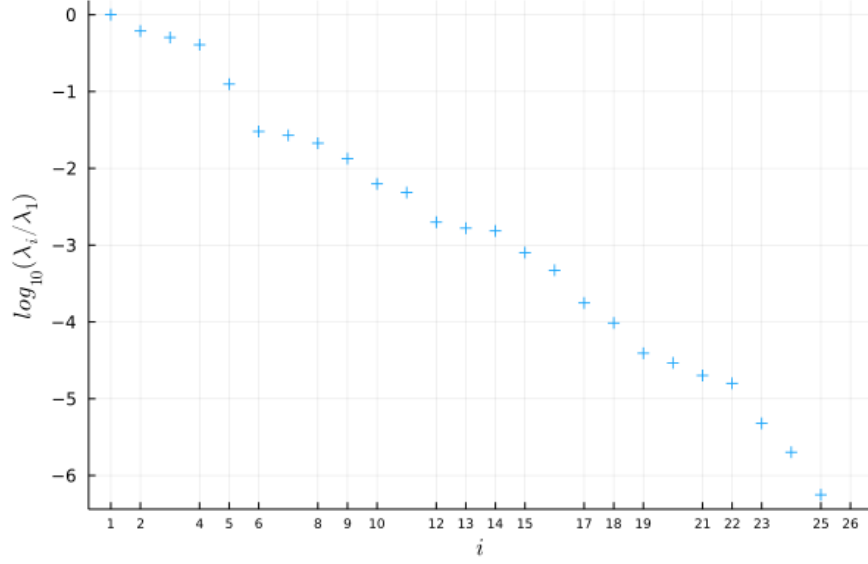


Figure 4: The ratio of each eigenvalue against the largest eigenvalue with hyperparameter $n_{af} = 4$, obtained from the PCA of ACSF on molecular level of QM9 dataset.

- Scale F such that

$$F_{lj} \in [0, 1] \quad (47)$$

by

$$\underline{F}_j := \min_l F_{lj} \quad (48)$$

$$\overline{F}_j := \max_l F_{lj} \quad (49)$$

$$F_{:j} \leftarrow (F_{:j} - \underline{F}_j) / (\overline{F}_j - \underline{F}_j), \text{ for } j = 1, 2, \dots, n_{mf}. \quad (50)$$

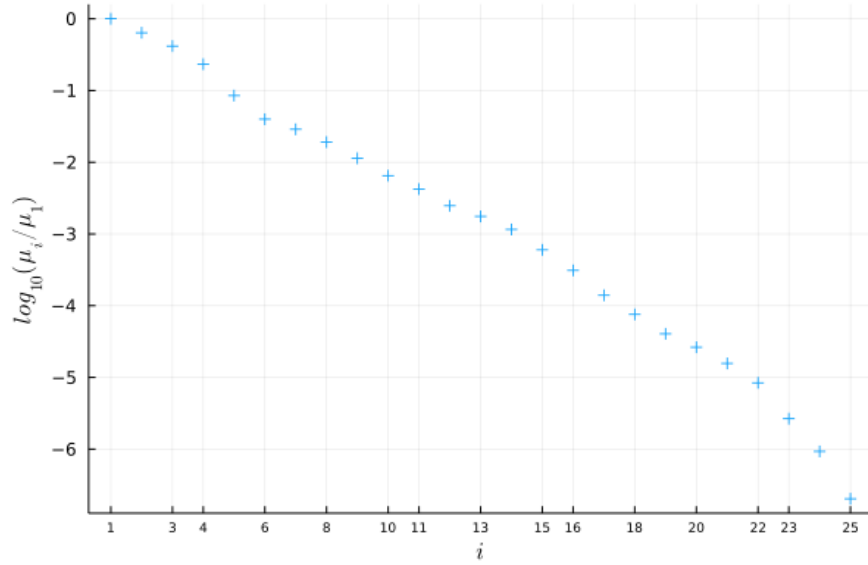


Figure 5: The distribution of the eigenvalues with hyperparameter $n_{af} = 4$. Obtained from the PCA of ACSF on molecular level of QM9 dataset.

Timing and storage The timing breakdown of the molecular feature selection is as the following:

- In the default setting, the molecular feature is only computed by the sums of the atomic features, for each molecule this has $\mathcal{O}(n_{\text{atom}}n_{\text{af}})$ order of operations, therefore for all molecules in the dataset the time complexity is $\mathcal{O}(Nn_{\text{atom}}n_{\text{af}})$; the storage requirement is $\mathcal{O}(Nn_{\text{mf}})$.
- Additional features which increase the complexity noticeably is the binomial feature, each binomial feature matrix takes $\mathcal{O}(n_{\text{af}}^2)$ order of computation and $\mathcal{O}(n_{\text{af}}^2)$ storage, for N this becomes $\mathcal{O}(Nn_{\text{atom}}n_{\text{af}}^2)$ time and storage requirement.

For molecular features, the PCA has similar time and storage complexity to the atomic feature counterpart, the only differences are the initial mean vector s , the symmetric matrix S computation, and the final projection, since the feature is now a vector with length n_{mf} for each molecule in contrast to the atomic features with $\mathbb{R}^{n_{\text{atom}} \times n_{\text{af}}}$ matrix for each molecule.

2.1.5 Alternative way to scale the features

Other than the min-max scaler as demonstrated in (30) and (50), we may use empirical cumulative distribution function (ECDF) to scale the feature vectors. ECDF is basically a "discrete" version of CDF constructed by fitting the CDF using data points, generally more data points lead to smoother ECDF; we can think of fitting and prediction using the ECDF as the process of binning the data points. It is stated that ECDF has a desirable property, which is it does not suffer from outlier sensitivity [35]. Another nice built-in feature that ECDF has is, it is possible to directly control the bias and variance trade-off by controlling the number of data points for the ECDF fitting; more data points generally lead to higher variance, while less data points give higher bias.

First, to compute the ECDF, we may select the subset of the data points as the "training data"; let $X \in \mathbb{R}^{n_{\text{data}} \times n_{\text{features}}}$, then

$$X^{\text{train}} \subseteq X,$$

then for each feature, i.e., for each column vector $X_{:k}$, we can compute its ECDF by

$$\text{ECDF}(X_{:k}; X_{:k}^{\text{train}}) = \frac{1}{n} \sum_i^n 1_{X_{ik} \leq X_{jk}^{\text{train}}}, \quad \text{for } j \in \text{eachindex}(X_{:k}^{\text{train}}). \quad (51)$$

In the case of molecular features, it is quite straightforward, each ECDF is fitted using each column of the molecular feature matrix (optionally, the selection of the row indices for "fitting" and "prediction"). Meanwhile for the atomic features, the vector of atomic feature matrices needs to be stacked into one big matrix of atomic features in order to compute the ECDF.

2.1.6 Feature transformation by message passing

The idea of *message passing* is to make the atoms within a molecule to be aware of each others' features. The message passing techniques were first introduced by [12] in their Message Passing Neural Networks paper. A molecule is represented by undirected graph G where the atoms are the nodes $v \in G$ where it indexes the node

features x_v ; and the edges are the interactions between the atoms (whether it is covalently bonded or not), it is also possible for each edge to contain edge features e_{vw} . Here it is assumed that the graph is fully connected with respect to a certain cutoff r_{cutoff} radius for each node.

In general, there are three major phases for message passing transformation: **the aggregation phase, the update phase, and the readout phase**. The aggregation and update phases are run for T time steps, and the readout phase is applied in the end. Suppose that h_v^t is a hidden state which represents the features of node v at time step t , we do an initialization at time step $t = 0$ as

$$h_v^{(0)} := f_i^l, \quad e_{vw}^{(0)} := r_{ij}; \quad (52)$$

then for each time step t , we do an aggregation phase,

$$m_v^{t+1} = \frac{1}{|N(v)|} \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}^t), \quad \text{for } v \in G \quad (53)$$

which is just the mean of the neighbours' features, with an aggregation function M_t at time step t , which in the simplest case is

$$M_t(h_v^t, h_w^t, e_{vw}^t) := \text{concat}(h_v^t, h_w^t, e_{vw}^t), \quad (54)$$

where $\text{concat}(\cdot, \cdot)$ is the concatenation. The sum ensures that m_v^{t+1} is invariant with respect to the set of neighbours $N(v)$. The update phase for the hidden states is defined by

$$h_v^{t+1} = U_h^t(h_v^t, m_v^{t+1}), \quad e_{vw}^{t+1} = U_e^t(h_v^t, h_w^t, e_{vw}^t), \quad (55)$$

where U_h^t is an update function for hidden states at time step t , one of the most common example is

$$U_h^t(h_v^t, m_v^{t+1}) := \sigma(W^t m_v^{t+1}), \quad (56)$$

where σ is an activation function which takes in the product of the learned "weight" matrix W^t at time step t with the aggregated vector m_v^{t+1} ; it is also possible to have a fixed quantities instead of a learned matrix hence there exist minimal numbers of hyperparameters to be optimized. It is common to keep the edge features fixed throughout the time steps, i.e.,

$$U_e^t(h_v^t, h_w^t, e_{vw}^t) := e_{vw}^t, \quad (57)$$

in some cases it is possible to use a learned function analogous to (56). After T time steps, the readout phase is defined by

$$f_i^l = R(h_v^T) \quad \text{for } v \in G, \quad (58)$$

where R is the readout function, one of the simplest example is

$$R(h) := h. \quad (59)$$

2.1.7 Completeness of molecular features

Features of a molecular geometry (or configuration) is said to be complete if they describe distinct (numerical) values for each distinct molecular geometry; otherwise if the features of some distinct geometries result in equal numerical value, then the features are incomplete.

One way to check the completeness is by conducting a quality test of the molecular features. Consider the lattices

$$\mathcal{L} = \sqrt{2}Z_3, D_3, \quad (60)$$

they have a fixed set of distances

$$\mathcal{D} := \{0, \sqrt{2}, \sqrt{4}, \sqrt{6}, \sqrt{8}, \sqrt{10}, \dots\}, \quad (61)$$

the smallest nonnegative distance corresponds to the equilibrium distance, which is $r_{eq} = \sqrt{2}$. The coordinates of $\mathcal{L} = \sqrt{2}Z_3$ lattice are defined by

$$x := (\sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3), \text{ where } x_i \in \mathbb{Z}, \quad (62)$$

meanwhile the coordinates of $\mathcal{L} = D_3$ are defined by

$$x := (x_1, x_2, x_3), \text{ where } \sum_i x_i \text{ is even and } x_i \in \mathbb{Z}. \quad (63)$$

Let

$$S := \{x \mid \|x\|_2 < 3\}, \quad (64)$$

which implies the feasible set of distances

$$\mathcal{D}_{\mathcal{F}} := \{0, \sqrt{2}, \sqrt{4}, \sqrt{6}, \sqrt{8}\}. \quad (65)$$

For all possible combinations of $x_i \in \mathbb{Z}$, if (62) and (63) must satisfy (64), then we can define

$$\begin{aligned} A &:= \{x \text{ with (62)} \mid \|x\|_2^2 \leq 4\}, \\ B &:= \{x \text{ with (63)} \mid \|x\|_2^2 \leq 8\}, \end{aligned} \quad (66)$$

where the set A corresponds to $\mathcal{L} = \sqrt{2}Z_3$, and B corresponds to $\mathcal{L} = D_3$; this implies for A and B :

$$x_i \in \{0, \pm 1, \pm 2\}. \quad (67)$$

The goal here is to find possible "counterexamples" for completeness, by using the features computed from x_i data. The algorithm is as shown in Alg.(3).

Algorithm 3 Completeness check

```

1: for each  $\mathcal{L}$  and each size  $N$  do
2:   Let  $\Lambda$  be an empty storage.
3:   Pick a batch of random subsets  $\Gamma \in \mathbb{R}^{M \times N \times 3} \subset S$ , where each subset is in
   the form of a matrix  $T \in \mathbb{R}^{N \times 3}$  which corresponds to one data point.
4:   Compute the feature vector  $v^k$  from  $T^k$ , for  $k = 1, 2, \dots, M$ ; and normalize  $v$ 
   such that  $v_i \in [-1, 1]$  for  $i = 1, 2, \dots, |v|$ .
5:   Compare  $(T^k, v^k)$  and  $(T^j, v^j)$  for all  $k \neq j = 1, 2, \dots, M$ :
6:   for each  $k$  do
7:     if the feature vector  $v^k \equiv v^j$  then
8:       check for the isomorphism between  $T^k$  and  $T^j$ .
9:       if not isomorphic then
10:        mark  $j$  as an counterexample of  $k$ , record this in  $\Lambda$ .
11:       else if isomorphic then
12:        continue to the next  $j$ .
13:       end if
14:     end if
15:   end for
16: end for

```

In the first step of the algorithm, M corresponds to the size of the batch, where usually it is preferable to have it in the order of thousands, generally, larger M requires larger N ; meanwhile N corresponds to the number of atoms in the molecule, where $N > 3$ is of interest. The storage Λ in practice can be any data structures such as dictionary, vector, or list, however in this case a list of dictionaries will be preferable since for each element of Λ some key-value pairs will be needed, for example, the k index as key with a vector of j indices as its value. In step 3, T is a matrix of size $N \times 3$ where each row corresponds to a vector of atomic coordinates x . In step 4, the feature vector is in the form of

$$v = (v_1, v_2, \dots)$$

where v_i is the list of features of the i th atom. In practice, v_i is in the form of vector or matrix, here it should be generalized by arranging the entries of v_i in a consistent way for all i ; for example, if we use the features from Section 5, then

$$v_i = (U[i], Y[i], G[i]),$$

where $G[i] \in \mathbb{R}^{d_1 \times d_2}$ itself is a matrix, hence v_i can be arranged such that

$$v_i = (U[i], Y[i], G[i]_{1,1}, \dots, G[i]_{d_1,1}, G[i]_{1,2}, \dots, G[i]_{d_1,d_2}),$$

in fact, this is a column-major indexing which is preferred for performance by some programming languages, such as Julia; thus

$$v = (U[1], Y[1], G[1]_{1,1}, \dots, G[1]_{d_1,d_2}, U[2], Y[2], G[2]_{1,1}, \dots, G[2]_{d_1,d_2}, \dots).$$

The normalization of each component $w := v_i$ can be done by

$$\begin{aligned} \overline{w} &= \max w, \quad \underline{w} = \min w, \\ \mu &= 0.5(\overline{w} + \underline{w}), \quad \lambda = 0.5(\overline{w} - \underline{w}) \implies |w - \mu| \leq \lambda, \\ w_{\text{norm}} &:= \frac{w - \mu}{\lambda}. \end{aligned} \tag{68}$$

The comparison in step 7 can be done in several ways, the simplest way is by accepting j where

$$\|v^k - v^j\|_2 \leq \delta, \text{ where } \delta \ll 1;$$

another way would be by multiplying v by preferably some constants $c \geq 100$ (to avoid rounding errors) and round v to the nearest integer, then compare the rounded v^k and v^j .

2.2 Models for general chemical data fitting

In this section, the models related to this thesis are explained. By any means this section is not an exhaustive list of machine learning models available in the literature.

2.2.1 Standard linear model

We start with the most general linear model where the data is structured as a matrix A (that can be square or rectangular). The general form is

$$A\theta = E, \tag{69}$$

where A_{ij} encodes the j th feature component of the i th data point (molecule), θ is the coefficients to be found, and E_i is the target energy value of the i th data point.

The linear system of equations (69) can be solved by row-reduction techniques for any general matrix A . Conjugate gradient linear least-squares (CGLS) [36, 37] method can be used to solve the system in the form of least squares

$$\min_{\theta} \|A\theta - E\|_2^2,$$

usually this is preferred when the system is large or early stopping is required to avoid overfitting.

If A has special structures, for example, if it is a symmetric positive definite matrix, then we can use Cholesky factorization that gives

$$A = LL^T,$$

where L is a lower triangular matrix, then the system becomes

$$LL^T\theta = E,$$

first, we solve for $y := L^T\theta$ by forward substitution:

$$Ly = E,$$

then we can obtain θ by back substitution:

$$L^T\theta = y.$$

If A is an overdetermined system, then we can use the normal equation

$$A^T A \theta = A^T E.$$

2.2.2 Kernel models

One popular model for chemical quantities' prediction is the **kernel** model. Due to its robustness, useful properties, and compact representation of the model over the encoding of the data.

The **representer theorem** states that a minimizer of an optimization problem can be represented by an expansion of kernel on the input data points.

Theorem (Theorem 9 in [38]). *Given:*

- *a positive definite kernel $k : X \times X \mapsto \mathbb{R}$, the corresponding reproducing kernel Hilbert space H , and the data points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \in X \times \mathbb{R}$;*
- *an arbitrary cost function $c : (X \times \mathbb{R}^2)^n \mapsto \mathbb{R} \cup \{\infty\}$,*
- *a strictly monotonically increasing function $g : [0, \infty) \mapsto \mathbb{R}$,*

then each minimizer $f \in H$ of the regularized risk functional

$$c((x_1, y_1, f(x_1)), (x_2, y_2, f(x_2)), \dots, (x_n, y_n, f(x_n))) + g(\|f\|_H^2)$$

admits a representation of the form:

$$f(x) = \sum_{i=1}^n \theta_i k(x, x_i)$$

where $\theta_i \in \mathbb{R}$.

For the kernel application, given a pair of feature vectors u, v , a kernel function $\kappa(u, v)$ maps the pair into a scalar. Two types of kernel functions are used, the Gaussian kernel:

$$\kappa(u, v) = \exp\left(-\frac{\|u - v\|_2^2}{2\sigma^2}\right), \quad (70)$$

where the width of the Gaussian σ is a hyperparameter; and the linear kernel:

$$\kappa(u, v) = u^T v. \quad (71)$$

The kernel function then forms a kernel matrix $K = (K_{ij})$ which contains in each row molecule's condensed information. Let F be a matrix where each row is a molecular feature vector, then the entries of the kernel matrix are

$$K_{ij} := \kappa(F_i, F_j). \quad (72)$$

Typically, the set of features are available on the atomic level, in this case, the entries are

$$K_{ij} := \sum_l \sum_{l'} \delta_{Z_l^i Z_{l'}^j} \kappa(f_{l:}^i, f_{l':}^j), \quad (73)$$

where Z_l^i and $Z_{l'}^j$ are the nuclear charges of atom l and l' in molecule i and molecule j respectively; $f_{l:}^i$ and $f_{l':}^j$ are row vectors containing atomic features of an atom indexed by l in a molecule indexed by i and features of an atom indexed by l' in a molecule indexed by j . The Kronecker delta $\delta_{Z_l^i Z_{l'}^j}$ gives the "elemental screening" effect, i.e., it makes sure that only the same atomic environments are compared, which leads to faster computation speed and higher accuracy [17].

Given E and K formed from the training set, the regression coefficients θ can be found by a closed form solution if K is invertible,

$$\theta = (K + \lambda I)^{-1} E, \quad (74)$$

where λ is a regularization constant. Typically in practice the solution is computed by row reduction techniques or by numerical linear solvers instead of explicitly inverting K .

Given θ and K , we can predict the energy of a molecule indexed by i by

$$E_i = \sum_j K_{ij} \theta_j, \quad (75)$$

where j runs over the indices of molecules in the training set.

2.2.3 Neural network (NN)

Neural network (NN) has been known as the universal approximator. The **universal approximation theorems** can be summarized as a statement that says, for each function f from some function space, f can be approximated by a sequence of neural networks $\phi_1, \phi_2, \dots$, such that $\phi_n \rightarrow f$. The theorems only state existence of such sequence of neural networks, they are not constructive on how to choose the depth of hidden layers, the numbers of nodes on the layers, etc. In general, sufficiently large network approximates f well enough. In regards to the convergence, NNs have multiple local minima, therefore depending on the optimization algorithm, there is no guarantee to reach only a single solution. However, for real problems it is not preferred to obtain the global minima of the NN with respect to the cost function

constructed by the training data, since the modelling with NN leans more towards model generalization capability given unseen data.

For the k th layer output z_k , vanilla NNs have the general recursive form of

$$\begin{aligned} z_0 &= x, \\ z_i &= \sigma_i(W_i z_{i-1} + b_i), \quad \text{for } i = 1, 2, \dots, n \end{aligned} \quad (76)$$

where $x \in \mathbb{R}^{n_f}$ is the input feature vector that corresponds to a molecule; $W_k \in \mathbb{R}^{n_k \times n_f}$ is a weight matrix of the k th hidden layer; $b_k \in \mathbb{R}^{n_k}$ is the bias vector of the k th hidden layer; σ_k is the nonlinear activation function, such as sigmoid, relu, tanh, etc; and n_k is the number of nodes within layer k .

One of the most quoted universal approximation theorems is in the form of arbitrary width (number of neurons) case theorem in [39]. The theorem states that a finite sum of sigmoidal functions can approximate any functions in the space $C(I_n)$ of continuous functions on I_n with supremum norm; where I_n is n -dimensional unit cube $[0, 1]^n$.

Theorem (Theorem 1 in [39]). *Let σ be a sigmoidal function. Then finite sums of the form*

$$g(x) = \sum_{i=1}^N \theta_i \sigma(w_i^T x + b_i), \quad (77)$$

are dense in $C(I_n)$. Where $w_i \in \mathbb{R}^n, \theta_i, b_i \in \mathbb{R}$. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum, $g(x)$ of the above form, such that

$$|g(x) - f(x)| < \epsilon, \quad \forall x \in I_n.$$

The characteristics of NNs should be applicable for potential energy surface approximations as well. In the context of potential energy surface approximations, given set of features that are invariant to rotation, translation, and atomic permutation of the same type, NNs would be able to learn the nonlinear relationships between the molecular geometry and the energy.

Variations of NN for energy prediction exist in many literature, one of the earliest major breakthrough for potential energy surface is the usage of shallow NN to capture individual atomic energy E_i by [10], the structure of the network for each atom i is shown in Figure 6. Then the total energy for a molecular system is summed $E = \sum_i E_i$.

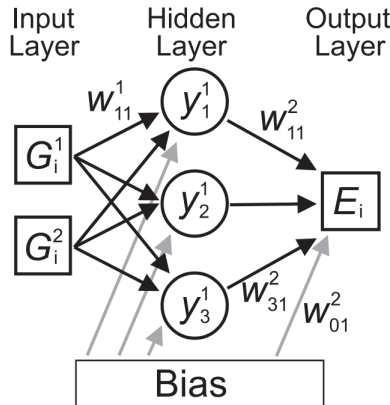


Figure 6: The subnets for atom i for ACSF [10], where G_i^k is a set of symmetry features generated from the k th function of ACSF, y_l^k corresponds to the l th node in the k th layer.

Other types of NN for potential energy surfaces — or more generally for quantum chemical properties’ predictions exist — for example, the message passing scheme in Section 2.1.6 is in fact a part of the message passing neural network (MPNN).

Commonly NNs are trained by minimizing the cost function with respect to the collection of the coefficients (weight matrices) to be optimized $\theta := [W_1, b_1, W_2, b_2, \dots, W_n, b_n]$ given the dataset X indexed by training indices I_{train} . The cost function is mean squared error (MSE):

$$f(\theta) = \frac{1}{|I_{\text{train}}|} \sum_{i \in I_{\text{train}}} \left(z_n(\theta, X_i) - E_i \right)^2. \quad (78)$$

NNs are trained by using the stochastic gradient descent (SGD) algorithm variants, such as ADAM [40]. In the generic form, SGD tries to solve the problem of

$$\min_{\theta} f(\theta) := \langle f(\theta|E) \rangle, \quad (79)$$

where $\langle \cdot \rangle$ denotes the expectation. This says that the expectation of the cost function is the minimum the posed problem. SGD minimizes the cost function by

$$\theta_{i+1} := \theta_i - \gamma_i g_i(\theta_i), \quad (80)$$

where γ_i is the learning rate or step size that satisfies

$$\sum_i \gamma_i^2 < \infty, \quad \sum_i \gamma_i = \infty,$$

and the gradient approximation is

$$g_i(\theta) := \frac{1}{|B_i|} \sum_{l \in B_i} \nabla f(\theta|E_l), \quad (81)$$

computed from a minibatch B_i of the dataset in the i th iteration, where typically $|B_i| \ll |X|$. This gives efficient computation in terms of memory and timing especially if the size of the data for training is large. The gradients are computed by the **backpropagation** algorithm — a special case of **reverse mode automatic differentiation**. Automatic differentiation (AD) is an interesting topic to explore by itself too, however in this section we shall limit the discussion to only the relevant machine learning models and ways to solve them.

2.3 Hyperparameter optimization

In essence, hyperparameter optimization selects the best combinations of any or all components involved in the machine learning process. For example, the selection of the most relevant feature can be encapsulated as an optimization process. The techniques related to hyperparameter optimization is different to the usual optimization used for machine learning fitting; the function to be optimized is often assumed to be a **black box**, where it is difficult or impossible to obtain the derivatives of the function. Hence derivative-free techniques for the search space is usually used.

Generic machine learning pipeline for chemical data involves feature extraction, feature selection, data selection, and fitting. Additional blocks can be introduced to the pipeline, for example, pre-fitting using lower resolution molecular features to reduce the total energy for the actual fitting process. In this thesis, the machine

learning pipeline augmented with hyperparameter optimization is depicted in Figure 7 (especially relevant for Chapters 6 and 7).

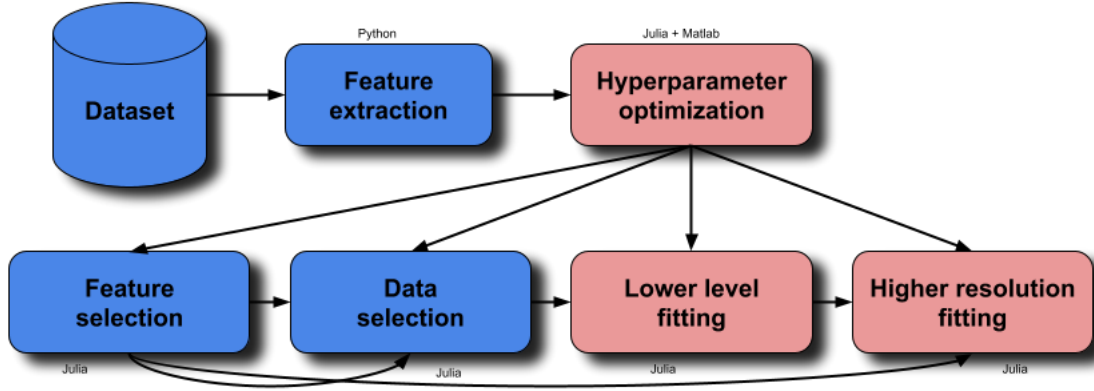


Figure 7: Main processes affected by the hyperparameter optimization, tagged with the programming language involved.

The hyperparameter optimization problem can be formulated as a bound-constrained black-box mixed integer optimization problem

$$\begin{aligned} \min_x & f(x) \\ \text{s.t. } & x_i \in [\underline{x}_i, \overline{x}_i]. \end{aligned}$$

By mixed integer, the variables are not just limited to integer and real values, they can also include special type variables such as **categorical** variables and **ordered integer** variables (both will be explained in Section 2.3.1). It is also possible that the variables are the indices of the equation, for example, if we want to select the best combination of training data points that minimize fitting errors. An example of the hyperparameter optimization problem, suppose that we want to do a regression fitting of energy values given the molecular geometries after transformed into molecular features, and we have a bunch of selections such as the type of molecular features, the size of molecular features, etc; in which we try to find the most optimal combinations of model's components, we can encode the possible combinations as:

- $x_1 \in \{\text{ACSF}, \text{SOAP}, \text{MBDF}\}$, a categorical variable, corresponds to the selection of molecular feature types.
- $x_2 \in \{5, 6, \dots, 20\}$, an integer variable, corresponds to the size of the molecular features.
- $x_3 \in [0, 10]$, a real variable, corresponds to the value of σ in (70).

The hyperparameter optimization process can be decomposed into three main components, the **simulator**, **solver**, and **main interface**. The simulator computes the objective function value $f := f(x)$ given the iterate x passed by the solver. The solver receives a feedback in the form of the objective function value f and then decides the next iterate x based on the feedback (or any additional information). Often times an interface between the simulator and the solver is needed, especially if the computing environments are different from each other, for example, different programming languages being used (such as in this thesis). The interface also works as the translator of the data flow between the simulator and the solver, especially

if some components of the decision variables are not real or integer types, then the translation functions should be placed inside the interface. Overall this scheme gives more freedom and is environment agnostic therefore we could design the simulator and the solver more flexibly; for example, parallelization of the simulators can be simpler to design since we can put the data traffic manager within the main interface (although the implementation remains nontrivial).

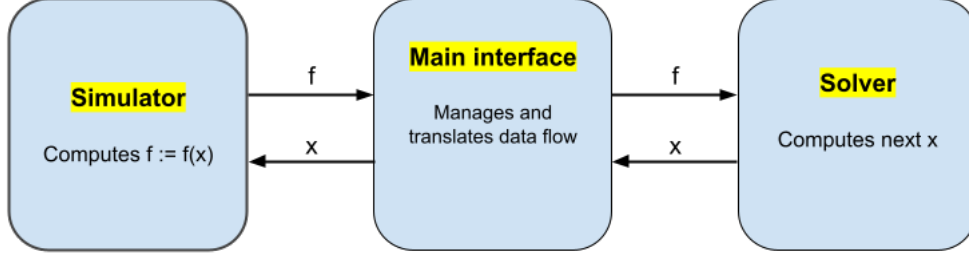


Figure 8: Data flow between simulator and solver.

2.3.1 Handling special-typed variables

Typically variables with **real** and **integral** types can be easily passed into an optimization routine given the bounds and types, as most optimization software handles both as mandatory types (e.g. in mixed-integer solvers). This is not usually the case for types such as **categorical** and **ordered integers**, hence special treatments are needed.

2.3.1.1 Treating categorical variables Categorical variable is a variable that does not encode ordering relationship between the values, unlike real or integer values. For example, suppose we want to select an optimization algorithm from a list {SGD, BFGS, Levenberg–Marquardt} then it is not possible to order the elements of the list in any way.

Given a variable with **categorical** type, we can encode it as a one-hot vector. The most general way, i.e., a way of passing the variable such that the variable can be handled by most optimization routines is by setting the one-hot vector type as **real**. A mechanism to handle the categorical variable, which is called "round to simplex", says: given $x \in \mathbb{R}^n$, find p such that

$$\begin{aligned} & \min \|p - x\|_2^2 \\ & \text{s.t } p \geq 0, \quad e^\top p = 1, \end{aligned} \tag{82}$$

where e is an all-1 vector. In other words, we want to find a "probability" vector p that is closest to x , here x can be an output of the optimization solver or an arbitrary initial point which corresponds to the one-hot vector. (82) can be solved analytically in order to find p . By considering the first order optimality condition of (82), which gives

$$\inf(p - x - \lambda e, p) = 0,$$

this implies

$$p_i = p_i(\lambda) := (x_i - \lambda)_+,$$

where λ is such that

$$S(\lambda) := e^\top p(\lambda) = \sum_i (x_i - \lambda)_+ = 1,$$

where $S(\lambda)$ is piecewise linear with breakpoints at x_i . From here, in order to solve $S(\lambda)$ for λ , we do the following steps:

- Let $z = \text{sort}(\bar{x} - x)$, where $\bar{x} = \max_i(x_i)$, now we have: $0 = z_1 \leq z_2 \leq \dots \leq z_n$.
- Let $\hat{x}_i = \bar{x} - z_i$, and $\mu = \bar{x} - \lambda$, then

$$S(\lambda) = \sum_i (\hat{x}_i - \lambda)_+ = \sum_i (\bar{x} - \lambda - z_i)_+ = \sum_i (\mu - z_i)_+ = \sum_{i=1}^k (\mu - z_i)_+ = k\mu - S_k,$$

if $z_k \leq \mu \leq z_{k+1}$, where $z_0 := -\infty, z_{k+1} := \infty$.

- Since $k = 0$ gives $\mu \leq 0$ and $S(\lambda) = 0$, thus we need to find $k > 0$ in order to satisfy the $S(\lambda) = 1$:

$$k\mu - S_k = 1,$$

$$\mu = \frac{1 + S_k}{k}, \text{ where } k \text{ is the smallest with } 1 \leq kz_{k+1} - S_k$$

- Given the smallest possible k , finally we can obtain feasible p

$$\lambda = \bar{x} - \mu,$$

$$p_i = (x_i - \lambda)_+.$$

Using the feasible p from the steps above, we can pick the value y of the categorical variable by

$$q_i = \sum_{j \leq i} p_j, \text{ for } i = 0, 1, 2, \dots$$

where $q_0 = 0$. We generate a random value $r \in \mathbb{R} \cap [0, 1]$, then we pick $y := i$ which satisfies

$$q_i \leq r \leq q_{i+1}.$$

2.3.1.2 Treating ordered integer variables Ordered integer variables are integers that are connected with each others by a relationship or constraint. For example, the size of the molecular features should be less than or equal the size of the atomic features since molecular features are derived from the atomic features, this can be encoded by $x_m \leq x_a$. Or a more complicated relationship, such as the size of the nodes in a hidden layer is only greater than 0 if the corresponding hidden layer of the neural network exists.

Analogous to the categorical variables, we encode n ordered integers into an arbitrary vector $p \in \mathbb{R}^{n+1}$. It is allowed for either the initial point or the output of the optimizer to yield infeasible p , i.e., $e^\top p \neq 1$, thus we need to apply the "round to simplex" procedure for p , then we can obtain the values of the n ordered integers by

$$x_i = x_{i-1} + p_i dx, \text{ for } i = 0, 1, \dots, n$$

where

$$x_0 := \underline{x}, \quad x_{n+1} := \bar{x}, \quad dx = \bar{x} - \underline{x}. \quad (83)$$

On the opposite direction, suppose x is known,

$$\underline{x} = x_0 \leq x_1 \leq \dots \leq x_n \leq x_{n+1} = \bar{x},$$

and we want to compute p instead, then we encode them into a "probability" vector of size $n + 1$ by

$$p_i = \frac{x_i - x_{i-1}}{dx}.$$

3 Functional forms of potential energy surfaces

In this section, we discuss (largely traditional) functional forms of potential energy surfaces collected throughout the literature. The collected functional forms give insight on the fundamentals of how to design physical-machine learning models to capture the patterns of potential energy surfaces.

3.1 Permutational-invariant PES (BRAAMS & BOWMAN 2009)

Historically, the invariance property of the PES was first stated by MURRELL et al [41], where it was noted that an invariant fitting basis with this can be represented by an integrity basis. In general, any method that incorporates the invariance property into the representation of the PES must express it in the terms of a set of variables that is closed under all permutations, e.g., Cartesian coordinates, internuclear distances, etc. The approach described by BRAAMS & BOWMAN [42] uses monomial symmetrization alongside the invariant polynomial theory, where invariant bases are obtained in terms of **primary** and **secondary polynomials**.

The example of the monomial symmetrization application on acetylene (C_2H_2) by ZOU et al [43] will lead into the general form of the monomial symmetrization later, the multinomials expansion for the potential is

$$V = \sum_{\substack{a,b,c,d,e,f \\ a+b+c+d+e+f \leq M}}^M C_{abcdef} \left[y_{12}^a y_{13}^b y_{14}^c y_{23}^d y_{24}^e y_{34}^f \right], \quad (84)$$

where y_{ij} is the **Morse variable** with the following form

$$y_{ij} = \exp(-r_{ij}/r_0), \quad (85)$$

where H atoms are labelled 1, 2, and the C atoms are labelled 3, 4; r_0 is a chosen parameter. The summation over all powers of each y is constrained by the total degree M . The Morse variable indices (consequently the internuclear distances r_{ij}) are written in lexical order $i < j$; N by convention is the number of atoms, which in this case is four. Another possible form of (85) is $y_{ij} = 1 - \exp(-r_{ij}/r_0)$.

As the number of variables in V increases, the number of terms grows nonlinearly for a fixed M . Monomials with identical coefficients could be grouped into a single polynomial given by the sum of monomials multiplied by one coefficient.

BRAAMS & BOWMAN [42] proceed with the following equation to replace Eq(84):

$$V = \sum_{\substack{a,b,c,d,e,f \\ a+b+c+d+e+f \leq M}}^M D_{abcdef} S \left[y_{12}^a y_{13}^b y_{14}^c y_{23}^d y_{24}^e y_{34}^f \right], \quad (86)$$

where S denotes the symmetrization of the monomials. To symmetrize the monomials, the mapping of atom permutations to permutations of the internuclear distances is needed. For example, for A_3 molecules, let 1, 2, 3 denote the initial arrangement of the atoms; consider the permutation: 2, 3, 1. The initial internuclear distances r_{12}, r_{13}, r_{23} are mapped into r_{23}, r_{21}, r_{31} , which is then rewritten into the lexical order of r_{23}, r_{12}, r_{13} . Thus the monomial $r_{12}^a r_{13}^b r_{23}^c$ is mapped into $r_{23}^a r_{12}^b r_{13}^c$ or equivalent to the permuted powers $r_{12}^b r_{13}^c r_{23}^a$. The addition of four other monomials corresponding to the four additional permutations gives a fully symmetrized sum of monomials. Consequently y_{ij} is also permuted analogously, which yields the symmetrized basis.

It is now clear that only one term and one unknown coefficient D_{abcdef} appears for every set of tuples (a, b, c, d, e, f) which are permutationally equivalent. The details of the symmetrization on larger and more complex molecule formations are such as A_2B , A_3 , A_2B_2 , etc, are provided in Table 1-5 of BRAAMS & BOWMAN [42].

3.2 CHIPR (ROCHA & VARANDAS 2021)

The **Combined-Hyperbolic-Inverse-Power-Representation (CHIPR)** was first developed by VARANDAS [44] and assessed for triatomic molecules H_3 and H_2O . It was recently updated by ROCHA & VARANDAS [16] for the PES up to tetratomic molecule. In general, for simpler molecules, CHIPR is directly used as the potential energy value; for more complex molecules, for example molecules with multiple states (excited states and ground state), CHIPR acts as the components of each element of a matrix where each eigenvalue of the matrix represents one potential energy value on a particular state of the molecule.

3.2.1 General representation

In CHIPR, The molecular potential energy is represented as the summation of n energy terms:

$$V(\mathbf{r}) = \sum_{i=1}^n \mathcal{V}_S^{(i)}(\mathbf{r}), \quad (87)$$

where $\mathbf{r} = (r_{12}, r_{13}, \dots)$ is the list of all interatomic distances, and the sub-index S denotes that the functional must reflect the appropriate permutational symmetry. $\mathcal{V}_S^{(1)}$ represents the one-body term (within this particular framework, it is set to 0),

$$\mathcal{V}_S^{(2)}(\mathbf{r}) = \sum_{i < j} V_{ij}^{(2)}(r_{ij}) \quad (88)$$

is the sum of two-body potentials, the sum of the three-body potentials is

$$\mathcal{V}_S^{(3)}(\mathbf{r}) = \sum_{i < j < k} V_{ijk}^{(3)}(r_{ij}, r_{ik}, r_{jk}), \quad (89)$$

$\mathcal{V}_S^{(4)}(\mathbf{r})$ represents the four-body interaction, etc. For example, in the case of tetratomic molecules, the assumed many-body expansion form of the CHIPR method is

$$V(\mathbf{r}) = \mathcal{V}_S^{(1)} + \mathcal{V}_S^{(2)}(\mathbf{r}) + \mathcal{V}_S^{(3)}(\mathbf{r}) + \mathcal{V}_S^{(4)}(\mathbf{r}), \quad (90)$$

The general form of the CHIPR n -body term is

$$V^{(n)}(\mathbf{r}) = \sum_{i_1=0, \dots, i_\tau=0}^L C_{i_1, \dots, i_\tau} \prod_{p=1}^{\tau} y_p^{i_p}, \quad (91)$$

where $C_{i_1, \dots, i_\tau}$ are the expansion coefficients (Table I by VARANDAS [44] provides the example of the index values of the expansion coefficients), y_p ($p = 1, 2, \dots, \tau$) is the set of coordinates relative to some reference geometry, where

$$\tau = n(n-1)/2 \quad (92)$$

is the total number of internal degrees of freedom. Every y_p in (91) is expanded as

$$y_p = \sum_{\alpha=1}^M c_{\alpha} \phi_{p,\alpha}, \quad (93)$$

where c_α are contraction coefficients to be determined from fitting, with α as the index of each primitive function $\phi_{p,\alpha}$. The primitive function $\phi_{p,\alpha}$ has the form of

$$\phi_{p,\alpha} = \begin{cases} \text{sech}^{\eta_\alpha}(\gamma_{p,\alpha}\rho_{p,\alpha}), & \text{if long-range terms are ignored,} \\ \left(\frac{\tanh(\beta_\alpha R_p)}{R_p}\right)^{\sigma_\alpha} \text{sech}^{\eta_\alpha}(\gamma_{p,\alpha}\rho_{p,\alpha}), & \text{otherwise,} \end{cases} \quad (94)$$

where R_k , $k \in \mathbb{Z}^+$, ($R_1 := r_{12}$, $R_2 := r_{13}$, $R_3 := r_{23}$, ...) is introduced as the atomic distance notation to simplify the indices' enumeration; the deviation coordinate is expressed as

$$\rho_{p,\alpha} = R_p - R_{p,\alpha}^{\text{ref}}, \quad (95)$$

$\gamma_{p,\alpha}$ represents the nonlinear parameters, and the appearing constants are

$$\begin{aligned} \eta_\alpha &\equiv \eta = 1, \\ \sigma_\alpha &\equiv \sigma = 6, \\ \beta_\alpha &\equiv \beta = 1/5. \end{aligned} \quad (96)$$

To avoid strong linear dependencies, the reference geometries of the various basis functions are related by

$$R_{p,\alpha}^{\text{ref}} = \zeta(R_p^{\text{ref}})^{\alpha-1}, \quad (97)$$

where ζ and $R_{p,0}^{\text{ref}}$ are constants. Table II by VARANDAS [44] provides the example of the coefficients' values for ground state OH and O₂.

3.2.2 One-body term

The form of the one-body term is

$$\mathcal{V}_S^{(1)} = \sum_i V_i^{(1)}, \quad (98)$$

$\mathcal{V}_S^{(1)}$ is zero for the atomic ground state and positive for the atomic excited state. In the models for H_xO_y, the relevant states for the one-body term are

$$\begin{aligned} &\text{H}(^2S), \\ &\text{O}(^1D), \text{O}(^3P), \text{O}^+(^4S). \end{aligned} \quad (99)$$

3.2.3 Two-body terms

The functional form of two-body terms (diatomic molecule) is

$$V^{(2)}(r_{ij}) = \frac{Z_i Z_j}{r_{ij}} \sum_{l=1}^L C_l y^l, \quad (100)$$

where Z_i and Z_k are the nuclear charges of atoms i and k ; the appearing y is the summation factor described in (93). The relevant states for H_xO_y are

$$\begin{aligned} &\text{H}_2(X^1\Sigma_g^+), \\ &\text{O}_2(^1\Delta_g), \text{O}_2(X^3\Sigma_g^-), \\ &\text{OH}(X^2\Pi), \text{OH}^+(X^3\Sigma^-). \end{aligned} \quad (101)$$

3.2.4 Three-body terms

The three-body CHIPR function of (91) has the form of

$$V^{(3)}(\mathbf{r}) = \mathcal{D}^{(3)} \sum_{i,j,k=0}^L C_{i,j,k} \left\{ \sum_{g \in G} \mathcal{P}_g^{(i,j,k)} [y_1^i y_2^j y_3^k] \right\}, \quad (102)$$

where $C_{i,j,k}$ are the expansion coefficients subject to several constraints depending on the molecule's species, y_p ($p = 1, 2, 3$) are the coordinates from (93), g is the permutation element, and G is a subgroup of the $\mathcal{S}_3$ symmetric group; therefore, $\mathcal{P}_g^{(i,j,k)}$ are the operators that reflect the action of the atom permutation g on y_p . Table I in ROCHA & VARANDAS [16] shows the expansion coefficients' constraints and the permutation elements. The additional factor which serves to dampen the three-body terms is the case $n = 3$ of

$$\mathcal{D}^{(n)}(\mathbf{R}) = \left[\prod_{i=1}^{\tau} h(R_i) \right]^{\xi}, \quad (103)$$

where $\mathbf{R} = \{R_1 := r_{12}, R_2 := r_{13}, R_3 := r_{23}, \dots\}$, and $h(R_i)$ is expressed as

$$h(R_i) = (1 + \tanh(\kappa(R_i - R_0)))/2, \quad (104)$$

with the chosen parameters are

$$\begin{aligned} R_0 &= 0.5a_0, \\ \kappa &= 100a_0^{-1}, \\ \xi &= 10. \end{aligned} \quad (105)$$

The relevant states for H_xO_y are

$$\begin{aligned} &\text{H}_2\text{O}(X^1A_1), \\ &\text{HO}_2(X^2A''), \\ &\text{HO}_2^+(X^3A''), \text{HO}_2^+(1^3A''). \end{aligned} \quad (106)$$

3.2.5 Four-body terms

For general tetratomic molecule, the four-body CHIPR function is

$$V^{(4)}(\mathbf{r}) = \mathcal{D}^{(4)} \sum_{i,j,k,l,m,n=0}^L C_{i,j,k,l,m,n} \left\{ \sum_{g \in G} \mathcal{P}_g^{(i,j,k,l,m,n)} [y_1^i y_2^j y_3^k y_4^l y_5^m y_6^n] \right\}, \quad (107)$$

similar to the three-body form, the constraints which depend on the molecular species, are provided in the Table II by ROCHA & VARANDAS [16]. The damping-term $\mathcal{D}^{(4)}$ is also analogous to the three-body's version, with exactly the same quantities to (103), except the product's end index is now 6. The relevant states of H_xO_y for four-body terms are

$$\text{H}_2\text{O}_2(X^1A). \quad (108)$$

In the tetratomic molecule case, in some instances, such as in COELHO & BRANDÃO [45] the **reference geometries** are used to better fit the curve around

the regions of interest, such as the minima, transition state configurations, and other regions which play important role in the molecular dynamics. The form of the reference geometry is

$$\sum_{rg} P(\mathbf{R}) \Theta_{rg}(\mathbf{R}) \quad (109)$$

where the sum is over all reference geometries rg ; $P(\mathbf{R})$ is a polynomial function which is expanded by using the reference geometries as the centres, shown as

$$P(\mathbf{R}) = \sum_{k=0}^M C_k (\mathbf{R} - \mathbf{R}_{rg})^k, \quad (110)$$

where C_k is the fitted coefficient and $\mathbf{R}_{rg}$ is the reference geometry;

$$\Theta(\mathbf{R}) = \prod_{i=1}^d \exp\left(\frac{\eta_i v_i - v_i^2}{8\lambda_i}\right) \quad (111)$$

is an exponential range factor term;

$$v_i = (\vec{\psi} \cdot \vec{u}_i) = \sum_{j=1}^d \psi_j \times u_{ij}, \quad (112)$$

where the appearing $\vec{u}_i$ is the eigenvector, λ_i is the eigenvalue, η_i is the eigenvector direction, ψ is the displacement integrity factor, and d is the total number of the displacement integrity basis (the complete form and explicit values are shown in Table 2 and (17) of COELHO & BRANDÃO [45]).

3.3 CHIPR for HO_2^+ (XAVIER et al 2019)

The explicit usage of CHIPR from Subsection 3.2 is demonstrated by XAVIER et al [46] for HO_2^+ . The functional form which represents the total interaction potential is

$$V(\mathbf{r}) = V_{\text{O}_2^+}^{(2)}(r_{23}) + V_{\text{OH}^+}^{(2)}(r_{12}) + V_{\text{OH}^+}^{(2)}(r_{13}) + V_{\text{HO}_2^+}^{(3)}(r_{23}, r_{12}, r_{13}), \quad (113)$$

where r_{23} is the distance between oxygen atoms, and r_{12} and r_{13} are the distances between hydrogen and oxygen atom. The diatomic potential's explicit form is the realization of (100) shown as

$$V^{(2)}(r_{ij}) = \frac{Z_i Z_j}{r_{ij}} (C_1 y_1 + C_2 y_1^2 + C_3 y_1^3), \quad (114)$$

where N_i and N_j are the nuclear charges of atom i and j respectively, C_1, C_2, C_3 are parameters to be fitted. According to (91), with $L = 3$, without the damping term, the explicit form of the three-body interaction energy is

$$\begin{aligned} V^{(3)}(\mathbf{r}) = & C_{110} y_1 (y_2 + y_3) + 2C_{011} y_2 y_3 + 2C_{111} y_1 y_2 y_3 \\ & + C_{120} y_1 (y_2^2 + y_3^2) + C_{210} y_1^2 (y_2 + y_3) + C_{012} (y_2 y_3^2 + y_2^2 y_3^2), \end{aligned} \quad (115)$$

where C_{i_1, i_2, i_3} are the coefficients to be fitted. It can be noted that since $C_{110} = C_{101}$ (permutational symmetry), only one is chosen, analogous argument applies to terms with higher y powers.

The appearing y factors are the contracted basis with general form described in (93). The y_i is defined as

$$y_i = \begin{cases} \begin{cases} \sum_{k=1}^4 c_{1k} \operatorname{sech}^{\eta_{1k}}(\gamma_{1k}([R_1 - \zeta_1(R_1^{\text{ref}})^{k-1}])) & \text{for } V_{\text{OH}^+}^{(2)}, \\ \sum_{k=1}^4 c_{1k}(s_k R_1 + t_k) \operatorname{sech}^{\eta_{1k}}(\gamma_{1k}([R_1 - \zeta_1(R_1^{\text{ref}})^{k-1}])) & \text{for } V_{\text{O}_2^+}^{(2)}, \\ \sum_{k=1}^3 c_{1k}(R_1^{\text{ref}})^{1-k}((\tanh[\beta_1(R_1 - \zeta_1(R_1^{\text{ref}})^{k-1}])) \\ \times \operatorname{sech}^{\eta_{1k}}(\gamma_{1k}[R_1 - \zeta_1(R_1^{\text{ref}})^{k-1}])) & \text{for } V_{\text{HO}_2^+}^{(3)}, \end{cases} & \text{if } i = 1, \\ \begin{cases} \sum_{k=1}^3 c_{ik}((\tanh[\beta_i(R_i - \zeta_i(R_i^{\text{ref}})^{k-1}])) \\ \times \operatorname{sech}^{\eta_{ik}}(\gamma_{ik}[R_i - \zeta_i(R_i^{\text{ref}})^{k-1}])) \times (R_i^{\text{ref}})^{1-k} & \text{for all } V^{(2)} \text{ and } V^{(3)}, \end{cases} & \text{for } i = 2, 3 \end{cases} \quad (116)$$

where $R_1 := r_{23}$, $R_2 := r_{12}$, $R_3 := r_{13}$; The coefficients c_{ik} , γ_{ik} , β_i , η_{ik} , ζ_i , s_k , t_k are obtained from fitting. The complete coefficients' values found are provided in the supplementary materials of XAVIER et al [46].

3.4 CHIPR fitting

In CHIPR, generally the fitting is done by utilizing the weighted least-square method. The objective function for diatomic energy to be minimized is

$$\chi^2 = \sum_{h=1}^N \left\{ W_h [V^{(2)}(R_h; C_k, c_\alpha, R^{\text{ref}}, \zeta) - E(R_h)] \right\}^2, \quad (117)$$

where N is the total *ab initio* points, W_h is the least-squares weight of the h th calculated point at R_h , and $E(R_h)$ is the corresponding *ab initio* energy.

The three-body (triatomic molecules) energy for fitting, $\epsilon^{(3)}$, is determined by

$$\epsilon^{(3)}(\mathbf{r}) = E(\mathbf{r}) - \mathcal{V}_S^{(2)}(\mathbf{r}) - \mathcal{V}_S^{(1)}. \quad (118)$$

In order to obtain the optimum basis set parameters y_p , first, the objective function to be minimized is

$$\chi_p^2 = \sum_{h=1}^{N_{\text{basis}}} \left\{ W_h \left[y_p(R_p; c_\alpha, R_p^{\text{ref}}, \zeta) - \epsilon_h^{(3)} \right] \right\}^2, \quad (119)$$

where $p = \{1\}, \{1, 2\}, \{1, 2, 3\}$ for A_3 , AB_2 , and ABC molecular species respectively. After obtaining the optimum basis set parameters, the next objective function to be minimized is

$$\chi^2 = \sum_{h=1}^{N_{\text{pol}}} \left\{ W_h \left[V^{(3)}(\mathbf{R}; C_I, c_\alpha, R_p^{\text{ref}}, \zeta) \times \mathcal{D}^{(3)} - \epsilon_h^{(3)} \right] \right\}^2, \quad (120)$$

where $I = \{i, j, k\}$ set of indices.

Analogous to the triatomic energy, the tetratomic energy is obtained from the subtraction of the total energy with the ($n < 4$)-body terms, its form is

$$\begin{aligned}\epsilon^{(4)}(\mathbf{r}) &= E(\mathbf{r}) - \mathcal{V}_S^{(2+3)}(\mathbf{r}) \\ &= E(\mathbf{r}) - \left[\sum_{i < j} V_{ij}^{(2)}(r_{ij}) + \sum_{i < j < k} V_{ijk}^{(3)}(r_{ik}, r_{jk}, r_{ij}) \right].\end{aligned}\quad (121)$$

The objective function for calibrating y_p follows from (119), with $I = \{i, j, k, l, m, n\}$; $\epsilon^{(4)}(\mathbf{R})$ replaces $\epsilon^{(3)}(\mathbf{R})$ and $p = \{1\}, \{1, 4\}, \{1, 3, 6\}, \{1, 2, 3, 6\}, \{1, 2, 3, 4, 5, 6\}$ for A_4, AB_3, A_2B_2, ABC_2 , and $ABCD$ molecular species respectively; as shown in (122) and (123) which are analogous to (119) and (120) respectively.

$$\chi_p^2 = \sum_{h=1}^{N_{\text{basis}}} \left\{ W_h \left[y_p(R_p; c_\alpha, R_p^{\text{ref}}, \zeta) - \epsilon_h^{(4)} \right] \right\}, \quad (122)$$

$$\chi^2 = \sum_{h=1}^{N_{\text{pol}}} \left\{ W_h \left[V^{(4)}(\mathbf{R}; C_I, c_\alpha, R_p^{\text{ref}}, \zeta) \times \mathcal{D}^{(4)} - \epsilon_h^{(4)} \right] \right\}^2. \quad (123)$$

3.5 2 x 2 diabatic matrix for HO_2^+ dissociation (XAVIER & VARANDAS 2021)

Employed by XAVIER & VARANDAS [47]. The dissociation scheme of HO_2^+ for the double-sheeted PES which will determine the many-body terms is

$$\text{HO}_2^+(X^3A'') \rightarrow \begin{cases} \text{O}_2^+(X^2\Pi_g) + \text{H}(^2S), \\ \text{OH}^+(X^3\Sigma^-) + \text{O}(^3P), \end{cases} \quad (124)$$

$$\text{HO}_2^+(X^3A'') \rightarrow \begin{cases} \text{O}_2(X^3\Sigma_g^-) + \text{H}^+, \\ \text{OH}^+(X^3\Sigma^-) + \text{O}(^1D), \end{cases} \quad (125)$$

$$\left. \begin{array}{l} \text{HO}_2^+(X^3A'') \\ \text{HO}_2^+(1^3A'') \end{array} \right\} \rightarrow \text{O}(^3P) + \text{O}^+(^4S) + \text{H}(^2S). \quad (126)$$

The function to model a cusp in the PES is the eigenvalue expression of a 2×2 diabatic matrix, shown as

$$V_d = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}, \quad (127)$$

where the main diagonal elements are the diabatic functions, while the off-diagonal elements $V_{12} = V_{21} \neq 0$ are coupling potentials. When V_d is diagonalized (numerically), it yields two solutions which are the eigenvalues (the adiabatic energies) in the adiabatic matrix ($V_{12} = V_{21} = 0$). The form of the adiabatic potentials is

$$V_{\pm} = \frac{V_{11} + V_{22}}{2} \pm \frac{1}{2} \sqrt{(V_{11} - V_{22})^2 + 4V_{12}^2}, \quad (128)$$

where $V_{\pm} = \{V_-, V_+\}$ is the set of eigenvalues of matrix described in (127), V_- is the ground state potential and V_+ is the excited state potential. The V_{ij} terms are formed from the basis of **Double Many Body Expansion (DMBE)** theory as the following:

$$V_{11} = V_{\text{O}_2}^{(2)}(X^3\Sigma_g^-; R_1) + V_{\text{OH}^+}^{(2)}(X^3\Sigma^-; R_2) + V_{\text{OH}^+}^{(2)}(X^3\Sigma^-; R_3) + V_{\text{HO}_2^+}^{(3)}(X^3A''; R_1, R_2, R_3), \quad (129)$$

$$V_{22} = V_{\text{O}_2^+}^{(2)}(X^2\Pi_g; R_1) + V_{\text{OH}^+}^{(2)}(X^3\Sigma^-; R_2) + V_{\text{OH}^+}^{(2)}(X^3\Sigma^-; R_3) + V_{\text{HO}_2^+}^{(3)}(1^3A''; R_1, R_2, R_3), \quad (130)$$

$$V_{12} = V_{21} = \left[\left(\frac{R_1^2 + R_3^2 - R_2^2}{2R_1R_3} \right)^2 - \left(\frac{R_1^2 + R_2^2 - R_3^2}{2R_1R_2} \right)^2 \right] S^2, \quad (131)$$

where V_{O_2} , $V_{\text{O}_2^+}$ and V_{OH^+} are two-body terms of diatomic potential modelled by CHIPR (explained in Subsection 3.2). Each term within the summation has the general form of $V(\text{state}; R_i)$, the appearing molecular states can be referred to the dissociation scheme shown in (124) and (126); R_i ($i = 1, 2, 3$) represents the atomic distances, where R_1 is the distance between two oxygens ($\text{O}^1 - \text{O}^2$), R_2 is the distance between the first oxygen and the hydrogen ($\text{O}^1 - \text{H}$), and R_3 is the distance of the hydrogen and the second oxygen ($\text{H} - \text{O}^2$). The three-body term ($V_{\text{HO}_2^+}$) when modelled by CHIPR, typically has the following form:

$$\begin{aligned} V_{\text{HO}_2^+}^{(3)} = & C_{110}y_1(y_2 + y_3) + 2C_{011}y_2y_3 + 2C_{111}y_1y_2y_3 \\ & + C_{120}y_1(y_2^2 + y_3^2) + C_{210}y_1^2(y_2 + y_3) \\ & + C_{012}(y_1y_3^2 + y_2^2y_3) + C_{121}y_1(y_1^2y_3 + y_2y_3^2) \\ & + \dots, \end{aligned} \quad (132)$$

where y_i , ($i = 1, 2, 3$) are the contracted basis; C_{jkl} are the coefficients obtained from fitting, in this case, a maximum of 10 for the overall power of the polynomial expansion is considered, which results in 34 coefficients; S is CHIPR properly decaying polynomial-type form.

3.6 3 x 3 diabatic matrix for H_2O_2 dissociation (COELHO & BRANDÃO 2017)

The PES functional in the form of a 3×3 symmetric matrix was employed by COELHO & BRANDÃO [45] for tetratomic dissociation of H_2O_2 . Consider the dissociation channels as follows:

$$\text{H}_2\text{O}_2(X, {}^1A) \rightarrow \begin{cases} \text{OH}(X^2\Pi) + \text{OH}(X^2\Pi) \\ \text{H}_2(X^1\Sigma_g^+) + \text{O}_2(a^1\Delta_g) \\ \text{O}(^1D) + \text{H}_2\text{O}(X^1A_1) \\ \text{H}(^2S) + \text{HO}_2(X^2A'') \\ 2\text{H}(^2S) + \text{O}_2(^3\Sigma_g^-) \\ \text{H}_2(X^1\Sigma_g^+) + 2\text{O}(^3P) \\ \text{H}(^2S) + \text{O}(^3P) + \text{OH}(X^2\Pi) \\ 2\text{H}(^2S) + 2\text{O}(^3P), \end{cases} \quad (133)$$

similar to Subsection (3.5), the channels will determine the summation for each term presents in the matrix. The 3×3 symmetric matrix has the following form:

$$V_l = V_l^T = \begin{bmatrix} V_{11} & V_{12} & 0 \\ V_{21} & V_{22} & 0 \\ 0 & 0 & V_{33} \end{bmatrix}, \quad (134)$$

$$V_{12} = V_{21}, \quad (135)$$

where only the lowest eigenvalue of V_l is considered. Simplifications are assumed where each diagonal term consists of only one, two, and three body terms. The diagonal terms has the following forms:

$$\begin{aligned}
V_{11} = & V_{\text{O}}^{(1)}(^1D) + V_{\text{O}}^{(1)}(^1D) + V_{\text{O}_2}^{(2)}(^3\Sigma_g^-; R_1) \\
& + V_{\text{H}_2}^{(2)}(^1\Sigma_g^+; R_2) + V_{\text{OH}}^{(2)}(^2\Sigma; R_3) \\
& + V_{\text{OH}}^{(2)}(^2\Sigma; R_4) + V_{\text{OH}}^{(2)}(^2\Sigma; R_5) + V_{\text{OH}}^{(2)}(^2\Sigma; R_6) \\
& + V_{\text{HO}_2}^{(3)}(^2A''; R_1, R_3, R_5) + V_{\text{HO}_2}^{(3)}(^2A''; R_1, R_4, R_6) \\
& + V_{\text{H}_2\text{O}}^{(3)}(^1A_1; R_2, R_3, R_6) + V_{\text{H}_2\text{O}}^{(3)}(^1A_1; R_2, R_4, R_5),
\end{aligned} \tag{136}$$

$$\begin{aligned}
V_{22} = & V_{\text{O}_2}^{(2)}(^3\Sigma_g^-; R_1) + V_{\text{H}_2}^{(2)}(^3\Sigma_u^+; R_2) \\
& + V_{\text{OH}}^{(2)}(^2\Pi; R_3) + V_{\text{OH}}^{(2)}(^2\Pi; R_4) + V_{\text{OH}}^{(2)}(^2\Pi; R_5) + V_{\text{OH}}^{(2)}(^2\Pi; R_6) \\
& + V_{\text{HO}_2}^{(3)}(^2A''; R_1, R_3, R_5) + V_{\text{HO}_2}^{(3)}(^2A''; R_1, R_4, R_6) \\
& + V_{\text{H}_2\text{O}}^{(3)}(^1A_1; R_1, R_4, R_6) + V_{\text{H}_2\text{O}}^{(3)}(^1A_1; R_2, R_4, R_5),
\end{aligned} \tag{137}$$

$$\begin{aligned}
V_{33} = & V_{\text{O}_2}^{(2)}(^1\Delta_g; R_1) + V_{\text{H}_2}^{(2)}(^1\Sigma_g^+; R_2) \\
& + V_{\text{OH}}^{(2)}(^2\Pi; R_3) + V_{\text{OH}}^{(2)}(^2\Pi; R_4) + V_{\text{OH}}^{(2)}(^2\Pi; R_5) + V_{\text{OH}}^{(2)}(^2\Pi; R_6) \\
& + V_{\text{HO}_2}^{(3)}(^2A''; R_1, R_3, R_5) + V_{\text{HO}_2}^{(3)}(^2A''; R_1, R_4, R_6) \\
& + V_{\text{H}_2\text{O}}^{(3)}(^3A'; R_1, R_4, R_6) + V_{\text{H}_2\text{O}}^{(3)}(^3A'; R_2, R_4, R_5).
\end{aligned} \tag{138}$$

The V_{12} term describes the diabatic crossings within the $\text{H}_2\text{O}(^1A_1)$ molecule is given by

$$V_{12} = V_{\text{H}_2\text{O}}^{(3)}(^1A_1; R_2, R_3, R_6) + V_{\text{H}_2\text{O}}^{(3)}(^1A_1; R_2, R_4, R_5). \tag{139}$$

Each term within the summation for each V_{ij} is possible to be analogous to Subsection (3.5), even though originally in COELHO & BRANDÃO [45] the authors refer to older methods, particularly: $\text{H}_2\text{O}(^1A_1)$ by BRANDÃO & RIO [48] (Subsection 3.7), $\text{H}_2\text{O}(^3A')$ by BRANDÃO et al [49] (Subsection 3.7), and $\text{HO}_2(^2A'')$ by BRANDÃO et al [50] (Subsection 3.8).

A total of 21 reference geometries (Table 3 of COELHO & BRANDÃO [45]) are used by (95) to indicate the deviation of the coordinates; several of the reference geometries used are 8 stationary points (Table 1 of COELHO & BRANDÃO [45]); two polynomial centres are used to warrant a good description of the insertion and abstraction entrance channels for $\text{O}(^1D) + \text{H}_2\text{O}$, named as O_insertion and H_abstraction ; a reference geometry H_approach is used to describe the approach of a hydrogen atom to HO_2 ; 10 of the reference points are generated by k -means algorithm to cover all configurational space of the computed points.

3.7 EHF and long range term for $\text{H}_2\text{O}(^1A_1)$ (BRANDÃO & RIO 2003)

3.7.1 Elements of the matrix

Functional form for the double-valued PES for $\text{H}_2\text{O}(^1A_1)$ was employed by BRANDÃO & RIO [48]. The dissociation channels present in $\text{H}_2\text{O}(^1A_1)$ is

$$\text{H}_2\text{O}(\tilde{X}^1A') \rightarrow \begin{cases} \text{H}_2(\tilde{X}^1\Sigma_g^+) + \text{O}(^1D) \rightarrow 2\text{H}(^2S) + \text{O}(^1D) \\ \text{H}_2(\tilde{a}^1\Sigma_u^+) + \text{O}(^3P) \rightarrow 2\text{H}(^2S) + \text{O}(^3P) \\ \text{OH}(\tilde{A}^2\Sigma) + \text{H}(^2S) \rightarrow 2\text{H}(^2S) + \text{O}(^1D) \\ \text{OH}(\tilde{X}^2\Pi) + \text{H}(^2S) \rightarrow 2\text{H}(^2S) + \text{O}(^3P). \end{cases} \quad (140)$$

The double-valued functional form is represented as the eigenvalues of a 2×2 matrix, analogous to Subsection 3.5. Each term within the matrix is represented as:

$$V_{11} = V_{\text{O}}^{(1)}(^1D) + V_{\text{OH}}^{(2)}(^2\Sigma; R_2) + V_{\text{OH}}^{(2)}(^2\Sigma; R_3) + V_{\text{HH}}^{(2)}(^1\Sigma_g^+; R_1) + V_{11(\text{LR})}^{(3)}(\mathbf{R}) + V_{11(\text{EHF, nele})}^{(3)}(\mathbf{R}), \quad (141)$$

$$V_{22} = V_{\text{OH}}^{(2)}(^2\Pi; R_2) + V_{\text{OH}}^{(2)}(^2\Pi; R_3) + V_{\text{HH}}^{(2)}(^3\Sigma_u^+; R_1) + V_{22(\text{LR})}^{(3)}(\mathbf{R}) + V_{22(\text{LR, nele})}^{(3)}(\mathbf{R}), \quad (142)$$

$$V_{12} = V_{12(\text{EHF, nele})}^{(3)}(\mathbf{R}). \quad (143)$$

BRANDÃO et al [49] employs the **Extended Hartree-Fock** (EHF) and long range term for $\text{H}_2\text{O}(^3A')$ analogously, however, instead of using the eigenvalues of a matrix, the functional form is single energy valued:

$$V = V_{\text{HH}}^{(2)}(^1\Sigma_g^+, R_1) + V_{\text{OH}}^{(2)}(^2\Pi, R_2) + V_{\text{OH}}^{(2)}(^2\Pi, R_3) + V_{(\text{LR})}^{(3)}(\mathbf{R}) + V_{(\text{EHF, nele})}^{(3)}(\mathbf{R}), \quad (144)$$

with different coefficients' and constants' values to build the functional (the explicit values are described in BRANDÃO et al [49]).

3.7.2 Diatomic terms

The diatomic terms $V_k^{(2)}(k = \text{OH}, \text{HH})$ is the sum of the dispersion correlation term and the (EHF) term, which is shown as the following:

$$V^{(2)}(R) = V_{\text{dc}}^{(2)}(R) + V_{\text{EHF}}^{(2)}(R), \quad (145)$$

where the form of the dispersion correlation energy is

$$V_{\text{dc}}^{(2)}(R) = - \sum_{n=6,8,10\dots} C_n \chi_n(R) R^{-n}, \quad (146)$$

with the damping function's formula shown as

$$\chi_n(R) = \left[1 - \exp\left(-\frac{A_n R}{\rho} - \frac{B_n R^2}{\rho^2}\right) \right]^n, \quad (147)$$

with coefficients' values as the following:

$$\begin{aligned} A_n &= 25.9528n^{-1.1868}, \\ B_n &= 15.7381\exp(-0.09729n), \\ \rho &= \frac{1}{2}(R_m + 2.5R_0), \end{aligned} \quad (148)$$

where R_m is the equilibrium diatomic geometry and R_0 is the **Le Roy parameter** introduced by R. J. LE ROY [51] shown as the following:

$$R_0(X - Y) = 2 \left(\langle r_X^2 \rangle^{\frac{1}{2}} + \langle r_Y^2 \rangle^{\frac{1}{2}} \right). \quad (149)$$

The EHF term varies depending on the diatomic form, in the case of $\text{H}_2(^3\Sigma_u^+)$, the expression is shown as

$$V_{\text{EHF}}^{(2)}(R) = -DR^m \left(1 + \sum_{i=1}^3 a_i X^i \right) \exp \left[- \sum_{i=1}^3 e_i X^i \right] + \chi_6(R) V^a(R), \quad (150)$$

with the appearing X defined as

$$X = R - R_m, \quad (151)$$

and V^a denotes the asymptotic exchange energy, which has the following expression:

$$V^a(R) = 0.805 R^{2.5} \exp(-2R). \quad (152)$$

The explicit values of the coefficients and constants appearing in the $V_{\text{dc}}^{(2)}(R)$ and $V_{\text{EHF}}^{(2)}(R)$ term for $\text{H}_2(^3\Sigma_u^+)$ are listed in the Table II of BRANDÃO & RIO [48], the aforementioned coefficients and constants are: D , $a_i (i = 1, 2, 3)$, $e_i (i = 1, 2, 3)$, R_0 , R_m , $C_i (i = 6, 8, 10, 11, 12, 13, 14, 15, 16)$ and ε . The EHF term of OH for $^2\Pi$ and $^2\Sigma$ states has the following expression:

$$V_{\text{EHF}}^{(2)}(R) = -DR^m \left(1 + \sum_{i=1}^3 a_i X^i \right) \exp[-\gamma(R)X] + \chi_6(R) V^a(R), \quad (153)$$

with the factor inside the exponential is described as

$$\gamma(R) = \gamma_0 [1 + \gamma_1 \tanh(\gamma_2 X)], \quad (154)$$

and a different V^a expression, shown as

$$V^a(R) = -\tilde{A} R^{\tilde{\alpha}} (1 + \tilde{a} R) \exp(-\tilde{\gamma} R), \quad (155)$$

where the values of the coefficients and constants are described in Table III of BRANDÃO & RIO [48], the aforementioned coefficients and constants are: D , $a_i (i = 1, 2, 3)$, $\gamma_0 (i = 0, 1, 2)$, m , $\tilde{A}$, $\tilde{\alpha}$, $\tilde{a}$, $\tilde{\gamma}$, R_0 , R_m , $C_i (i = 6, 8, 10)$, and ε .

3.7.3 Triatomic terms

The functional form to represent $V_{i(\text{EHF, nele})}^{(3)}$ (nele here denotes non-electrostatic energy), where $i = 11, 22, 12$, is shown as the following:

$$V_{i(\text{EHF, nele})}^{(3)}(R_1, R_2, R_3) = P_i^{(3)}(Q_1, Q_2, Q_3) D_i^{(3)}. \quad (156)$$

The appearing $P_i^{(3)}(Q_1, Q_2, Q_3)$, ($i = 11, 22, 12$) is a polynomial function, it has the following pattern:

$$P_i^{(3)}(Q_1, Q_2, Q_3) = \sum_j^{n_c} c_j Q_1^k Q_2^l Q_3^m S_{2a}^{2p} S_{2b}^{2q} S_3^{3r}, \quad (157)$$

where n_c indicates the number of the c_j coefficients (implies the number of terms within the polynomial function); the $Q_1, Q_3, S_{2a}^2, S_{2b}^2$, and S_3^2 are the integrity bases of the S_2 permutation symmetry; $k, m, p, q, r \in \mathbb{N}^0$, and in this case $l = 0$ (the complete values of n_c, k, m, p, q, r are available in Eqs.(20),(21), and (22) of BRANDÃO & RIO [48]). The symmetry coordinates (Q_1, Q_2 , and Q_3) has the following form:

$$\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} = \begin{bmatrix} \sqrt{1/3} & \sqrt{1/3} & \sqrt{1/3} \\ 0 & \sqrt{1/2} & -\sqrt{1/2} \\ \sqrt{2/3} & -\sqrt{1/6} & -\sqrt{1/6} \end{bmatrix} \begin{bmatrix} R_1 - R_1^{(i)} \\ R_2 - R_2^{(i)} \\ R_3 - R_3^{(i)} \end{bmatrix}, \quad (158)$$

where $R_j^{(i)} (j = 1, 2, 3)$ refers to the reference geometries (the explicit values are available in Table V of BRANDÃO & RIO [48]); the rest of the integrity bases have the following form:

$$\begin{aligned} S_{2a}^2 &= Q_2^2 + Q_3^2, \\ S_{2b}^2 &= Q_2^2 - Q_3^2, \\ S_3^3 &= Q_3^3 - 3Q_2^2 Q_3. \end{aligned} \quad (159)$$

The considered degrees for the polynomials are: 8 (95 terms) for $i = 11$, 5 (34 terms) for $i = 22$, and 3 (13 terms) for $i = 12$. The complete coefficients' values are available in Table VI, VII, and VIII in BRANDÃO & RIO [48] for $i = 11, 22$ and 12 respectively. The range factor is expressed as the product of hyperbolic tangents, except for $i = 12$, the complete form is shown as the following:

$$D_{11}^{(3)} = \left\{ 1 - \tanh \left[c_{96} (R_1 - R_1^{(11)}) \right] \right\} \left\{ 1 - \tanh \left[c_{97} (R_2 - R_2^{(11)}) \right] \right\} \left\{ 1 - \tanh \left[c_{96} (R_3 - R_3^{(11)}) \right] \right\}, \quad (160)$$

$$D_{22}^{(3)} = \left\{ 1 - \tanh \left[c_{132} (R_1 - R_1^{(22)}) \right] \right\} \left\{ 1 - \tanh \left[c_{133} (R_2 - R_2^{(22)}) \right] \right\} \left\{ 1 - \tanh \left[c_{133} (R_3 - R_3^{(22)}) \right] \right\}, \quad (161)$$

$$D_{12}^{(3)} = \sin(\angle \text{HOH}) \exp \left[-100(V_{11} - V_{22})^2 \right] \exp \left[(-c_{147} (R_1 - R_1^{(12)})^{(2)}) \right] \exp \left[(-c_{148} (R_2 + R_3 - 2R_3^{(12)})^{(2)}) \right], \quad (162)$$

where the value of $\angle \text{HOH}$ is mentioned in Table IX in in BRANDÃO & RIO [48], which also describes the minimum C_{2v} for water molecule. The $V_{i(\text{LR})}^{(3)}$, ($i = 11, 22$) terms appearing in (141-142) are the long-range terms (described by BRANDÃO & RIO [52]), represented as the sum of electrostatic energy, induction energy, and dispersion correlation energy:

$$V_{i(\text{LR})}^{(3)} = V_{(\text{ele})}^{(3)} + V_{(\text{ind})}^{(3)} + V_{(\text{dc})}^{(3)}, \quad (163)$$

with

$$V_{(\text{ele})}^{(3)} = \frac{3}{4} \Theta_{\text{H}_2}(R) \Theta_{\text{O}} \mathcal{A}(\omega) \chi_5(r) r^{-5} \quad (164)$$

as the electrostatic energy (each factor and its coefficients are explained in more detail by VARANDAS & BRANDÃO [53] and VARANDAS & PAIS [54]); the induction energy is

$$V_{(\text{ind})}^{(3)} = - \sum_{i=2}^3 \mu_{\text{OH}_i}^2 (R_i) \alpha_{\text{H}} (3 \cos^2 \theta_i + 1) \chi_6(r_i) / (2r_i^6); \quad (165)$$

the dispersion correlation energy is

$$V_{\text{(dc)}}^{(3)} = \sum_{i=1}^3 S(R_i, r_i) \sum_{n=6}^{10} C_n^i(R_i, \theta_i) \chi_n(r_i) r_i^{-n} \\ + \sum_{i=1}^3 \left[\prod_{j \neq i} (1 - S(R_j, r_j)^2) \right] \sum_{n=6}^{10} C_n^i \chi_n(R_i) R_i^{-n}, \quad (166)$$

with

$$S(R, r) = \frac{1}{2} \left\{ 1 + \tanh \left[\gamma_1 \left(\frac{r}{R} - \gamma_2 \right) \right] \right\}, \quad (167)$$

where R is diatom separation, r is atom-diatom distance, and the index i specifies the atom-diatom combination; γ_1 and γ_2 are calibration parameters, where in this case $\gamma_1 = \gamma_2 = 2$.

3.8 V_R potentials for HO₂ (BRANDÃO et al 2009)

BRANDÃO et al [50] employed a functional form for HO₂, which has the following expression:

$$V_{\text{Rspect}} = T \exp(-D), \quad (168)$$

where T is a third degree polynomial function with 13 adjustable coefficients (the values are obtainable from Table VII in BRANDÃO et al [50]), shown as the following:

$$T = c_1 + c_2 R_{1d} + c_3 S_{1d} + c_4 R_{1d}^2 + c_5 S_{1d}^2 + c_6 R_{1d} S_{1d} + c_7 S_{2d} + c_8 R_{1d}^3 + c_7 S_{2d} + c_8 R_{1d}^3 \\ + c_9 R_{1d} S_{1d}^2 + c_{10} R_{1d}^2 S_{1d} + c_{11} R_{1d} S_{2d} + c_{12} S_{1d}^3 + c_{13} S_{1d} S_{2d}, \quad (169)$$

and the appearing D in (168) is a decay term with fixed $c_{i_{\text{fix}}}$ ($i = 1, 2, 3, 4$) (BRANDÃO et al [50] set all $c_{i_{\text{fix}}}$ values to 2), shown as the following:

$$D = c_{1_{\text{fix}}} R_{1d}^2 + c_{2_{\text{fix}}} S_{1d}^2 + c_{3_{\text{fix}}} R_{1d} S_{1d} + c_{4_{\text{fix}}} S_{2d}^2, \quad (170)$$

R_2 and R_3 denotes the O^{*i*}-H ($i = 1, 2$) interatomic distances; the R_{1d} , S_{1d} , and S_{2d} are the displacements coordinates, shown as the following:

$$R_{1d} = R_1 - R_{1_{\text{eq}}}, \\ S_{1d} = \frac{1}{\sqrt{2}} [(R_2 + R_3) - (R_{2_{\text{eq}}} + R_{3_{\text{eq}}})], \\ S_{2d} = \frac{1}{2} [(R_2 - R_3)^2 - (R_{2_{\text{eq}}} - R_{3_{\text{eq}}})^2], \quad (171)$$

where the values of the experimental equilibrium geometry ($R_{i_{\text{eq}}}$ ($i = 1, 2, 3$)) are set as the following:

$$R_{1_{\text{eq}}} = 2.5143 \text{a}_0, \\ R_{2_{\text{eq}}} = 1.8346 \text{a}_0, \\ R_{3_{\text{eq}}} = 3.4592 \text{a}_0. \quad (172)$$

3.9 MBE/DC PES for H₂O (GALVÃO & RODRIGUES 2008)

Employed by GALVÃO & RODRIGUES [55]. The functional form is

$$G^{(3)}(\mathbf{R}) = \sum_{i=1}^{37} P_i(\mathbf{R}) \exp \left[- \sum_{j=1}^3 b_{ij} (R_j - R_{ij}^0)^2 \right], \quad (173)$$

where the polynomials (three interatomic coordinates) are written in the terms of D_{3h} symmetry coordinates, which is shown as

$$\begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \end{pmatrix} = \begin{pmatrix} \sqrt{1/3} & \sqrt{1/3} & \sqrt{1/3} \\ 0 & \sqrt{1/2} & -\sqrt{1/2} \\ \sqrt{2/3} & -\sqrt{1/6} & -\sqrt{1/6} \end{pmatrix} \begin{pmatrix} R_3 \\ R_2 \\ R_1 \end{pmatrix}, \quad (174)$$

where R_1 and R_2 are OH bond distances while R_3 is HH. Hence, the general form of $P_i(\mathbf{r})$ is

$$P_i(\mathbf{r}) = c_{i0} + c_{i1}Q_2^2 + c_{i2}Q_2^4 + c_{i3}Q_2^6 + c_{i4}Q_1 + c_{i5}Q_1^2 + c_{i6}Q_3 + c_{i7}Q_3^2. \quad (175)$$

To warrant proper symmetry in a permutation of R_1 and R_2 , no odd powers of variable Q_2 are used to construct the polynomials. In total, the function has 212 adjustable parameters, including the polynomial and Gaussian coefficients.

3.10 Neural network for H₃O (CHEN et al 2013)

A PES model in the form of feed-forward **neural network** (NN) with two hidden layers is employed by CHEN et al [56]. Formally, each layer is

$$y_{k,j} = f_k \left(b_{k,j} + \sum_{i=1}^{N_{k-1}} (w_{k,j,i} \times y_{k-1,i}) \right), \quad j = 1, 2, \dots, N_k, \quad (176)$$

where N_k indicates the number of neurons in the k th layer; $y_{k,j}$ is the output on the k th layer and j th node, in the case of $k = 0$, the $y_{0,i} (i = 1, \dots, N_0)$ is the input value, which is one value from the set of the molecular distances; $b_{k,j} \in \mathbb{R}$ is the bias on the k th layer and j th node; $w_{k,j,i} \in \mathbb{R}$ is the weight on the k th layer and j th node which connects the i th neuron of the $(k-1)$ th layer; f_k is the activation function of the k th layer, in this case $\tanh$ (the hyperbolic tangent) is chosen. Consequently, the final output neuron of the output layer $y_{\bar{k}}$ follows analogously to (176), however it is not necessary to apply the activation function f_k , it is formally given as

$$y_{\bar{k}} = b_{\bar{k}} + \sum_{i=1}^{N_{\bar{k}-1}} (w_{\bar{k},j,i} \times y_{\bar{k}-1,i}). \quad (177)$$

In this case of H₃O system, the permuted set of distances $\{r_{\text{OH}_1}, r_{\text{OH}_2}, r_{\text{OH}_3}, r_{\text{H}_1\text{H}_2}, r_{\text{H}_1\text{H}_3}, r_{\text{H}_2\text{H}_3}\}$, satisfying $r_{\text{OH}_1} \leq r_{\text{OH}_2} \leq r_{\text{OH}_3}$, is fed into the input layer.

3.11 Lennard-Jones-like pair potential (DEITERS & NEUMAIER 2016)

A modification of the **Lennard-Jones** (LJ) potential [13] is proposed by DEITERS & NEUMAIER [14] for the PES of Argon. The form of the well-known LJ potential is

$$V(R) = 4\epsilon \left[\left(\frac{\sigma}{R} \right)^{12} - \left(\frac{\sigma}{R} \right)^6 \right], \quad (178)$$

where R is the interatomic distance, σ is the collision diameter, and ϵ is the depth of the "well" in the potential energy curve. It has been noted by DEITERS & NEUMAIER [14] that the short distance repulsion of LJ potential is less realistic when compared to the experimental results, hence the new form of the pair potential is proposed. The earliest attempt on devising a more realistic pair potential was in the form of BUCKINGHAM [57] potential

$$V(R) = c_1 e^{-\alpha R} - \frac{c_2}{R^6}, \quad (179)$$

the drawback of the Buckingham potential is that it is infinitely attractive, i.e., it is infinitely attractive for $R \rightarrow 0$. PATHAK & THAKKAR [58] state that for monoatomic gases, the united-atom perturbation theory implies

$$V(R) = c_1 \frac{Z^2}{R} + c_2 + O(R^2) \quad \text{for } R \ll R_e, \quad (180)$$

where Z is the nuclear charge, R_e is the equilibrium distance, $c_1 = 1$ (in Hartree unit), and c_2 depends on the substance. With exponential screening of the Coulomb potential, the functional form becomes

$$V(R) = e^{-\alpha R} \left(c_1 \frac{Z^2}{R} \left(1 - \frac{(\alpha R)^2}{2} \right) + c_2 (1 + \alpha R) \right) \quad \text{for } R \ll R_e. \quad (181)$$

This implies that pair potentials fitted to data at larger R should also include the correct short distance behaviour such as in (181). This can be implemented by matching c_2 and α to the value $V_s = V(R_s)$ and the slope $V'_s = V'(R_s)$ of the potential at small switching radius R_s . It can be verified that

$$\begin{aligned} & \alpha^2 R V(R) + (1 + \alpha R) V'(R) \\ &= e^{-\alpha R} c_1 \frac{Z^2}{2R^2} (1 + \alpha R) (2 + 2\alpha R + 3(\alpha R)^2 - (\alpha R)^3 - (\alpha R)^4) \end{aligned} \quad (182)$$

when evaluated at R_s gives a nonlinear equation to be solved for α , which then c_2 can be found from the equation for $V(R)$. By varying R_s , one can obtain different variants of the same potential, e.g., to assess the effect of the repulsive part.

The newly proposed form by DEITERS & NEUMAIER [14] is used to correct the R^{-6} behavior at large distances, the desired short-range behavior, and to avoid unrealistic (infinitely attractive) pole at $R = 0$. The form of the modified potential is

$$V(R) = c_1 e^{-2\alpha R} \left(\alpha + \frac{1}{R} \right) - \frac{c_2 R^2}{c_3 + R^8} + c_4, \quad (183)$$

where $\alpha, c_1 \dots c_3$ are chosen non-negative parameters, and c_4 is the energy shift parameter; the explicit parameters' values are shown in Table 1 of DEITERS & NEUMAIER [14].

3.12 Pair potential for noble gases (DEITERS & SADUS 2019)

DEITERS & SADUS [15] proposed a simplified *ab initio* atomic potential (SAAP) for the noble gases (He, Ne, Ar, Kr, Xe). The functional form is

$$V(R) = \frac{\left(\frac{c_0}{R}\right) e^{c_1 R + c_6 R^2} + c_2 e^{c_3 R} + c_4}{1 + c_5 R^6} + c_7, \quad (184)$$

where $c_1...c_4 < 0$, $c_0, c_5 > 0$, $c_6 = 0$ except for He where $a_6 < 0$ to represent the repulsive behaviour in extremely small interatomic distances, and c_7 is the energy shift parameter. The complete parameters are summarized in Table 1 of DEITERS & SADUS [15]. The form is proposed in order to address the correct potential curve with minimal computational cost, the considerations of the functional form are:

- The potential curve should pass the abscissa once at σ , which flips the sign from positive to negative of $V(R)$ for $R > \sigma$.
- The domination of dispersion interaction at large R , which is represented by R^{-6} , with V monotonously converge to zero.
- The **Pauli repulsion** at short distances should be represented, in particular, the $\lim_{R \rightarrow 0} V(R) = \infty$, in this case it is described by $\frac{e^R}{R}$ term.

4 Datasets and proposed functions of pair potentials containing H and O atoms

The first half of this section (Section 4.1-4.3) contains the pair potential datasets that were able to be acquired during the research timeline. Most of them were curated from old papers in which sometime the electronic versions are unavailable, or, if available, they are already in a *compiled* version in the tables or texts where they require thorough data cleaning.

The second half (Section 4.4-4.6) contains the proposed functions to be fitted on the pair potential datasets and the corresponding benchmark.

4.1 H₂ data

The data points for ground state H₂ (Figure 9) are obtained from HAGER-ROISER [59], it contains:

- Data sizes:
 - KOLOS & WOLNIEWICZ (1965) [60]: 87 data points.
 - WOLNIEWICZ (1993) [61]: 670 data points.
- The energy values of WOLNIEWICZ (1993) are shifted by $V_i := V_i - V_{670}$, $i = 1, 2, \dots, 670$, to accommodate the asymptotic energy $\lim_{R \rightarrow \infty} V(R) = 0$.

4.1.1 H₂⁺ data

The H₂⁺ datasets are obtained from SHARP (1971) [23], with the following information:

- Data sizes:
 - $X^2\Sigma_g^+ 1s\sigma_g$ state: 29 data points.
 - $\sigma_g 1s$ state: 29 data points.
 - $2p\sigma_u$ state: 29 data points.
 - $\sigma_u 1s$ state: 29 data points.
- The energy and distance units are converted from (Angstrom, eV) to (Bohr, Hartree).
- The data of $\sigma_u 1s$ state is corrected by removing the (R_{25}, V_{25}) data point.
- For each state, the data are shifted by the first ionization energy of H, $V_{\text{ion}} = 1312$ kJ/mol.
- Only $X^2\Sigma_g^+ 1s\sigma_g$ and $2p\sigma_u$ states are used while $\sigma_g 1s$ and $\sigma_u 1s$ states are excluded due to redundancy.

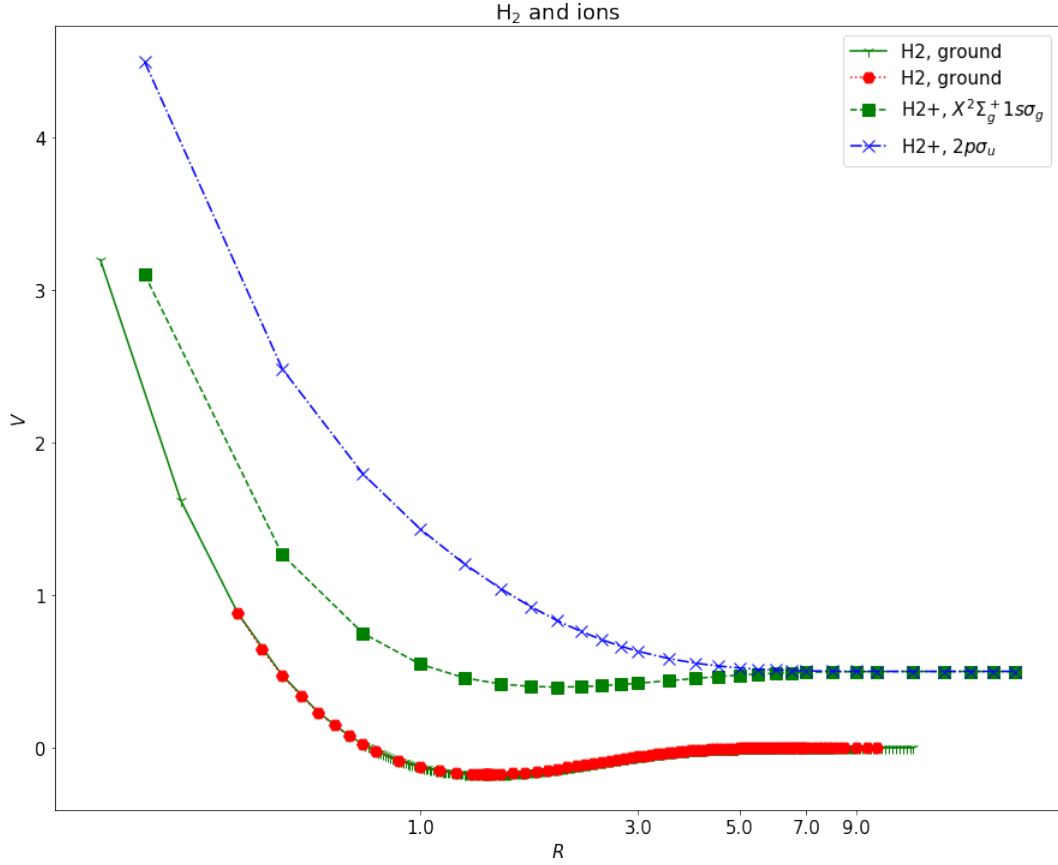


Figure 9: Potential energy curves of H₂ and its ions.

4.2 O₂ data

The data points for O₂ (Figure 10) are provided by BYTAUTAS et al (2010)[62], which describes:

- Data sizes:
 - $^3\Sigma_g^-$ state: 26 data points.
 - $^1\Delta_g$ state: 26 data points.
 - $^1\Sigma_g^+$ state: 26 data points.
- The energy and distance units are converted from (Angstrom, millihartree) to (Bohr, Hartree) (1 Hartree = 1000 millihartree).

4.2.1 O₂⁺ data

The data points for O₂⁺ are provided by XAVIER et al (2018) [63], the information contains:

- Data sizes:
 - $X^2\Pi_g$ state:
 - * AV5Z method: 51
 - * AV6Z method: 51
 - * CBS method: 51

- The unit of R is converted from Angstrom to Bohr.
- The V is shifted by the first ionization energy of O, $V_{\text{ion}} = 1313.9$ kJ/mol.
- Only one dataset is chosen due to the fact that each dataset is identical to each other (e.g., only the dataset from CBS method is picked).

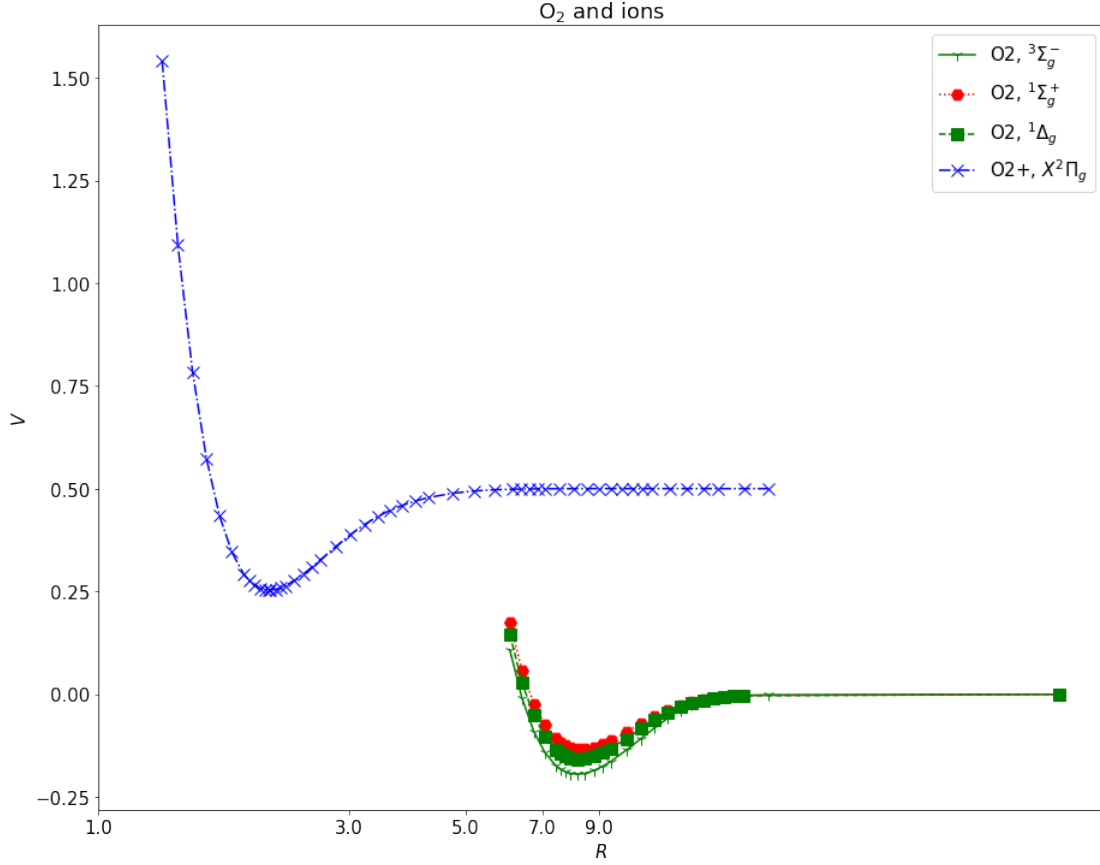


Figure 10: Potential energy curves of O_2 and its ions.

4.3 OH data

The obtained OH datasets (Figure 11) are described by:

- Data sizes:
 - $X^2\Pi$ state:
 - * PRADHAN (1995) [64] (from SHIZGAL (1999) [65]): 29 data points.
 - * NEMUKHIN & GRIGORENKO (1997) [66] (from SHIZGAL (1999) [65]): 32 data points.
 - * VAN DISHOECK ET AL (1983) [67] (from SHIZGAL (1999) [65]): 24 data points.
 - * HODGES (1993) [68] (from SHIZGAL (1999) [65]): 32 data points.
 - * CHU et al (1974) [69] using MCSCF method: 22 data points.
 - * CHU et al (1974) [69] using CI method: 22 data points.
 - * WERNER et al (1983) [70], using MCSCF-SCEP method: 17 data points.

- * WERNER et al (1983) [70], using SCEP-CEPA: 12 data points.
- $A^2\Sigma^+$ state:
 - * CHU et al (1974) [69], using MCSCF method: 22 data points.
 - * CHU et al (1974) [69], using CI method: 22 data points.
- $^4\Sigma^-$ state, obtained from SHIZGAL (1999) [65]:
 - * VAN DISHOECK & DALGARNO (1983) [71]: 23 data points.
 - * HODGES (1993) [68]: 23 data points.
 - * COOPER et al (1984) [72]: 23 data points.
- $^2\Sigma^-$ state, obtained from SHIZGAL (1999) [65]:
 - * PRADHAN (1995) [64]: 21 data points.
 - * VAN DISHOECK et al (1983) [67]: 14 data points.
 - * HODGES (1993) [68]: 24 data points.
- $^4\Pi$ state, obtained from SHIZGAL (1999) [65]:
 - * VAN DISHOECK & DALGARNO [71]: 20 data points.
 - * HODGES (1993) [68]: 20 data points.
 - * COOPER et al (1984) [72]: 20 data points.
- Data shifts:
 - V values for each dataset of CHU et al (1974) [69] for $A^2\Sigma^+$ and $X^2\Pi$ states are shifted by its corresponding V_n .
 - V values on each dataset of WERNER et al (1983) [70] for $X^2\Pi$ state are shifted by its corresponding V_n .
- Data corrections:
 - Data of CHU et al (1974) [69] for $X^2\Pi$ state is corrected by removing (R_1, V_1) .
 - Data of VAN DISHOECK & DALGARNO [71] for $^4\Pi$ state is corrected by removing (R_1, V_1) .
 - Data of VAN DISHOECK & DALGARNO [71] for $X^2\Pi$ state is corrected by removing (R_4, V_4) .
- The datasets used for fitting are:
 - $\{X^2\Pi, \text{Pradhan}(1995)\}$.
 - $\{X^2\Pi, \text{Nemukhin and Grigorenko}(1997)\}$.
 - $\{X^2\Pi, \text{vanDishoeck et al}(1983)\}$.
 - $\{^2\Sigma^-, \text{Pradhan}(1995)\}$.
 - $\{^2\Sigma^-, \text{vanDishoeck et al}(1983)\}$.
 - $\{^2\Sigma^-, \text{Hodges}(1993)\}$.
 - $\{^4\Sigma^-, \text{vanDishoeck and Dalgarno}(1983)\}$.
 - $\{^4\Sigma^-, \text{Hodges}(1993)\}$.
 - $\{^4\Pi, \text{vanDishoeck and Dalgarno}(1983)\}$.
 - $\{^4\Pi, \text{Hodges}(1993)\}$.

- $\{X^2\Pi, \text{Chu et al (1974)}\}$.
- $\{A^2\Sigma^+, \text{Chu et al (1974)}\}$.
- $\{A^2\Sigma^+, \text{Fallon et al(1960)}\}$.
- Below datasets are excluded due to disagreeing with the majority of the datasets in the same molecular state:
 - $\{^4\Sigma^-, \text{Cooper et al(1984)}\}$.
 - $\{^4\Pi, \text{Cooper et al(1984)}\}$.
 - $\{A^2\Sigma^+, \text{Chu et al (1974)}\}$.
 - $\{X^2\Pi, \text{Hodges(1993)}\}$.
 - $\{X^2\Pi, \text{Werner et al(1983)}\}$.
 - $\{X^2\Pi, \text{Chu et al (1974)}\}$.

4.3.1 OH⁺ data

The OH⁺ data is obtained from WERNER et al (1983) [70] and XAVIER et al (2018) [63], the contained information is:

- Data sizes:
 - $X^3\Sigma^-$ state:
 - * XAVIER et al (2018) [63]:
 - AV5Z method: 32
 - AV6Z method: 32
 - CBS method: 32
 - * WERNER et al (1983):
 - MCSCF-SCEP method: 12 data points.
 - SCEP-CEPA method: 12 data points.
- The unit of XAVIER et al (2018) [63] R is converted from Angstrom to Bohr.
- The V values are shifted by the first ionization energy of O, $V_{\text{ion}} = 1313.9$ kJ/mol.
- Only the datasets of XAVIER et al [63] with AV5Z method and WERNER et al (1983) with SCEP-CEPA method are included, meanwhile the excluded datasets (in the form of {state, author}) are:
 - $\{X^3\Sigma^-, \text{Werner et al(1983)}\}$ due to disagreeing with other datasets.
 - $\{X^3\Sigma^-, \text{Xavier(2018) - AV6Z method}\}$ due to redundancy.
 - $\{X^3\Sigma^-, \text{Xavier(2018) - CBS method}\}$ due to redundancy.

4.3.2 OH⁻ data

The OH⁻ data is obtained from WERNER et al (1983) [70], the contained information is:

- Data sizes:

– $X^1\Sigma^+$ state:

- * MCSCF-SCEP method: 17 data points.
- * SCEP-CEPA method: 10 data points.

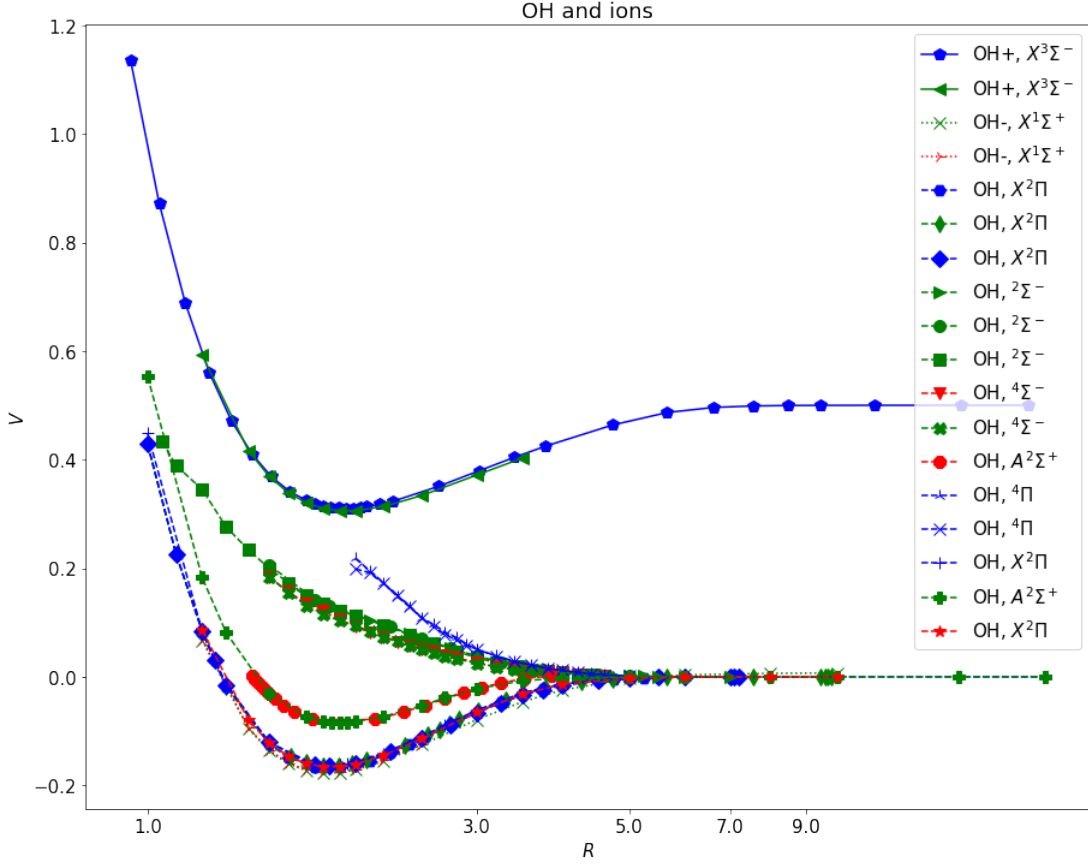


Figure 11: Potential energy curves of OH and its ions.

4.4 Proposed pair potential ansatz

In this section, we propose some functional forms to be used for pair potential fitting. The numerical results are shown in Section 4.6.

4.4.1 Polynomial fit

The proposed function for diatomic potential takes the concept of monomial expansion from (86), the specific form for diatomic is

$$V = \sum_{k=0}^M C_k y^k, \quad (185)$$

where M is the maximum polynomial degree; C_k is the coefficient which corresponds to the k th power which needs to be fitted; in this case y is the Morse potential described in (85). For example, for $M = 3$, the explicit form is

$$V = C_0 + C_1 y + C_2 y^2 + C_3 y^3, \quad (186)$$

which results in four coefficients to be fitted. An additional note, any polynomials are evaluated by using the **Horner's scheme** [73] to maximize the computation's efficiency.

4.4.2 RATPOT1

A proposed functional form which contains Coulomb-like term, repulsion term, attraction term, and energy shift constant; generalized with M -degrees of polynomial, which in total $3M + 1$ parameters are adjustable; the functional form is

$$V = c_0 + \frac{P(R)}{S(R)}, \quad (187)$$

where

$$P(R) = Z_{ij} \left(\frac{1}{R} + c_1 R \right) + c_2 + c_3 R^2 + \dots + c_{2M-1} R^{2M-2}, \quad (188)$$

$$Q(R) = 1 + (c_{2M} + c_{2M+1} R + \dots + c_{3M} R^M) R, \quad (189)$$

$$S(R) = 1 + c_1 (RQ(R))^2, \quad (190)$$

and

$$c_1 > 0, \quad (191)$$

where

$$Z_{ij} := Z_i Z_j, \quad (192)$$

Z_k is the k th atom's nuclear charge, e.g., $Z_{\text{OH}} = Z_{\text{O}} Z_{\text{H}} = 1 \times 8 = 8$. For efficient computation, the polynomials are evaluated using Horner's scheme analogous to (185) and the rest of the operations are executed in certain order; (187 - 191) are evaluated algorithmically as the following:

Function: Diatomic General M

Input: C, R, Z, M

Output: V

if $C_1 < 0$:

$C_1 := -C_1$

end if

$R_2 := R^2$

$K := [C_3 \dots C_{2M-1}]$

$y := \text{horner}(R, K)$

$a := Z(1/R + C_1 R) + C_2 + y$

$K := [C_{2M} \dots C_{3M}]$

$y := \text{horner}(R, K)$

$b := 1 + yR$

$c := 1 + C_1 (Rb)^2$

$V := C_0 + a/c,$

where the Horner's scheme is

Function: horner

Input: x, C

Output: y

$n := \text{length of } C$

$y := C_n$

for $i = n - 1, n - 2 \dots 0$, do:

$y := yx + C_i$

end for.

4.4.3 RATPOT2

This version of RATPOT has a total of $4M + 7$ tuning parameters, which are the vectors a, b, c , and d . RATPOT2 does not have the nuclear charge as a dependency, it is replaced by one tuning parameter, this should allow more degree of freedom in terms of the data fitting. The functional form is

$$\begin{aligned} V_m(R) &= c_0 + P/Q, \\ P &= c_1 \prod_{i=1}^m ((R - a_i)^2 + b_i R), \\ Q &= R(R + d_1) \prod_{i=2}^{m+3} ((R - c_i)^2 + d_i R), \end{aligned} \tag{193}$$

where $m > 0$ and $b_i \geq 0$ for $i > 1$ and $d_i \geq 0$ for $i > 0$.

4.4.4 RATPOT3

Similar to RATPOT2, RATPOT3 has a, b, c , and d as the tuning parameters' vectors, with a total of $4M + 8$ tuning parameters. The nuclear charge coefficient is included back. The functional form is

$$\begin{aligned} P &= Z(1/R - 1/R_0) \prod_{i=1}^m ((1 - R/a_i)^2 + R/b_i), \\ Q &= \prod_{i=1}^{m+3} ((1 - R/c_i)^2 + R/d_i), \end{aligned} \tag{194}$$

where $m \geq 0$ and all $b_i, d_i \geq 0$.

4.4.5 RATPOTL

An ansatz in order to solve pair potentials linearly was devised. The main idea is that the pair potentials can be used as a basis function for potentials of molecules containing more atoms. Hence this requires a cheap-to-compute ansatz, or even only needed to be computed once in the beginning (pre-computation) without requiring the parameters to be tuned, which reduces the complexity of the optimization problem.

In the first step, the diatomic distances need to be scaled

$$\rho_{ij} := R_{ij}/R_{xy}, \tag{195}$$

where R_{ij} is the diatomic distance between atom i of type x and atom j of type y , R_{xy} is the **equilibrium distance** of the pair potential between atom type x and y . Atomic equilibrium distance is the diatomic distance which corresponds to the lowest pair potential energy, i.e., when the atoms neither repel nor attract each other. For example, the equilibrium distance of H_2 molecule is $R_{\text{HH}} \approx 1.4172946$ Bohr.

Then the equilibrium distance needs to be shifted to the center, in this case, 0 is chosen as the center, and the $[\min(\rho), \max(\rho)]$ are shifted to $[-1, 1]$, this is done by

$$q := \frac{1 - \rho}{1 + \rho} \in [-1, 1]. \tag{196}$$

Finally, the pair potential form is

$$V_k(R) = \frac{u(q)}{\rho + \rho^k}, \quad (197)$$

where $k = 1 : 6$ is a hyperparameter which denotes the asymptotic behavior of the transformed distance, where $k = 6$ is the theoretically expected best power since it complies with R^{-6} . The numerator

$$u(q) = \sum_l p_l(q) \theta_l. \quad (198)$$

is the linear combination of Chebyshev polynomials of degree d (which means it has $d + 1$ parameters)

Although it is possible to fit this pair potential ansatz directly with sufficiently high accuracy (low RMSE) and reasonable speed, the main idea is to solve this linearly, in order to do this, (198) needs to be transformed into a system of linear equations. With a total number of n data, and d degrees of polynomials, the system of the linear equation is

$$P\theta = V, \quad (199)$$

where $P \in \mathbb{R}^{n \times d+1}$ is a matrix containing the Chebyshev polynomials entries $p_l(q) \in \mathbb{R}^n$ for each column, where the first column is a $\vec{1}$ column vector since $p_0(q) = 1$, which then each entry is scaled (elementwise division) by

$$P := P/(\rho + \rho^k) \in \mathbb{R}^{n \times d+1}; \quad (200)$$

$\theta \in \mathbb{R}^{d+1}$ is the vector containing the tuning parameters; and $V \in \mathbb{R}^n$ is a vector containing the potential energy data points. In this manner, θ can be obtained by solving (199) linearly. For example in programming language such as Julia and MATLAB, this can be done simply by

```
theta = P \ V
```

Possible ansatz to scale the RATPOT for solving it linearly (other than (197)) are

$$u(q) \approx (\rho + \rho^k) V_k(R), \quad (201)$$

and

$$\frac{u(q)}{1 + \rho^{k-1}} \approx \rho V_k(R). \quad (202)$$

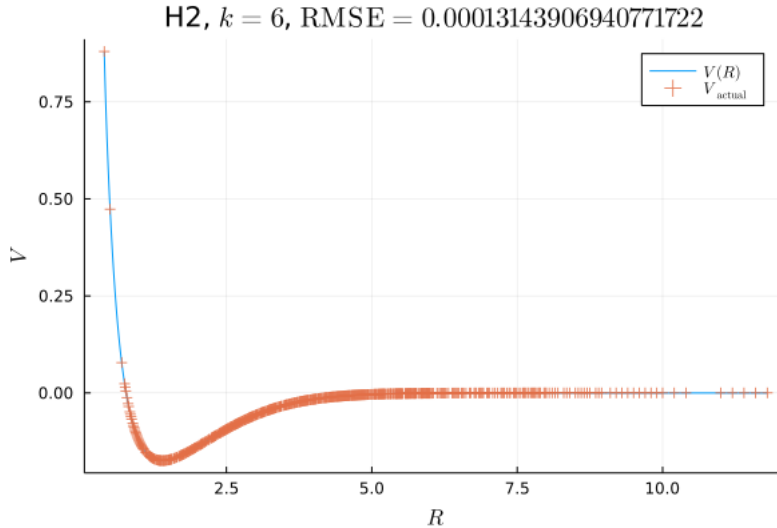


Figure 12: Nonlinear leastsquares fitting applied for linear RATPOT.

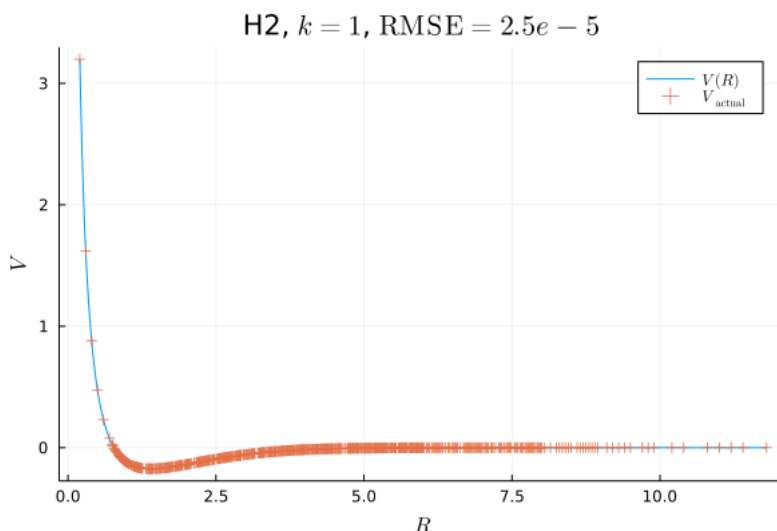


Figure 13: Linear RATPOT is solved using linear solver.

4.5 Fitting method

4.5.1 Fitting for general pair potential ansatz

There exist several possible objective function forms which can be used to fit the diatomic functional forms. One possible objective function form is the RMSE in (286); another one would be the sum of squared residuals

$$\sum_{i=1}^n [V(R_i) - V_{i,\text{ab initio}}]^2. \quad (203)$$

There are two types of optimization routines for nonlinear least squares form, one requires the objective function to return a residual vector, another one is the standard optimization routine which requires the objective function to return a scalar. In the case of overdetermined system (more data than tuning parameters), **Levenberg-Marquardt** (LM) algorithm [74] is employable; otherwise, local search such as **trust region reflective** (TRF) [75] and **dogleg** algorithm with rectangular trust regions [76] are employable. This rule is in fact molecule-size-agnostic, i.e., applicable to any molecule size.

4.5.2 Fitting dissociated energy

For any functional form of potential energy, it is possible to scale the curve in a way that the repulsive and attractive part of the potential curve is more visible. This is done by scaling both the functional form and the data by the dissociation energy

$$\Delta = V_l - V_{\min}, \quad (204)$$

where V_l is the potential energy at the largest distance and $V_{\min}$ is the minimum observed potential energy. This is done during the fitting (also applies to direct solvers):

$$\theta_{\min} = f(V_{\text{ab initio}}/|V_{\text{ab initio}}| + \Delta, V(R)/|V(R)| + \Delta), \quad (205)$$

where $\theta_{\min}$ is a vector containing optimized tuning parameters, here f is an objective function for data-fitting, for example (203); and $V(R)$ is the functional form. Similarly, an adjusted RMSE for dissociated energy is

$$\text{aRMSE} := \Delta \text{RMSE}(\delta), \quad (206)$$

where

$$\delta := \epsilon / (|V| + \Delta), \quad (207)$$

and

$$\epsilon = V_{\text{ab initio}} - V(R). \quad (208)$$

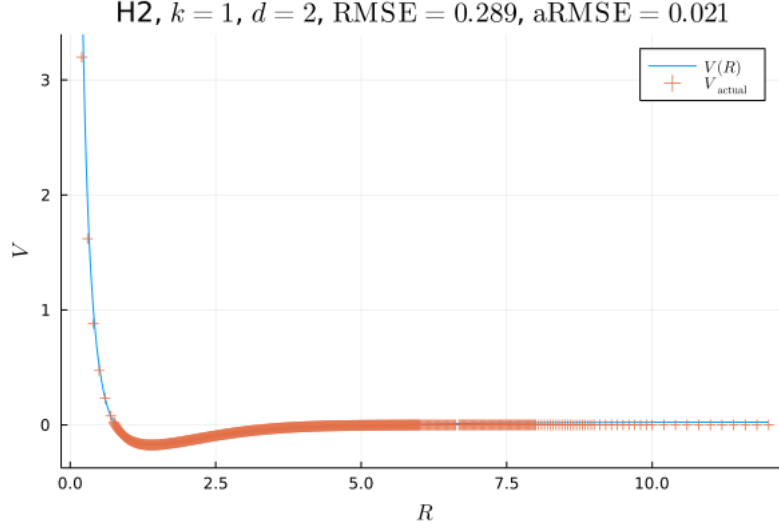


Figure 14: Direct linear solve for linear dissociated RATPOT with 5 initial data points and 5 end data points as validation data.

4.6 Pair potentials benchmark

We compare the fitting performance of the RATPOTs against CHIPR (which is described in Section 3.2). For all batches of experiments, random multirestart scheme is used to find better minima with respect to the generalization error.

The first batch of experiments are done on OH^+ dataset, where 25% of it is used for the test evaluation. RATPOT1, RATPOT2, and CHIPR are compared on varied size of fitting parameters. As shown in Figure 15, RATPOT2 has the advantage of accuracy and time evaluation with only 27 fitting parameters.

The second batch of experiments are also done on the same setting as the first batch, the difference here is that RATPOT3 ansatz is added for benchmark. The main goal here is to see if the errors can only decrease if we freeze a subset of the fitting parameters for each fitting run. In particular, for a batch of experiments $t = 1, 2, \dots, T$, experiment t would have n_t amount of fitting parameters for each function ansatz, where $n_{t-1} < n_t$, and we freeze n_{t-1} amount of fitting parameters; freezing means that the values of the fitting parameters optimized at experiment $t - 1$ are re-used for experiment t . The sample result of the experiments are shown in Figure 16.

Last batch of experiments on the pair potentials make use the datasets in Section 4.1-4.3. The results are shown in Figure 17. For each dataset, a 5-fold cross validation scheme is used, and the min, median and max errors from the cross validation are captured. We sample OH^+ benchmark to see the the timing of the evaluation; here we see that RATPOT2 has the lowest error and evaluation time.

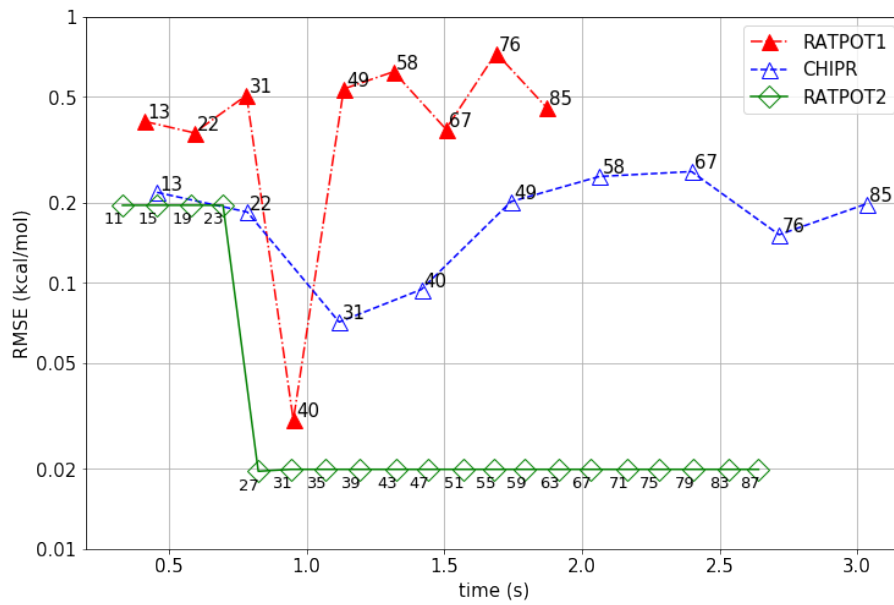


Figure 15: Benchmark of RATPOT1, RATPOT2, and CHIPR on a OH^+ dataset. The numbers on the markers denote the number of fitting parameters.

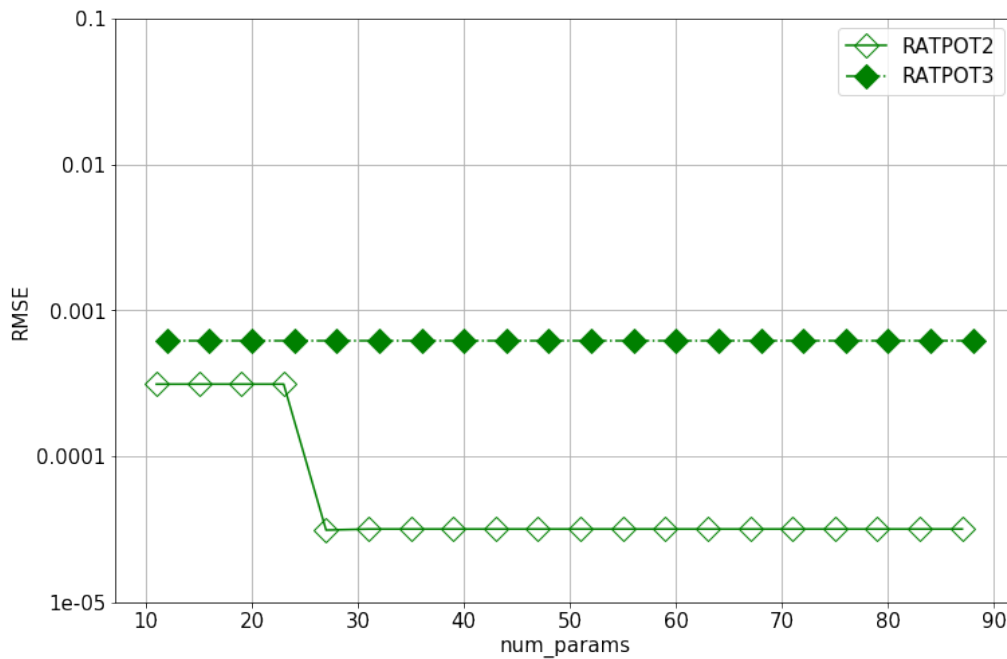


Figure 16: Experiments on freezing subsets of parameters of RATPOT2 and RATPOT3. RMSE is in Hartree.

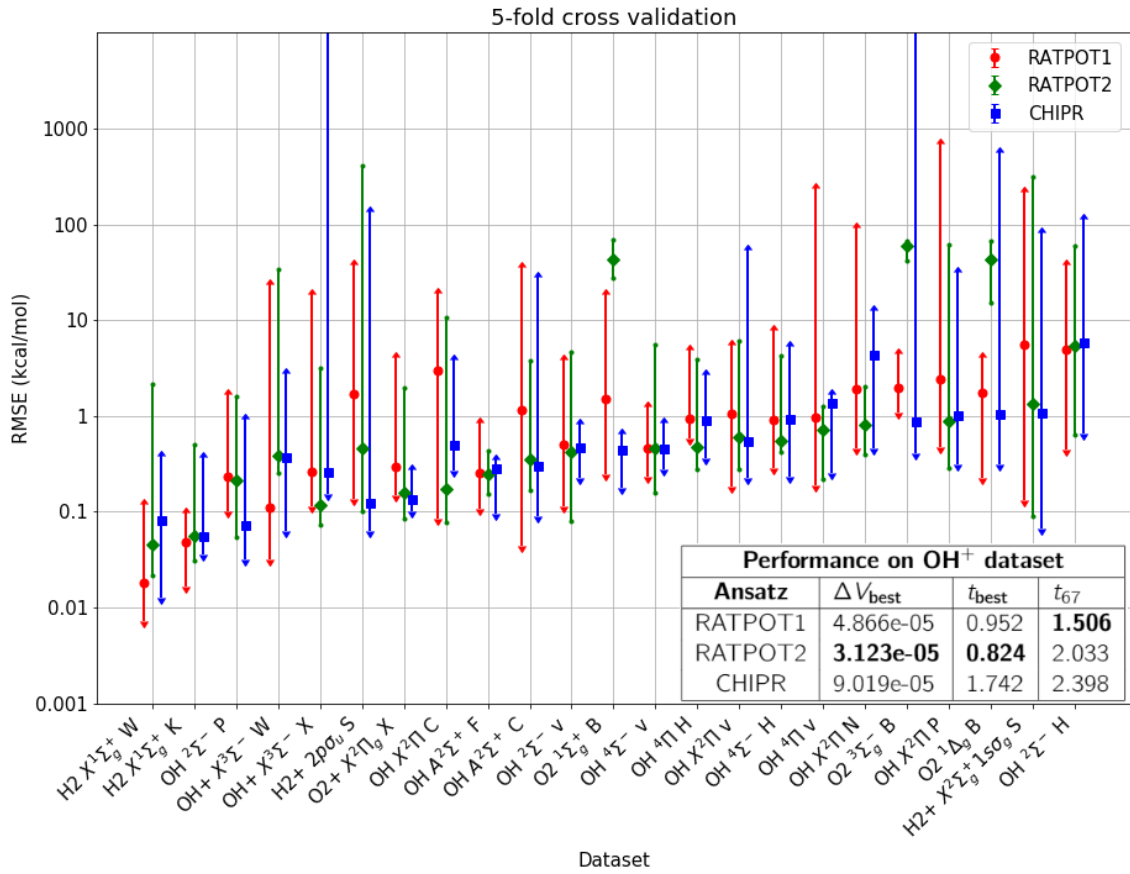


Figure 17: Benchmark of RATPOT1, RATPOT2, and CHIPR on the datasets in Section 4.1-4.3. For each dataset, the min, median and max of errors of RATPOT1, RATPOT2, and RATPOT3 are shown. On OH⁺ dataset, RATPOT2 has the lowest mean of errors and timing evaluation.

5 Potential energy surface with bonding features

Several bonding features and models are applied for Hydrogen clusters. Each feature is a basis function, each model is composed of the combinations of multiplications and summations of several basis functions with different polynomial degrees and wrapped in several forms which are then multiplied by tuning coefficients. Most of the models explained in this section were obtained from parts of an earlier research by [59], with some additional extensions.

5.1 Bonding features

5.1.1 Bonding function

The bonding function b is formed by the **Chebyshev polynomials** which defines the polynomial degree of the **bond strength**. The bond strength is a piecewise continuous smooth function which transforms each diatomic distance depending on a lower bound cut off and an upper bound cut off parameters, if the distance is below the lower bound cut off parameter then the bond strength is constant, if the distance is in between the lower and upper bound cut off parameters then the bond strength is equal to some smooth function, otherwise the bond strength is 0. In particular, the bond strength is in the form of

$$s := s(R) = \begin{cases} 1 & R < \underline{R} \\ \frac{t_0}{t+t_0} & \underline{R} \leq R \leq \overline{R}, \quad t := t(R) = \left(\frac{R^2 - \underline{R}^2}{\overline{R}^2 - R^2} \right)^c, \\ 0 & \overline{R} < R \end{cases}, \quad (209)$$

where $\underline{R}, \overline{R}, c$ are tuning constants. The bonding polynomials in terms of Chebyshev polynomials $T_i(x)$ is defined as

$$p_1(s) := s, p_d(s) := s(1-s)T_{d-2}(1-2s) \text{ for } d > 1. \quad (210)$$

Then the bonding function can be defined as

$$b_{ijd} := p_d(s) \text{ for } d = 1, 2, \dots \quad (211)$$

Bonding function b is relevant for almost all of the features used except for the **reference energy** feature. Given a total number of atoms n_{atom} — which implies degree of freedom $f_{\text{free}} := \frac{1}{2}(n_{\text{atom}}^2 - n_{\text{atom}})$ — polynomial degree d_{deg} , and total number of data points n_{data} . The bonding function is collected in an array with $(d_{\text{deg}}, f_{\text{free}}, n_{\text{data}})$ shape.

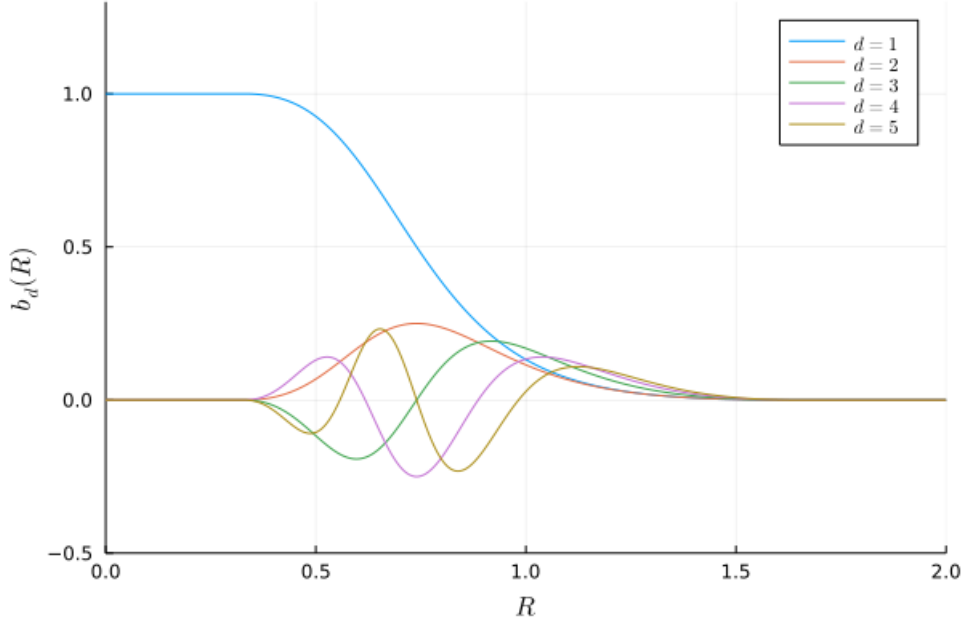


Figure 18: Bonding function curves by varying d .

5.1.2 Reference energy

The **reference energy** U is a diatomic energy function which sums each energy of the possible combinations of the diatomic distances. For each atom i , it is the sum of energies of all of the possible distances which includes atom i , this depends on the degree of freedom of the molecule. In particular, for atom i , the reference energy is

$$U[i] := \sum_{i \neq j} V_{ij}, \quad (212)$$

where V_{ij} is the diatomic energy function of atom i and j .

For example, for triatomic molecules, the reference energy of each atom is composed of 2 terms of diatomic energy function to be summed; as a concrete example, for $i = 1, 2, 3$, then $U := (U[1] := V(R_{12}) + V(R_{13}), U[2] := V(R_{21}) + V(R_{23}), U[3] := V(R_{31}) + V(R_{32}))$. Since diatomic distance is permutationally invariant, then $R_{ij} = R_{ji}$, which means the indices can be re-arranged into $\{[(1, 2), (1, 3)], [(1, 2), (2, 3)], [(1, 3), (2, 3)]\}$, this type of indexing would be more efficient given ordered vector of distances $\mathbf{R}$, since it is possible to map each tuple of distance index into a scalar index, which needs only to be computed once at the beginning. Since for each atom i , $U[i]$ sums the diatomic energies which depend on the degree-of-freedom indices, then given n_{atom} and n_{data} , U is an array with $(n_{\text{atom}}, n_{\text{data}})$ shape. This indexing rule is, in fact, applicable to any molecule with $n_{\text{atom}} > 2$.

The reference energy has various diatomic terms depending on the model (the models will be explained in later sections). For model with trainable reference pair potential, for each atom, the diatomic energy function $V_{ij} := V(R)$ is similar to the bond strength function in Section 5.1.1,

$$V(R) := \begin{cases} \infty & \text{if } R \leq R_h \\ -C(R_C^2 - R^2)^g \frac{R^2 - R_0^2}{R^2 - R_h^2} & \text{if } R_h < R \leq R_C \\ 0 & \text{else} \end{cases}, \quad (213)$$

which is a function that depends on a lower bound cut off parameter and upper bound cut off parameter; where C, R_h, R_0, R_C are tunable parameters. For model

with fixed H_2 contribution, it uses a smooth and continuous rational polynomial functions that are fitted independently on the diatomic data and then the resulting parameters are fixed during the training for the triatomic molecules, e.g., one of the RATPOTs from Section 4.4.

5.1.3 Coordination vector

For atom i , the **coordination vector** $Y[i] = (Y_1[i], Y_2[i], \dots, Y_{d_{\text{deg}}}[i])$, where

$$Y_d[i] := \sum_{j \neq i} b_{ijd}, \quad (214)$$

is the sum of bonding function for atom i where the summation indices are equal to the summation indices of the reference energy in Section 5.1.2. All components of $Y[i]$ are invariant under translation, rotation, and reflection due to each being the sum of the degree of freedom of each atom. When considering all atoms, Y is an array with $(n_{\text{atom}}, d_{\text{deg}}, n_{\text{data}})$ shape.

5.1.4 Orientation vector

The **orientation vector** $r[i] = (r_1[i], r_2[i], \dots, r_{d_{\text{deg}}}[i])$, where

$$r_d[i] := \sum_{j \neq i} b_{ijd}(x_j - x_i) \in \mathbb{R}^3. \quad (215)$$

is the sum of the multiplication between bonding function and difference of two atomic coordinates; similar to the previous section, the summation uses the indexing rule in Section 5.1.2. However this does not imply complete invariant (translation, rotation, and reflection), due to the difference of two coordinates only represents translation invariant property; e.g., let x_k be the atomic coordinate of atom k in three-dimensional Cartesian coordinate system, then $x_i - x_j \neq x_j - x_i$, which breaks the rotation and reflection invariant properties. Given n_{data} , d_{deg} and f_{free} of the bonding function, the dimension of the coordinate from x_k for all k (which in molecular system case it is always 3), and n_{atom} obtainable from the indexing routine, then r is an array with $(n_{\text{atom}}, d_{\text{deg}}, n_{\text{data}}, 3)$ shape.

Additionally, since the available *ab initio* data points throughout the literatures usually only provides the tuples of vector of distances and energy $[(\mathbf{R}_1, V_1), \dots, (\mathbf{R}_{n_{\text{data}}}, V_{n_{\text{data}}})]$, then it would be necessary to transform the distance vectors into coordinate matrices whenever the coordinate matrices are not available from the data sources.

5.1.4.1 Converting distances to coordinates

Algorithm. To calculate the Gram matrix G with entries $G_{ik} := x_i \cdot x_k$ in a coordinate system where the center of the mass is 0, proceed as follows:

$$q_{ik} := r_{ik}^2, \quad (216)$$

$$\gamma_i := \frac{1}{N} \sum_k q_{ik}, \quad (217)$$

$$\gamma := \frac{1}{2N} \sum_i \gamma_i, \quad (218)$$

$$G_{ii} = \gamma_i - \gamma, \quad (219)$$

$$G_{ik} = \frac{1}{2}(G_{ii} + G_{kk} - q_{ik}), \quad (220)$$

with N defines the number of atom points.

Given G , calculate its eigenvalues and eigenvectors. Since G is a positive semi-definite matrix, all eigenvalues are non-negative, any near 0 negative eigenvalue should be replaced by 0. Finally, multiply the corresponding 3 eigenvectors by the square root of the corresponding eigenvalues. This defines the 3 components of the coordinate vectors x_i .

Proof. The definition in (216) is required to be expanded in the terms of coordinates, it is written as

$$r_{ik}^2 = (x_i - x_k) \cdot (x_i - x_k).$$

Since the center of the mass is 0, we have

$$\sum_k x_k = 0.$$

Here and later in this section, summations run over all atom points,

$$k = 1, \dots, N.$$

Using this, we find

$$\begin{aligned} \gamma_i &= \frac{1}{N} \sum_k q_{ik} = \frac{1}{N} \sum_k r_{ik}^2, \\ &= \frac{1}{N} \sum_k (x_i - x_k) \cdot (x_i - x_k), \\ &= \frac{1}{N} \left(\sum_k x_i^2 - \sum_k 2x_i \cdot x_k + \sum_k x_k^2 \right), \\ &= \frac{1}{N} \left(N(x_i^2) - 2x_i \cdot \sum_k x_k + \sum_k x_k^2 \right), \\ &= x_i^2 + \frac{1}{N} \sum_k x_k^2, \\ \gamma &= \frac{1}{2N} \sum_i \gamma_i, \\ &= \frac{1}{2N} \sum_i x_i^2 + \frac{1}{2N^2} N \sum_k x_k^2, \\ &= \frac{1}{N} \sum_k x_k^2. \end{aligned}$$

Therefore

$$G_{ii} = \gamma_i - \gamma = x_i^2 + \frac{1}{N} \sum_k x_k^2 - \frac{1}{N} \sum_k x_k^2 = x_i^2$$

validates (219). Using this we find

$$G_{ik} = \frac{1}{2}(x_i^2 + x_k^2 - (x_i - x_k) \cdot (x_i - x_k)) = x_i \cdot x_k$$

which validates (220).

5.1.5 Gram matrix

For each atom i , **Gram matrix**

$$G[i]_{d_1 d_2} := r_{d_1}[i] \cdot r_{d_2}[i], \quad (221)$$

is a symmetric and positive semidefinite matrix formed by the dot product of the orientation vectors. As a feature, it is invariant to translation, rotation, and reflection. Given n_{atom} , n_{data} , and orientation vectors with maximum polynomial degree of d_{deg} , then G is collected in an array with 4 indices with $(n_{\text{atom}}, d_{\text{deg}}, d_{\text{deg}}, n_{\text{data}})$ shape.

5.1.6 Neighbourhood matrix

In the case of triatomic systems, the neighbourhood matrix

$$\Theta[i]_{d_1 d_2} := \frac{1}{2} \sum_{\substack{j, k \neq i, \\ j \neq k}} b_{ij d_1} b_{ik d_2} (R_{ij}^2 + R_{ik}^2 - R_{jk}^2), \quad (222)$$

is the Gram matrix $G[i]$ without the two-body terms, hence the neighbourhood matrix only represents the three-body terms. In the program implementation, the shape of Θ is equal to the shape of G .

5.2 Models

There are two types of model employed for H_3 molecule, one model makes use of trainable reference pair potential, and the other uses fixed H_2 contribution. All of the models employ the previously mentioned features (basis functions) as monomials. Each feature is collected in

$$\Phi = \Phi(U, Y, H), \quad (223)$$

where H is chosen either G or Θ for each model. For example, for the vector of basis functions up to degree 1: $\Phi(1) = (U, Y_1)$, for basis functions up to degree 2: $\Phi(2) = (UY_1, Y_1^2, Y_2, G_{11})$; then these vectors of basis functions of different degrees of polynomials are arranged in a contiguous way within a single vector

$$\Phi = (U, Y_1, UY_1, Y_1^2, Y_2, G_{11}, \dots). \quad (224)$$

For H_3 molecule, there exists a total of 59 basis functions for all of the combinations of monomials up to degree 5. Implementation wise, given n_{atom} , n_{data} , and total number of basis functions n_{basis} , the shape of array Φ is $(n_{\text{atom}}, n_{\text{basis}}, n_{\text{data}})$.

5.2.1 Model with trainable reference pair potential

There are a total of 7 tuning parameters from the bonding function and the U monomial combined, collected in a vector $\rho := (\rho_1, \rho_2, \dots, \rho_7)$, where $\rho_2 \leq \rho_3 \leq \dots \leq \rho_7$ relations need to be satisfied.

For atom i , the atomic energy term is a combination of three partial energy terms,

$$\epsilon_0[i] = A[i] - \sqrt{B[i] + C[i]}, \quad (225)$$

where A is

$$A[i] = \frac{\sum_k \theta_k \Phi_k[i]}{1 + \left(\sum_k \theta'_k \Phi_k[i] \right)^2}, \quad (226)$$

and B, C are

$$T_0[i] = \frac{\left(\sum_k \theta_k \Phi_k[i] \right)^2}{1 + \left(\sum_k \theta'_k \Phi_k[i] \right)^2}, \quad (227)$$

where θ and θ' are the tuning coefficients; and Φ is the enumeration of basis functions in (224). The total energy is

$$V_{\text{tot}} = \sum_i \epsilon_0[i], \quad (228)$$

for i that runs over all atoms in the molecule. The potential energy of the molecular system is the sum of all atomic energies, each atomic energy is composed of 3 different partial energy terms, and each partial energy term is in the form of rational potential where each of the numerator and denominator has n_{basis} tuning coefficients; in total there are $6 \times n_{\text{basis}}$ tuning coefficients for this model. For H_3 molecule, there is a maximum of 59 basis functions (where $H := G$ in (223), up to 5 degree of monomial combinations), this results in $6 \times 59 + 7 = 361$ tuning parameters.

5.2.2 Model with fixed diatomic partial model

The difference with the model in Section 5.2.1 is that the diatomic model in the form of U here is pre-trained, and the tuning coefficients are fixed during the training of the $n \geq 3$ molecules. This simplifies the computation in terms of speed and model complexity that needs to be optimized.

5.3 Objective function

The objective function has the form of nonlinear least squares

$$f(\theta) = \sum_l (V_{\text{tot}}(\theta)[l] - V_{\text{ab initio}}[l])^2 \quad (229)$$

where θ is the collection of tuning coefficients, l runs over the training molecules, and $V_{\text{ab initio}}$ refers to the actual energies. The objective function considers the possibility of fitting multiple different datasets at once, this is compensated by the addition of two tuning parameters for each nonlinear least squares associated to a particular dataset: energy shift μ and the order of accuracy σ .

In case of the model with trainable pair potential in Section 5.2.1, the first 7 tuning parameters ρ are slightly changed for regularization purpose and to compensate

the constraints, they become $\pi := (\pi_1 = \log(\rho_1)/20, \pi_2 = \log(\rho_2)/20, \pi_3 = \sqrt{\rho_3}, \pi_4 = \sqrt{\rho_4 - \rho_3}, \pi_5 = \sqrt{\rho_5 - \rho_4}, \pi_6 = \sqrt{\rho_6 - \rho_5}, \pi_7 = \sqrt{\rho_7 - \rho_6})$ as the input of the optimization routine; which is then inverted into $\rho := (\exp(20\pi_1), \exp(20\pi_2), \pi_3^2, \pi_4^2 + \rho_3, \pi_5^2 + \rho_4, \pi_6^2 + \rho_5, \pi_7^2 + \rho_6)$ for the potential evaluation. We want to emphasize that ρ is computed from the pre-training process of the diatomic molecules, independent of θ . For model with fixed H_2 , the tuning coefficients of partial energy terms are scaled by the expected magnitude of Φ .

5.4 Extension: primitive bonding features

In this section, additional ansatz for intermediate values for computing the total potential energy are explained.

5.4.1 Coordination function

A spline-like C^2 function if $t^2 < 1$ and 0 otherwise, in the form of

$$z(t) = a - bt - ct^5 + dt^3, \quad (230)$$

where $t \in \mathbb{R}$. The coordination function is formed by fulfilling these conditions:

$$\begin{aligned} z(-1) &= 1, \\ z(1) &= 0, \\ z'(\pm 1) &= 0, \\ z''(\pm 1) &= 0. \end{aligned} \quad (231)$$

In order to obtain the correct a, b, c, d , the conditions in (231) need to be expanded, which results in a linear system of equations

$$\begin{aligned} z(-1) &= a + b + c - d = 1, \\ z(1) &= a - b - c + d = 0, \\ z'(\pm 1) &= -b - 5c + 3d = 0, \\ z''(\pm 1) &= 20c - 6d = 0. \end{aligned} \quad (232)$$

By solving (232) and writing the coefficients explicitly for the $t^2 < 1$ condition, the coordination function is

$$z(t) = \begin{cases} 1, & \text{if } t \leq -1 \\ 0, & \text{if } t \geq 1 \\ \frac{1}{2} - t \left(\frac{15}{16} + s \left(\frac{3}{16}s - \frac{10}{16} \right) \right), & \text{if } s = t^2 < 1 \end{cases} \quad (233)$$

(233) can be verified by differentiating $z(t)$ directly and expanding the second last condition in (231)

$$\begin{aligned} z'(t) &= \frac{d}{dt} \left[\frac{1}{2} - t \left(\frac{15}{16} + t^2 \left(\frac{3}{16}t^2 - \frac{10}{16} \right) \right) \right] = -\frac{15}{16}(t + t^4 - 2t^2), \\ z'(\pm 1) &= 0. \end{aligned} \quad (234)$$

5.4.2 Bump functions

The bump functions, has the form

$$f_k(t) = (1 - (t - k)^2)_+^3 \quad (235)$$

where $k \in \mathbb{Z}$, is a C^2 function everywhere except when $t \geq k + 1$ or $t \leq k - 1$, in which f_k is 0. The transformed $r \in \mathbf{R}$ has the form of

$$t = c_{xy}(r^2 - r_{xy}^2), \quad (236)$$

where

$$c_{xy} = \frac{N}{\bar{r}_{xy}^2 - r_{xy}^2} \quad (237)$$

is a coefficient which depends on the largest atomic distance $\bar{r}_{xy}$ and smallest atomic distance r_{xy} . While r_{xy} itself denotes the equilibrium distance of xy atomic pair (i.e., the closest distance where $dV/dr \approx 0$), for example, $r_{\text{HH}} \approx 1.4172946$ Bohr.

From (236) we can define the upper bound and lower bound of t as

$$\begin{aligned} \underline{t} &= c_{xy}(r_{xy}^2 - r_{xy}^2), \\ \bar{t} &= c_{xy}(\bar{r}_{xy}^2 - r_{xy}^2), \end{aligned} \quad (238)$$

when $\bar{t}$ and $\underline{t}$ are rounded to the nearest integers, this tells

$$N = \bar{t} - \underline{t}, \quad (239)$$

which is a hyperparameter that describes the number of the primitive functions needed to approximate the associated pair-potential. The list of required k for the total number of $N - 1$ bump functions f_k can be obtained by

$$k \in \mathbb{Z} \cap \underline{t} < k < \bar{t}. \quad (240)$$

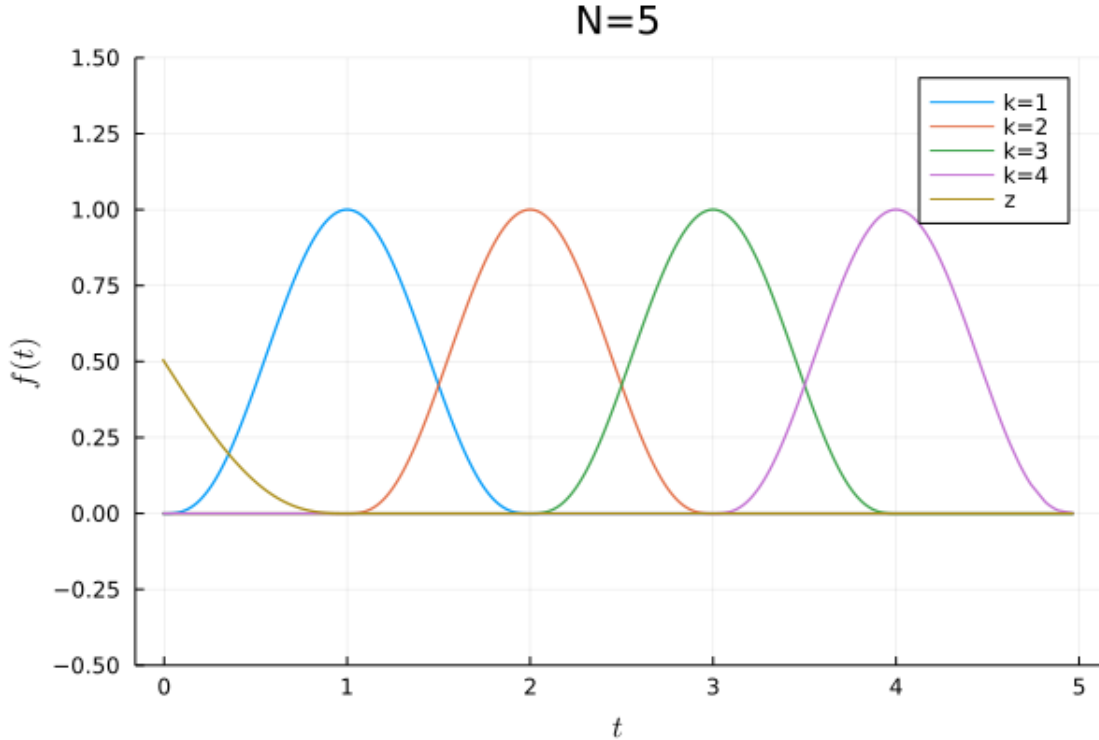


Figure 19: $z(t)$ and $f_k(t)$ for $k = 1, 2, \dots, 4$ computed from $\mathbf{R}_{\cdot,3}$ of H_3 dataset.

5.4.3 Bump functions for linear system

We present bump function ansatz with a total of $2N + 2$ tuning parameters for solving or pretraining the diatomic potential energy surface. First, the interatomic distance is scaled by

$$\rho := r/r_{xy},$$

where r_{xy} is the equilibrium distance of atom pair with types x and y . For all scaled distances data, each one is re-scaled by

$$q := \frac{N}{1 + \rho} \in (0, N]. \quad (241)$$

The basic bump formula is

$$h_k := [1 - (q - k)^2]_+^3. \quad (242)$$

Then we have the linear combination of the bumps:

$$u(q) = \sum_{k=0}^N (\theta_k(q - k) + \theta_{k+N+1}) h_k, \quad (243)$$

with the sum

$$w(q) = \sum h_k. \quad (244)$$

Since most h_k vanish

$$\begin{aligned} i &:= \lceil q \rceil \geq 1, \\ \epsilon &:= i - q \in [0, 1) \implies q = i - \epsilon \in (i - 1, i], \end{aligned} \quad (245)$$

then we define

$$\begin{aligned} \alpha &:= (\epsilon(2 - \epsilon))^3 \\ \beta &:= (1 - \epsilon^2)^3, \end{aligned} \quad (246)$$

thus we have the simplified bump function

$$h_k = \begin{cases} \alpha & \text{if } k = i - 1, \\ \beta & \text{if } k = i, \\ 0 & \text{otherwise} \end{cases}, \quad (247)$$

the simplified linear combination of the bumps

$$u(q) = (\theta_i(q - i + 1) + \theta_{i+N})\alpha + (\theta_i(q - i) + \theta_{i+N+1})\beta, \quad (248)$$

and the simplified bump sum

$$w(q) = \alpha + \beta. \quad (249)$$

Now we can compute the basic features as

$$x := h_k(q)/w(q), \quad (250)$$

$$y := (q - k)h_k(q)/w(q). \quad (251)$$

These features can also be used directly as a linear system of equations to solve diatomic systems, as the following:

$$\begin{aligned} u(q) &= \sum_k \left(\frac{h_k(q)}{w(q)} \theta_k + \frac{(q-k)h_k(q)}{w(q)} \theta_{k+N+1} \right) \\ &= A\theta = \sum_k A_{ik} \theta_k, \text{ if } q = q_i. \end{aligned} \quad (252)$$

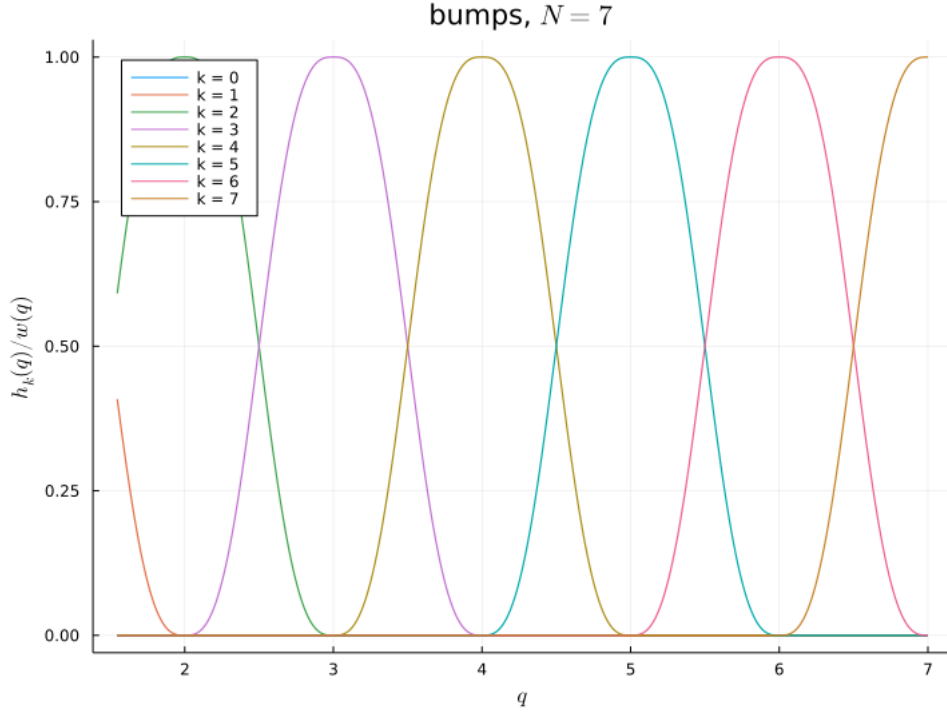


Figure 20: Bumps from $h_k(q)/w(q)$ for $N = 7$.

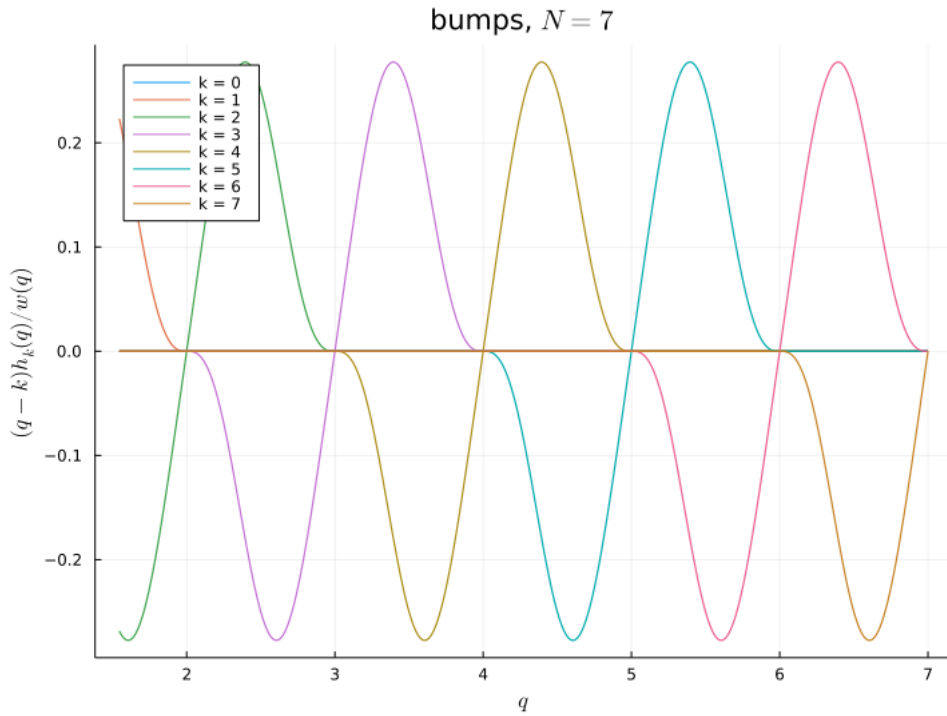


Figure 21: Bumps from $(q-k)h_k(q)/w(q)$ for $N = 7$.

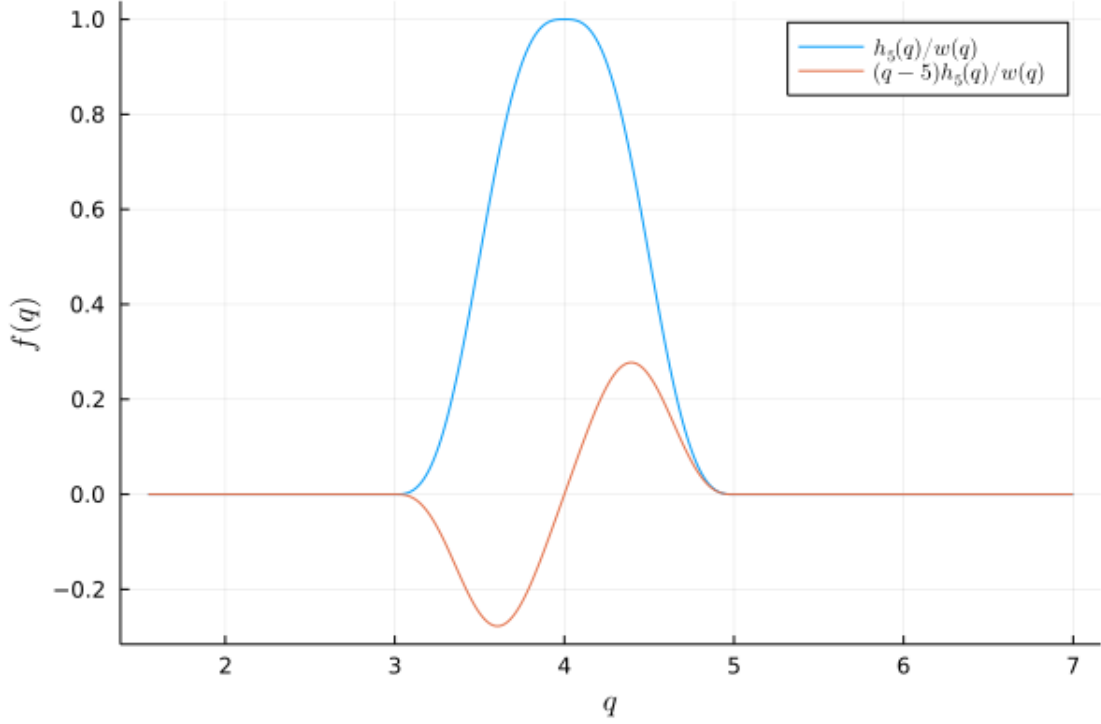


Figure 22: Bumps for x and y when $k = 5$.

5.5 Notes on implementation

To compute the bonding features for fitting (function evaluator), a wrapper function which takes in all of the tuning parameters alongside the hyperparameters, and computes all of the features in a certain order, which then returns the energy value at the final step is required. Suppose

```
V = f_pot_bond(R, X, rho, sigma)
```

is the function to evaluate the energy given a matrix of atomic distances $\mathbf{R} \in \mathbb{R}^{n_{\text{data}} \times f_{\text{free}}}$, array of atomic coordinates $\mathbf{X}$ with the shape of $(n_{\text{data}}, n_{\text{atom}}, n_{\text{dim}})$, parameter vector ρ , and hyperparameter vector σ ; with a vector of energy $\mathbf{V} \in \mathbb{R}^{n_{\text{data}}}$ as the returned quantity. The features (and sub features) within the function are computed in this order:

- Reference energy U .
- Bond strength b (sub feature for coordination vector Y).
- Coordination vector Y .
- Difference array Δ (sub feature for orientation vector r).
- Orientation vector r .
- Gram matrix G .
- Array of basis functions Φ .
- Partial (atomic) energy term $\epsilon[i]$, which finally sums into the energy value.

Additionally, index computation is done pre-function evaluation once for each atomic system, this indexer is passed into the function evaluation as an input. Suppose that the indexing function is

```
indexer = atom_indexer(n_atom)
```

this returns the n -body indices of molecular system containing n -atoms, let it be stored in a matrix $D \in \mathbb{Z}^{n_{\text{atom}} \times n_{\text{atom}} - 1}$. For example, with $n_{\text{atom}} = 4$, the indexer has entries

```
indexer = [
            [1, 2, 3],
            [1, 4, 5],
            [2, 4, 6],
            [3, 5, 6]
          ]
```

this is useful for vectorization of any arithmetic operations which make use the degree of freedom indices given an ordered vector of distances $\mathbf{R}_k \in \mathbf{R}, k = 0, 1, 2, \dots, n_{\text{data}}$, for example, for 4-body terms summation (using the indexer in the example above): $V_k = \sum_{j \neq i} \mathbf{R}_{k,i,j}$ can be represented by

```
V = sum(R[:, indexer])
```

This indexer allows computations on vectors of distances without needing to transform matrix of distances into an array of adjacency matrices with 0 at the diagonals (in graph-terms) with $(n_{\text{data}}, n_{\text{atom}}, n_{\text{atom}})$ shape, which in turn reduces the space complexity. The indexer is used in reference energy U , coordination vector Y , and orientation vector r .

Reference energy U . First the function

```
U = U_ref_energy(R, C, R_h, R_C, R_0, g)
```

decides the diatomic energy value within certain ranges for each element of $\mathbf{R}$ matrix. This is done using a sub function containing simple switch-statement

```
if R <= R_h:
    return infinity
else if R_h < R <= R_C:
    return some smooth function f(R, R_0, g)
else:
    return 0
```

where $C, R_h, R_C, R_0 \in \rho$ and $g \in \sigma$, then let $V^{\text{diatomic}} \in \mathbb{R}^{n_{\text{data}} \times f_{\text{free}}}$ be the matrix containing all of the returned values from above sub function. Next, the degree of freedom of each $V_k^{\text{diatomic}} \in \mathbb{R}^{f_{\text{free}}}$ is summed by using the indexer (essentially reducing the f_{free} index to n_{atom}), hence the array U has $(n_{\text{atom}}, n_{\text{data}})$ shape.

Bond strength b . The bond strength function

```
b = bond_strength(R, d_deg, n_atom, R_up, R_m, R_low, e)
```

computes the bond strength of each element of $\mathbf{R}$ matrix, where $\overline{R}, R_m, \underline{R} \in \rho$ and $d_{\text{deg}}, e \in \sigma$. First t_0 is obtained from a function which computes a smooth functional (polynomial with at least degree 2 in each numerator and denominator)

```
t_0 = f_t_R(R_m, R_up, R_low, e)
```

then matrix $t \in \mathbb{R}^{n_{\text{data}} \times f_{\text{free}}}$ is computed using the same function by passing the matrix $\mathbf{R}$ at the R_m argument position. Both t_0 and t are used for the bond strength computation, which is essentially a similar switch sub function as the reference energy U (notice the slight differences of the inequality relations)

```
if R < R_low:
    return 1
else if R_low <= R <= R_up:
    return t_0 / (t + t_0)
else if R_up < R:
    return 0
```

then the result is stored in $s \in \mathbb{R}^{n_{\text{data}} \times f_{\text{free}}}$ matrix. On the next step, the bond strength s will need to be wrapped in the terms of polynomials so that s can have different degree of powers, in order to do this, the polynomials are evaluated recursively in terms of Chebyshev polynomials

```
b = Chebyshev_pol(d_deg, s)
```

hence b is an array with $(d_{\text{deg}}, n_{\text{data}}, f_{\text{free}})$ shape.

Coordination vector Y . The function for coordination vector Y simply computes the sum of the bond strength b by using the indexer in the same manner with the second half of U computation. Hence Y has the shape of $(n_{\text{atom}}, d_{\text{deg}}, n_{\text{data}})$.

Difference array Δ . The delta array Δ with shape $(n_{\text{data}}, f_{\text{free}}, n_{\text{dim}})$ (specifically in molecular systems, $n_{\text{dim}} = 3$ always) contains $\mathbf{X}_{:,j} - \mathbf{X}_{:,i}, i < j$, where $\mathbf{X}$ with the shape of $(n_{\text{data}}, n_{\text{atom}}, n_{\text{dim}})$ is the matrix containing molecular coordinates. The loop goes over the n_{atom} index hence this is sort of the "inverse" of the indexer, since this transforms n_{atom} into f_{free} .

Orientation vector r . To compute the orientation vector r , first the (elementwise) array multiplication between the $(n_{\text{data}}, f_{\text{free}})$ indices of b and $(n_{\text{data}}, f_{\text{free}}, 3)$ indices of Δ needs to be computed for $d = 1, 2, \dots, d_{\text{deg}}$, then let the result be stored in an array η with shape $(d_{\text{deg}}, n_{\text{data}}, f_{\text{free}}, 3)$. Finally η is summed using the indexer rule, which the final result is stored in array r with the shape of $(n_{\text{atom}}, d_{\text{deg}}, n_{\text{data}}, 3)$.

Gram matrix G . For each atom $i = 1, 2, \dots, n_{\text{atom}}$, for each $d_1 = 1, 2, \dots, d_{\text{deg}}$, and for each $d_2 = 1, 2, \dots, d_{\text{deg}}$:

```
G[i] = sum(r[i][d_1] * r[i][d_2], axis=-1)
```

this means that the elementwise array multiplication is done on the $(n_{\text{data}}, 3)$ indices, then the last index is summed (denoted by $\text{axis} = -1$ argument), hence the last index disappears, this represents the dot product between vectors. In the end G is an array with $(n_{\text{atom}}, d_{\text{deg}}, d_{\text{deg}}, n_{\text{data}})$ shape.

Array of basis functions Φ . The array Φ with shape $(n_{\text{atom}}, n_{\text{basis}}, n_{\text{data}})$, consists of basis functions which are composed of the feature array which have been calculated previously, simply

```
phi = f_phi(U, Y, G)
```

for each degree, all possible polynomials are enumerated, each polynomial is the combination of each feature array, for example

```
# degree 1:
phi[:, 0] = U;
phi[:, 1] = Y[:, 0];

# degree 2:
phi[:, 2] = U*Y[:, 0];
phi[:, 3] = Y[:, 0]**2; phi[:, 4] = Y[:, 1]; phi[:, 5] = G[:, 0,0];

#degree 3:
...
```

which shows that $\Phi = (U, Y_1, UY_1, Y_1^2, Y_2, G_{1,1}, \dots)$.

Atomic energy terms $\epsilon[i]$. The atomic energy terms $\epsilon[i]$ are the sums of partial energy terms, where the partial energy terms are smooth rational functionals which depend on the tuning parameter matrices $A, B, C \in \mathbb{R}^{2 \times n_{\text{basis}}}$. Technically each element of each tuning parameter vectors are multiplied by Φ_k , in simpler terms

```
f_partial_energy(E, Phi):
    V_part_mat = matrix((2,n_basis), type="real")
    for k=0,1,...,n_basis:
        V_part_mat[0][k] = E[0][k]*Phi[i][k]
        V_part_mat[1][k] = E[1][k]*Phi[i][k]
    V_part = f_smooth_rat(V_part_mat) # some smooth rational function
    return V_part
```

where A, B , or C is an input argument in place of E . Then the partial energies are combined in a certain way, for example $\epsilon[i] = V_{\text{part}}(A) - \sqrt{V_{\text{part}}(B) - V_{\text{part}}(C)}$. Finally the total energy of the system is $V_{\text{tot}} = \sum_{i=1}^{n_{\text{atom}}} \epsilon[i]$.

5.6 Numerical experiments

5.6.1 Feature statistics

The statistics of U, Y, G features described within this section are displayed in Table 4 and Table 5. The features are used to compute the energy of 6032 data points of H_3 dataset. In general, low standard deviation is desirable for fitting since this may imply more general feature.

Table 4: Statistics of each feature per atom index using pre-optimization parameters.

feature[atom]	max	min	mean	std
$U[1]$	2.481150e+24	7.461384e+22	6.254963e+23	5.593566e+23
$U[2]$	2.435484e+24	4.340160e+22	5.548411e+23	5.030545e+23
$U[3]$	1.682283e+24	5.973577e+21	2.307342e+23	2.703311e+23
$Y[1]$	2.000000e+00	-2.234326e-02	3.997134e-01	7.999023e-01
$Y[2]$	2.000000e+00	-8.430624e-02	3.992064e-01	7.997078e-01
$Y[3]$	2.000000e+00	-1.055023e-01	3.989212e-01	7.996413e-01
$G[1]$	7.428453e-02	0.000000e+00	3.107268e-03	1.070428e-02
$G[2]$	1.743001e+00	7.438274e-16	3.483051e-02	1.402622e-01
$G[3]$	2.486132e+00	2.166612e-15	5.631022e-02	2.146574e-01
$\Phi[1]$	3.969833e+25	-1.901475e+22	3.286893e+23	1.990261e+24
$\Phi[2]$	3.896758e+25	-2.559325e+22	2.923480e+23	1.776411e+24
$\Phi[3]$	2.691653e+25	-7.797651e+20	1.213176e+23	8.457775e+23

Table 5: Statistics of each feature per atom index using post-optimization parameters, with RMSE = 0.069 across all H_3 data.

feature[atom]	max	min	mean	std
$U[1]$	1.450460e-276	-2.679209e-278	3.114115e-277	0.000000
$U[2]$	1.422420e-276	-4.622908e-278	2.679955e-277	0.000000
$U[3]$	9.601608e-277	-6.980936e-278	6.897668e-278	0.000000
$Y[1]$	1.999660e+00	-4.937911e-01	3.189007e-01	0.644286
$Y[2]$	1.999115e+00	-4.923274e-01	2.960542e-01	0.580598
$Y[3]$	1.999115e+00	-4.978102e-01	2.358900e-01	0.533017
$G[1]$	4.256561e+00	0.000000e+00	3.034012e-01	0.463637
$G[2]$	3.609277e+00	5.088141e-06	3.312816e-01	0.419666
$G[3]$	5.362877e+00	2.099490e-06	4.990154e-01	0.741347
$\Phi[1]$	3.197281e+01	-1.031954e+00	8.880903e-01	3.839208
$\Phi[2]$	3.192923e+01	-1.024145e+00	6.708816e-01	3.008128
$\Phi[3]$	3.192923e+01	-1.031435e+00	6.205682e-01	2.927580

5.6.2 Fitting H_3 , H_4 , and H_5 molecules

We conducted fitting experiments using H_3 , H_4 , and H_5 molecular datasets. The datasets are obtained from [59]. Given n amount of atoms for each molecule, then the molecule would have $n(n-1)/2$ interatomic distances. In this case, the datasets have 3, 6, 10 columns of initial features (the interatomic distances) for H_3 , H_4 , and H_5 respectively. The H_3 , H_4 , and H_5 datasets have 6032, 2508, and 2799 data points in total. The ratio for training and test data is 80%:20% (no validation).

The Levenberg-Marquardt (LM) algorithm from `LsqFit.jl` [77] is used for the training process, with the maximum of 1k loops. 100 restarts of multistart mode is used to avoid undesirable minima during training. In the `LsqFit.jl`, we supply the LM algorithm with the first derivative of function evaluation (228) with respect to the tuning parameters computed by LM algorithm supplied by reverse mode AD.

Here we compare the speed of the function evaluation of LM algorithm when supplied by AD against when the standard finite differences (FD) is used. 1k loop of LM algorithm on 6k molecules of H_3 data is used as the benchmark. FD results

in 523.82s, while reverse AD results in 129.02s, which is more than 4x speed up factor, on top of provably higher accuracy of AD (since AD gives exact derivatives, not approximation). Thus we proceed with AD for our method.

The current best obtained RMSE for each dataset with comparison to CHIPR [44] (we directly used the CHIPR-4.0 Fortran code to do the fitting) is summarized in Table 6. The best configuration is obtained by using (197) for the pre-trained pair potential to construct the reference energy U , which is also used for the bond strength in terms of Chebyshev polynomials for the coordination vector Y ; Gram matrices G is also used for the basis functions Φ .

Table 6: Current best obtained RMSE (in Hartree). The method in this section is dubbed as BondPOT. The "-" indicates that the program did not converge (failed to give reasonable accuracy), while "N/A" indicates that the program do not have the feature to compute the corresponding dataset.

Data	BondPOT	CHIPR
H ₃ train	7e−4	7e−4
H ₃ test	3e−3	8e−4
H ₄ train	1e−6	-
H ₄ test	4e−6	-
H ₅ train	1e−2	N/A
H ₅ test	4e−2	N/A

6 Semi supervised model based on Shepard interpolation

We introduce a new method aimed at the efficient solution of partially supervised continuous prediction problems, where a large set of unlabelled data points is given and a continuous quantity must be predicted for which only a tiny set of values may be queried. The method, called **ROSEMI** for **robust Shepard model for interpolation** is a modified version of the well-known Shepard interpolation procedure [78], adapted to meet the requirements needed for the above task.

6.1 Robust Shepard interpolation

ROSEMI involves several novelties compared to the method of moving least squares [79], which is the traditional modification of Shepard’s method:

- ROSEMI uses ridge functions for the numerator functions rather than quadratic polynomials.
- ROSEMI solves a single sparse linear least squares or L^1 problem rather than fitting of local approximations.
- ROSEMI produces without additional cost automatic estimates of the uncertainty of the predictions, leading to a very efficient active learning strategy.

The application of ROSEMI to molecular property prediction features additional novelties compared to such applications of previous modified Shepard methods:

- Our use of ROSEMI avoids the coordinate redundancy problem [80], by using models based on molecular fingerprints rather than internal coordinates. Any sufficiently unique set of numerical fingerprints may be used.
- ROSEMI does not produces holes in potential energy surfaces [81].
- Our use of ROSEMI does not depend on explicit models of bonding, hence can be applied in molecular dynamics studies of chemical reactions and dissociation processes.

6.1.1 Generalized Shepard model

In this section, we consider approximation functions of the **generalized Shepard form**

$$V_K(w) := R_K(w)/S_K(w) \quad \text{for } w \in \mathbb{R}^m \setminus \{w_k \mid k \in K\}, \quad (253)$$

where

$$R_K(w) := \sum_{k \in K} \frac{V_k(w)}{D_k(w)}, \quad S_K(w) := \sum_{k \in K} \frac{1}{D_k(w)} \quad (254)$$

using appropriate **numerator functions** $V_k(w)$ and nonnegative **denominator functions** $D_k(w)$ vanishing only at a single point $w = w_k$,

$$D_k(w) \geq 0, \quad \text{with equality iff } w = w_k. \quad (255)$$

Although (253) leaves $V_K(w)$ undefined for $w = w_k$, we may extend V_k to a smooth function on $\mathbb{R}^m$ by defining

$$V_K(w) := \begin{cases} V_k(w_k) & \text{if } w = w_k \text{ with } k \in K, \\ R_K(w)/S_K(w) & \text{otherwise.} \end{cases}$$

To guarantee sufficient smoothness for V_k , we assume in the following that the $V_k(w)$ and the $D_k(w)$ are twice continuously differentiable functions of w . We may interpolate a given set of pairs (w_k, E_k) ($k \in K$) by choosing numerator functions $V_k(w)$ satisfying the **interpolation condition**

$$V_k(w_k) = E_k. \quad (256)$$

This interpolation property was first observed in a special case by [78]. His approach was generalized by [82] and [83] to allow arbitrary numerator functions. The **moving least squares** method uses as numerator functions quadratic functions $V_k(w)$ whose coefficients are fitted to function values close to w_k by weighted least squares minimization with a distance-dependent weight; see [79, 84]. For an application to PES fitting see [85].

6.1.2 Interpolating equilibrium geometries

We can guarantee the interpolation condition (256) by representing the numerator functions by an ansatz of the form

$$V_k(w) = E_k + \sum_b \theta_{kb} (\phi_b(w) - \phi_b(w_k)) \quad (257)$$

with some coefficient matrix θ and n_B arbitrary basis functions $\phi_b(w)$ indexed by $b \in B$.

For equilibrium geometries the forces vanish. To achieve this we now need basis functions $\phi_b(w)$ with known derivatives

$$\phi'_b(w) = \frac{d\phi_b}{dw} \in \mathbb{R}^{1 \times m}.$$

We then replace the interpolation condition (257) by the modified condition

$$V_k(w) = E_k + \sum_b \theta_{kb} \phi_{kb}(w), \quad (258)$$

with the molecular fingerprint w_k of molecule k and

$$\phi_{kb}(w) := \phi_b(w) - \phi_b(w_k) - \phi'_b(w_k)(w - w_k) \quad \text{for } k \in K, b \in B. \quad (259)$$

Since $\phi_{kb}(w)$ has a vanishing derivative at $w = w_k$, the same holds for $V_k(w)$. By (262), this implies that $V_K(w)$ also has a vanishing derivative at $w = w_k$.

The estimated forces acting on a geometry x are given by the gradient of $V_K(w(x))$ with respect to x . By the chain rule, the force $\mathbf{F}_i(x)$ applying to the atom i of a geometry x is given by

$$\mathbf{F}_i(x) := \frac{\partial}{\partial x_i} V(w(x)) = V'(w(x)) \frac{\partial w(x)}{\partial x_i},$$

hence the approximation to these forces obtained by using V_K in place of V vanish at $w = w_k$, as required. Note that derivatives with respect to the geometry coordinates are needed to calculate the approximation to the forces, but not to calculate $V_K(w)$ and the $\Delta_{jK}(w)$. This is important since it implies that a lot of time can be saved during the training by precomputing and storing the fingerprints for all molecules prior to the training, which never refers to the original coordinates.

6.1.3 Interpolation properties

If the $D_k(w)$ are quadratic then (255) is equivalent to

$$D_k(w) = (w - w_k)^T G_k (w - w_k) = \|B_k(w - w_k)\|_2^2 \quad (260)$$

with symmetric positive definite matrices $G_k = B_k^T B_k$; in general, there may be higher order correction terms. In particular,

$$S(w) = \frac{1}{D_j(w)} + O(1) \text{ for } w \approx w_j.$$

Taking differences we find

$$V_K(w) - V_j(w) = \frac{1}{S_K(w)} \sum_{k \neq j} \frac{V_k(w) - V_j(w)}{D_k(w)},$$

hence

$$V_K(w) = V_j(w) + D_j(w) \sum_{k \neq j} \frac{V_k(w) - V_j(w)}{D_k(w)} + O(D_j(w)^2).$$

In particular,

$$V_K(w) = V_j(w) + O(\|w - w_j\|_2^2) \quad \text{for } w \approx w_j, \quad (261)$$

$$V_K(w) = V_j(w) + \gamma_j(w_j) D_j(w) + o(\|w - w_j\|_2^2) \quad \text{for } w \approx w_j, \quad (262)$$

where

$$\gamma_j(w_j) := \sum_{k \neq j} \frac{V_k(w_j) - V_j(w_j)}{D_k(w_j)}.$$

(253)–(254) may be viewed as defining a weighted average of the $V_k(w)$ with w -dependent weights $1/D_k(w)$.

(262) implies that arbitrary twice continuously differentiable functions of w can be approximated near w_j with an accuracy of $o(\|w - w_j\|_2^2)$ if this is the case for the $V_j(w)$. This is used in the moving least squares approach to interpolation by choosing the $V_j(w)$ as local quadratic approximations. If we delete an entry j from K , the effect on the functions defined above is

$$R_{K \setminus \{j\}}(w) = R_K(w) - \frac{V_j(w)}{D_j(w)}, \quad S_{K \setminus \{j\}}(w) = S_K(w) - \frac{1}{D_j(w)},$$

and $D_j(w)S_K(w) - 1 = D_j(w)S_{K \setminus \{j\}}(w) > 0$ when $|K| > 2$. Thus

$$V_{K \setminus \{j\}}(w) = \frac{R_{K \setminus \{j\}}(w)}{S_{K \setminus \{j\}}(w)} = \frac{D_j(w)R_K(w) - V_j(w)}{D_j(w)S_K(w) - 1}.$$

Therefore

$$\Delta_{jK}(w) := V_K(w) - V_{K \setminus \{j\}}(w) = \frac{R_K(w)}{S_K(w)} - \frac{D_j(w)R_K(w) - V_j(w)}{D_j(w)S_K(w) - 1}. \quad (263)$$

Since $R_K(w) = V_K(w)S_K(w)$ we conclude that

$$\Delta_{jK}(w, \theta) = \frac{V_K(w) - V_j(w)}{D_j(w)S_K(w) - 1}. \quad (264)$$

Here we made on the left hand side the parameter vector θ explicit, on which the V_k and hence all other expressions used depend. In essence, $\Delta_{jK}(w, \theta)$ represents how large the **deviation** of the interpolant is when the data point indexed by j is missing.

6.2 Computational details

6.2.1 Linear model form

It is possible to ease the process of solving (or minimizing) the objectives by expressing (264) as a linear system of equations. First we simplify it by writing

$$\begin{aligned} D_k &:= D_k(w), \\ S_K &:= S_K(w), \\ \phi_{kl} &:= \phi_{kl}(w), \\ \psi_{kl} &:= \theta_{kl}\phi_{kl}, \\ \gamma_k &:= D_k S_K, \\ \alpha_j &:= \gamma_j - 1, \end{aligned}$$

by this, (264) can be rearranged into

$$\begin{aligned} \Delta_{jK}(w, \theta) &= \frac{\left(\sum_{k \in K} \frac{E_k + \sum_l \psi_{kl}}{D_k}\right) / S_K - \left(E_j + \sum_l \psi_{jl}\right)}{\alpha_j} \\ &= \sum_k \frac{E_k}{\gamma_k \alpha_j} + \sum_{kl} \frac{\psi_{kl}}{\gamma_k \alpha_j} - \sum_l \frac{\psi_{jl}}{\alpha_j} - \frac{E_j}{\alpha_j} \\ &= \sum_k \frac{E_k}{\gamma_k \alpha_j} + \sum_{kl} \frac{\psi_{kl}}{\gamma_k \alpha_j} - \sum_l \frac{\delta_{jk} \psi_{kl}}{\alpha_j} - \frac{E_j}{\alpha_j} \\ &= \sum_{kl} \frac{\psi_{kl}(1 - \gamma_k \delta_{jk})}{\gamma_k \alpha_j} - \left(E_j - \sum_k \frac{E_k}{\gamma_k}\right) / \alpha_j, \end{aligned} \tag{265}$$

where δ_{jk} is the Kronecker delta. Hence if we pick $w = w_m$, then

$$\begin{aligned} \Delta_{jK}(w_m, \theta) &= \sum_{kl} A_{jm,kl} \theta_{kl} - b_{jm}, \\ &= A_{jm,:} \theta - b_{jm} = (A\theta - b)_{jm} \end{aligned} \tag{266}$$

where

$$A_{jm,kl} = \frac{\phi_{kl}(w_m)(1 - \gamma_k(w_m)\delta_{jk})}{\gamma_k(w_m)\alpha_j(w_m)} \tag{267}$$

is the data matrix, and

$$b_{jm} = \left(E_j - \sum_k \frac{E_k}{\gamma_k(w_m)}\right) / \alpha_j(w_m) \tag{268}$$

is the target vector.

6.2.2 B-spline basis function

In this paper we use basis functions of the form

$$\phi_b(w) := \beta_\tau((Pw)_t) \text{ for } b = [\tau, t], \tag{269}$$

where β_τ are univariate **B-splines** that will be explained below, and P is a fixed matrix (generated once for each choice of number n_s of B-splines and then stored). Here we set $P = I$. Alternatively, assuming the molecular fingerprints w to be

normalized, P has its first m_f rows the rows of the identity matrix of size m_f , and in later rows row vectors with random Gaussian $N(0, 1)$ coefficients.

The **standard cubic B-spline** is defined as

$$\text{bspline}(z) := \begin{cases} 1 + 0.75x^2(|x| - 2) & \text{if } |x| < 1, \\ 0.25(2 - |x|)^3 & \text{if } 1 \leq |x| < 2, \\ 0 & \text{otherwise.} \end{cases} \quad (270)$$

It is twice continuously differentiable with support $[-2, 2]$ and derivative

$$\text{bspline}'(z) := \begin{cases} x(2.25|x| - 3) & \text{if } |x| < 1, \\ -0.75\text{sign}(x)(2 - |x|)^2 & \text{if } 1 \leq |x| < 2, \\ 0 & \text{otherwise.} \end{cases} \quad (271)$$

If x takes values in $[\underline{x}, \bar{x}]$ then

$$u(x) := (x - \underline{x})/(\bar{x} - \underline{x}) \in [0, 1], \quad (272)$$

and

$$\beta_\tau(x) := \text{bspline}(c_s u(x) + 2 - \tau) \quad (\tau = 1 : n_s) \quad (273)$$

defines for $n_s \geq 3$ and $c_s := n_s - 3$ a family of n_s B-splines.

For example, given a feature matrix $F = [w_1, w_2, \dots, w_{n_{\text{data}}}]^T$, where $w_k \in \mathbb{R}^{n_f \times 1}$ are feature vectors. We can compute the univariate splines β and its derivative β' given F_{ij} for $i = 1, \dots, n_{\text{data}}$, $j = 1, \dots, n_f$ (vectorized over matrix F)

$$\begin{aligned} \Phi &:= \beta(F) \in \mathbb{R}^{n_{\text{data}} \times n_L}, \\ \Phi' &:= \beta'(F) \in \mathbb{R}^{n_{\text{data}} \times n_L}. \end{aligned} \quad (274)$$

where $n_L = n_f n_s$ and n_s is the number of splines for each feature.

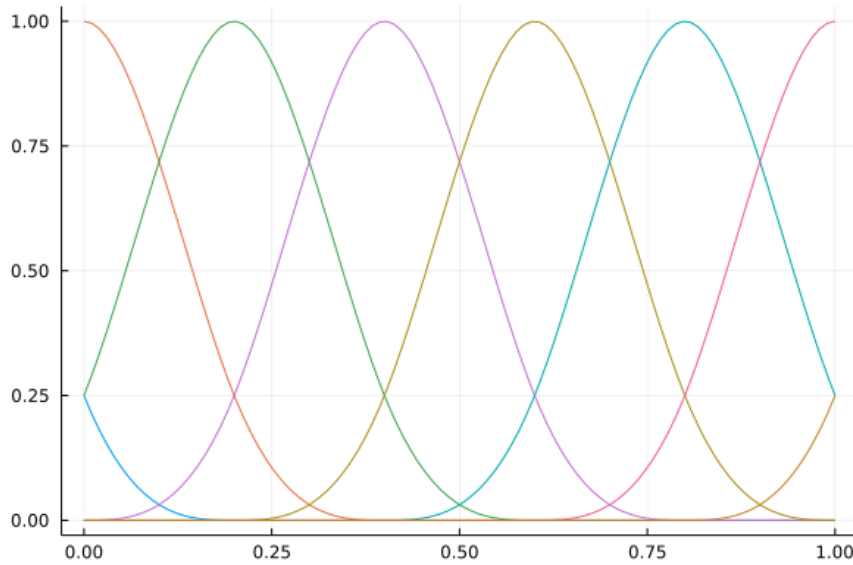


Figure 23: The `bspline` with $n_s = 8$.

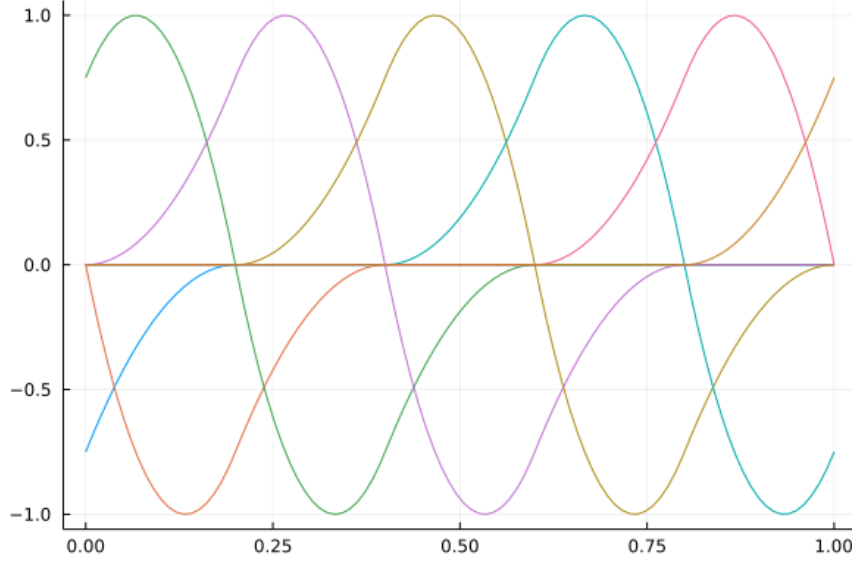


Figure 24: The derivative of `bspline` with $n_s = 8$.

6.2.3 Precomputed values

Several values can be computed independently of θ , this implies that these values are also independent of the objective function computation. Separating the computation of these values reduces the total running time needed to evaluate the objective function. We compute these values step-by-step, as follows:

- Compute the labels: Determine the "trial data points" T , by selecting some data points from the dataset. Then we can further determine the labels of the initial training data points $K \subset T$. Given K , we can obtain the labels $U := T \setminus K$ of the unsupervised data points and the labels $W := \{1, 2, \dots, n_{\text{data}}\} \setminus K$ of the evaluation data points.
- Compute the distance matrix $D \in \mathbb{R}^{n_{\text{data}} \times n_T}$, where $n_T = |T|$, in which the distance between two data points of index k and m is defined by

$$D_k(F_{m:}) := \|F_{m:} - F_{k:}\|_B^2 := \|B(F_{m:} - F_{k:})\|_2^2, \quad (275)$$

and here we pick $B = I$.

- Compute intermediate values:

$$\begin{aligned} S_K &\in \mathbb{R}^{n_U} := S_K(F_{m:}) \text{ for } m \in U, \\ \gamma &\in \mathbb{R}^{n_U \times n_K} := \gamma_k(F_{m:}) \text{ for } k \in K, m \in U, \\ \alpha &:= \gamma - 1. \end{aligned} \quad (276)$$

where $n_U = |U|$.

- Given the splines in (274), compute

$$\phi_{m,kl} := \Phi_{ml} - \Phi_{kl} - \Phi'_{kl}(F_{mt} - F_{kt}) \quad \text{for } m \in U, k \in K, l = 1, \dots, n_L, \quad (277)$$

where

$$t = \begin{cases} n_f, & \text{if } \hat{t} = 0 \\ \hat{t}, & \text{otherwise} \end{cases} \quad (278)$$

with

$$\hat{t} := l \bmod n_f, \quad (279)$$

this makes sure the correct column is indexed, since $\phi'_b(w)$ is only non zero at the t th column position. This results in the matrix ϕ where the block column is indexed by k, l indices.

6.2.4 Fitting the model

A good prediction model is reflected by small **mean absolute deviation (MAD)**:

$$\text{MAD}_K(w, \theta) := \frac{1}{|K|} \sum_{j \in K} |\Delta_{jK}(w, \theta)|, \quad (280)$$

or **root mean square deviation (RMSD)**:

$$\text{RMSD}_K(w, \theta) := \sqrt{\frac{1}{|K|} \sum_{j \in K} \Delta_{jK}(w, \theta)^2}, \quad (281)$$

hence a minimization routine is required. In order to solve the minimization problem, a linear system or least squares formulation is needed. For example, if the RMSD is chosen as the objective function then the problem formulation would be

$$\begin{aligned} f_{\text{obj}}(\theta) &:= |K| \sum_m \text{RMSD}_K(w_m, \theta)^2 = \sum_m \sum_j \Delta_{jK}(w_m, \theta)^2 \\ &= \sum_{jm} (A\theta - b)_{jm}^2 = \|A\theta - b\|_2^2, \end{aligned} \quad (282)$$

where A is the data matrix defined in (267) that has $n_K n_U$ rows and $n_K n_L$ columns, and θ is the coefficient vector.

For storage efficiency, it is desirable to not explicitly form the data matrix A , since it grows quickly with more data points included in the training set K and testing set T . We can increase the storage efficiency by writing a routine for computing the matrix vector product Au where u is a variable vector, hence instead of having a matrix with $\mathcal{O}(n_K^2 n_U n_L)$ storage we would have a vector with only $\mathcal{O}(n_U)$ storage. This can be done by augmenting the first sum of the last line of (265) with the precomputed intermediate values in (276):

$$Au = \sum_{kl} \frac{\phi_{m,kl} u_{kl} (1 - \gamma_{mk} \delta_{jk})}{\gamma_{mk} \alpha_{mj}}, \quad (283)$$

Here, CGLS [36, 37] method is used to minimize the objective function in (282). The memory-less form of CGLS requires not only Au but also $A^\top v$ routine, which is

$$A^\top v = \sum_{jm} \frac{\phi_{m,kl} v_{jm} (1 - \gamma_{mk} \delta_{jk})}{\gamma_{mk} \alpha_{mj}}. \quad (284)$$

Given the Au and $A^\top v$ routine, and b , the CGLS is called as follows [86, 87]:

$$\begin{aligned} \text{op} &\leftarrow \text{LinearOperators}(y, Au, A^\top v), \\ \theta &\leftarrow \text{CGLS}(\text{op}, b, \text{itmax}), \end{aligned}$$

where y is an arbitrary sized vector of real numbers (in Julia, y can be initialized by `y = Vector{Float64}()`) and `itmax` is the maximum number of iteration.

For the actual accuracy of the prediction, we can use either the **mean absolute error (MAE)**

$$\text{MAE} := \frac{1}{|W|} \sum_{m \in W} |E_m - V_K(w_m, \theta)|, \quad (285)$$

or the **root mean squared error (RMSE)**

$$\text{RMSE} := \sqrt{\frac{1}{|W|} \sum_{m \in W} (E_m - V_K(w_m, \theta))^2}. \quad (286)$$

6.3 Numerical experiments on pair potential H_xO_y dataset

The datasets from Sections 4.1 - 4.3 are used for the benchmark in this section. Each dataset is split into `5-fold` for training and testing. CHIPR use different `5-fold` sets than ROSEMI (Shepard model). The test RMSEs from the benchmark are in Table 7. Hyperparameters used for the experiments:

- Interpolation condition mode `force` $\in \{\text{true}, \text{false}\}$. If `force = true` then the interpolation condition which takes into account the equilibrium force in (258). Otherwise (257) is used.
- The power of the constants $n \in \{1, 2, 3\}$ of the fingerprint

$$F(r; r_{\text{eq}}, n) := \frac{r^n}{r^n + r_{\text{eq}}^n},$$

where r_{eq} is the equilibrium distance.

- The number of splines $c_s \in [1, 2, \dots, 10]$.
- The ratio $p_{\text{tr}} \in \{0.1, 0.2, \dots, 0.9\}$ of supervised training set K to be split from the selected centers T . The ratio for the unsupervised set U is $1 - p_{\text{tr}}$. See Subsection 6.2.3 for more details regarding the relationships of K , U , and T .

They are optimized using `Hyperopt.jl` [88] with `LHSampler()` algorithm, and maximum thread counts. The objective value `fobj` used is the mean of the RMSEs of all 5-folds. The current best hyperparameters found are in Table 27 in Appendix B.2.

Table 7: ROSEMI and CHIPR benchmark for pair potential H_xO_y molecules. Bold number indicates the minimum RMSE in each measurement category for each dataset.

Dataset	min		median		mean		max	
	ROSEMI	CHIPR	ROSEMI	CHIPR	ROSEMI	CHIPR	ROSEMI	CHIPR
H_2^1	7.880e-04	5.769e-05	1.325e-03	8.873e-05	2.603e-03	1.859e-04	8.280e-03	5.924e-04
H_2^2	8.398e-05	1.998e-05	4.812e-02	1.306e-04	5.063e-02	2.611e-04	1.521e-01	6.185e-04
H_2+^1	1.409e-03	1.079e-04	3.187e-02	1.729e-03	4.863e-02	2.814e-02	1.674e-01	1.313e-01
O_2^1	3.982e-04	6.599e-04	1.265e-03	1.396e-03	1.781e-03	1.633e+02	4.324e-03	8.166e+02
O_2^2	7.252e-04	5.012e-04	1.384e-03	1.696e-03	2.393e-03	1.810e-01	7.130e-03	8.999e-01
O_2^3	2.373e-04	2.818e-04	7.126e-04	7.042e-04	1.837e-03	6.837e-04	5.084e-03	1.046e-03
O_2+^1	1.603e-03	1.605e-04	3.161e-03	2.148e-04	3.593e-03	2.522e-04	6.422e-03	4.390e-04
OH^1	1.840e-04	4.928e-04	1.657e-03	1.656e-03	2.786e-03	1.319e-02	7.404e-03	5.107e-02
OH^2	7.729e-04	7.313e-04	4.006e-03	7.006e-03	3.944e-03	9.450e-03	6.918e-03	2.054e-02
OH^3	1.221e-04	3.566e-04	3.503e-03	8.724e-04	4.261e-03	1.820e-02	1.147e-02	8.789e-02
OH^4	1.340e-03	4.267e-04	2.652e-02	8.041e-04	2.068e-02	1.884e-03	4.523e-02	6.138e-03
OH^5	3.414e-04	1.430e-04	2.524e-03	4.876e-04	1.517e-02	9.391e-03	4.975e-02	4.456e-02
OH^6	5.703e-04	1.527e-04	1.381e-03	4.575e-04	1.237e-03	3.803e-04	1.719e-03	5.589e-04
$OH+^1$	2.054e-03	1.011e-04	4.914e-03	5.915e-04	8.715e-03	1.311e-03	2.798e-02	4.494e-03
$OH+^2$	8.436e-04	2.438e-04	4.048e-03	4.202e-04	5.422e-03	4.138e+00	1.099e-02	2.069e+01

6.4 Numerical experiments on QM9 dataset

Depending on the model and algorithm, fitting for QM9 dataset may require specific attention in particular for memory management and code execution efficiency, due to the size of the dataset. For instance, separating terms that can be computed independently of the optimization coefficients, this would make the training process more time efficient. Although in general this is applicable for any program that is related to implementation of models and optimization algorithms. For this section of numerical experiment, we also provide the structure of the program functions in Appendix B.1.

6.4.1 Memory usage

The computation of fitting and prediction of QM9 challenge is resource-demanding, hence we need to allocate variables responsibly such that the OS does not throw an out-of-memory exception. The first ingredient is to approximate the size of data structures allocated within the program in megabytes (MB) order of magnitude; assuming the data structure holds `Float64` entries (which is the most common data type for numerical operations and is also the most memory consuming) this can be done by a general formula

$$\text{mem_usage} = (8 \times 10^{-6}) \times \prod_i \text{size}(\text{data_structure}, i), \text{ for } i = 1, 2, \dots \quad (287)$$

where `size(data\_structure, i)` returns the number of entries of the i th dimension of a data structure. For example, the distance matrix $D \in \mathbb{R}^{133628 \times 300}$ with `Float64` data type, then

$$B_N X := B_N \times 300 = 320.7072 \text{ MB}, \quad (288)$$

where

$$B_N := (8 \times 10^{-6}) \times 133628 \approx 1.069 \quad (289)$$

is the storage needed for N entries of `Float64` in MB. Using Julia this can be confirmed simply by calling `Base.summarysize(D)*1e-6`, which gives the same result up to some roundoff error. For example, given $n_K = 100$, $n_f = 28$, and $n_b = 5$ for ROSEMI, the block matrix ϕ for prediction has the memory size of

$$B_N XY := B_N \times 100 \times 140 = 14966.336 \text{ MB} = 14.9 \text{ GB.} \quad (290)$$

With `batchsize = 10_000`, for each mini-batch loop the size of ϕ becomes 1/13 of its actual size which is around 1GB.

6.4.2 Hyperparameter space

We can construct the whole molecular fitting process from start to end by choosing many different possible combinations of hyperparameters. Table 8 contains the excluded hyperparameters, meanwhile Table 9 shows the list of variables included in the hyperparameter optimization.

Table 8: The list of fixed hyperparameters throughout all experiments.

hyperparameter	type	range	fixed value
<code>itmax</code>	<code>Int</code>	$[1, \infty]$	500
n_{pf}	<code>Int</code>	$[0, \infty]$	0
N	<code>Int</code>	$[1, N_{QM9}]$	300
k	<code>Int</code>	$[0, 95]$	38
n_{HL}	<code>Int</code>	$[1, 50]$	2
n_{nodes}	<code>Int</code>	$[1, 100]^{n_{HL}}$	$10^{n_{HL}}$
<code>f_act</code>	<code>Cat</code>	$[1, 3]^{n_{HL}}$	$1^{n_{HL}}$
<code>NN_opt</code>	<code>Cat</code>	$[1, 3]$	1
<code>pca_sf</code>	<code>Cat</code>	$[0, 1]$	0
<code>mfem_ss</code>	<code>Cat</code>	$[0, 1]$	0
<code>mfem_bf</code>	<code>Cat</code>	$[0, 1]$	0

Table 9: The hyperparameters passed into the hyperparameter optimization routine with its data type and possible range. Under the **type** column, **Int** means integer, **Cat** means categorical, and **OInt** means ordered integers.

hyperparameter	type	range
$n_{mf} \leq n_{af}$	<code>OInt</code>	$[16, 20]$
n_s	<code>Int</code>	$[1, 10]$
c	<code>Real</code>	$[0, 20]$
<code>f_type</code>	<code>Cat</code>	$[1, 3]$
<code>norm_af</code>	<code>Cat</code>	$[0, 1]$
<code>norm_mf</code>	<code>Cat</code>	$[0, 1]$
<code>model</code>	<code>Cat</code>	$[1, 5]$

During feature selection, there exist several possible choices of hyperparameters:

- $n_{mf} \leq n_{af} \in \mathbb{Z}_{>0}$ (**ordered integers**), the number of molecular features chosen n_{mf} and the number of atomic features n_{af} .
- $n_s \in \mathbb{Z}_{>0}$ (**integral**), number of splines.

- `fetype` $\in \{\text{ACSF}, \text{SOAP}, \text{FCHL}\}$ (**categorical**), the choice of set of atomic features.
- `norm_af` $\in \{\text{false}, \text{true}\}$ (**categorical**), the atomic features normalization mode, the choice of whether to normalize the atomic features.
- `norm_mf` $\in \{\text{false}, \text{true}\}$ (**categorical**), the molecular features normalization mode, the choice of whether to normalize the molecular features.
- `pca_sf` $\in \{\text{diag}, \text{sens}\}$ (**categorical**), the choice for the scaling factor of the correlation matrix: the diagonals of the covariance matrix or the sensitivity vector. $n_{pf} \in \mathbb{Z}_{>0}$ (**integral**), which is the number of features involved for the sensitivity vector computation.
- `mfem_ss` $\in \{\text{false}, \text{true}\}$ (**categorical**), the feature extraction mode, whether to include sum of squares features.
- `mfem_bf` $\in \{\text{false}, \text{true}\}$ (**categorical**), the feature extraction mode, whether to include binomial features.

For data selection, the possible hyperparameters are:

- $k \in \mathbb{Z}_{\geq 0}$ (**integral**), the index of precomputed set of selected centers by the `usequence`.
- $N \in [1 : N_{\text{QM9}}] \subseteq \mathbb{Z}_{>0}$ (**integral**), the number of centers requested;

For linear model fitting, the possible choices of hyperparameters are:

- `model` $\in \{\text{RoSeMI}, \text{KRR}, \text{NN}, \text{LLS}, \text{GAK}\}$ (**categorical**), the choice of models.
- $c \in \mathbb{Z}_{\geq 0}$ (**integral**), the scaling factor 2^c for KRR and GAK kernels.
- `itmax` $\in \mathbb{Z}_{>0}$ (**integral**), number of CGLS iterations for the linear models.

For neural network, the possibilities are:

- $n_{\text{HL}} \in \mathbb{Z}_{>0}$ (**integral**), the number of hidden layers for NN,
- $n_{\text{nodes}} \in \mathbb{Z}_{>0}^{1 \times n_{\text{HL}}}$ (**integral**), the number of nodes in each hidden layer,
- `f_act` $\in \{\text{ReLU}, \text{sigmoid}, \text{tanh}\}^{1 \times n_{\text{HL}}}$ (**categorical**), the choice of activation function for each node in each layer.
- `NN_opt` $\in \{\text{Adam}, \text{SGD}, \text{RMSProp}\}$ (**categorical**), the choice of NN optimizer.

All categorical variables are encoded in **one-hot-encoding** form, since there is no natural ordering of categorical variables (in contrast to variables with real or integer types). The encoding process transforms a value into a vector in which the length of the vector is equal to the size of the set variable, where the vector contains all 0 except at the relative position of the selected value within the set where we set it as 1. For example, if we choose "RoSeMI" as `model`, then

$$\text{f_encode}(\text{"RoSeMI"}, \text{model}) = [1, 0, 0, 0, 0],$$

another example would be if we choose "SOAP" as the feature type then

$$\text{f_encode}(\text{"SOAP"}, \text{fetype}) = [0, 1, 0],$$

here `f_encode(val, var)` is a function which encodes the value `val` given a set variable `var` into one-hot vector format.

6.4.3 Fitting results

The true error of the fitting is described by the mean absolute error (MAE) of the energies

$$\text{MAE} := \frac{1}{N_{\text{QM9}}} \sum_{m \in W} |E_m - \hat{E}_m|, \quad (291)$$

where $\hat{E}$ is the vector of predicted energies. The initial set of the fixed hyperparameters across all experiments is:

$$N_{\text{QM9}} = 133628, \quad n_T = 300, \quad n_K = 100, \quad \text{t_limit} = 900.$$

Each `data_setup()` process roughly takes around 300s. The fitting is done using `CGLS()` with `itmax` = 500; Specifically for ROSEMI, it requires the functions to compute Au and $A^\top v$ as the input variable. Currently active learning scheme has not been tested.

6.4.3.1 Null model and reduced energies The null model serves to reduce the molecular energy fitting difficulty by decreasing the order of magnitude of the fitted energy, since by default the molecular energy itself has a large order of magnitude (e.g., the internal energy at 0° Kelvin U_0 of QM9 molecules are between $\mathcal{O}(100)$ to $\mathcal{O}(1000)$ in Hartree unit). We can obtain the reduced energy of a molecule by removing the null energy

$$E^{\text{red}} = E - E^{\text{null}}, \quad (292)$$

where the null energy is

$$E^{\text{null}} = \sum_{j \in P} N_j E_j^{\text{atom}}, \quad (293)$$

where P is a set of atom types, N_j is the count of the atom type j within a fixed molecule, and E_j^{atom} is the energy of atom type j . In fact, null energy is equivalent to the energy computed by dressed atom, while the reduced energy is equivalent to the effective energy after the total energy is subtracted by the null energy.

The atomic energy itself can be obtained by fitting a LLS model. For QM9 challenge, (69) can be used to obtain the atomic energies, explicitly by defining

$$\begin{aligned} A_i &:= [N_{\text{H}}^i, N_{\text{C}}^i, N_{\text{O}}^i, N_{\text{N}}^i, N_{\text{F}}^i], \\ \theta &:= [E_{\text{H}}^{\text{atom}}, E_{\text{C}}^{\text{atom}}, E_{\text{O}}^{\text{atom}}, E_{\text{N}}^{\text{atom}}, E_{\text{F}}^{\text{atom}}], \\ b_i &:= E_i, \\ &\text{for } i \in K. \end{aligned} \quad (294)$$

6.4.3.2 Choice of training sets The set K of training labels depends on the set of features computed during the feature selection process. The complete sets of features used to generate the 95 sets of labels are displayed in Table 28 in the appendix. The scatter plot of the training MAE against the full MAE of the whole 95 sets of labels is shown in figure 25. Each set is generated by `usequence`, here each set of features is used to generate 5 distinct sets of training labels. 10 results of the atomic energies fitting against the molecular energies E which gives the lowest training MAE is shown in Table 10. The labels from the 10 sets are displayed in Table 29 in the appendix. The full MAE from fitting the atomic energies is in fact refers to the mean absolute value of the E^{red} of all molecules in the QM9 dataset itself.

Table 10: 10 best (lowest training MAE) numerical experiments of fitting the null model to predict the null energies picked from 95 experiments. The **MAEs** and the atomic energies are in kcal/mol. The last row shows the experiment of null model fitted to N_{QM9} data points.

k	E_H^{atom}	E_C^{atom}	E_O^{atom}	E_N^{atom}	E_F^{atom}	Train MAE	Full MAE
38	-378.435	-23890.923	-34354.419	-47206.604	-62689.712	15.960	20.243
95	-379.462	-23890.242	-34352.997	-47203.710	-62663.338	16.004	20.079
31	-379.173	-23891.959	-34354.331	-47199.134	-62666.546	16.134	20.280
48	-378.438	-23891.821	-34352.819	-47204.099	-62661.154	16.201	20.035
83	-380.433	-23888.399	-34349.627	-47206.065	-62689.308	16.458	21.250
8	-377.764	-23894.064	-34354.560	-47202.149	-62684.920	16.912	20.726
1	-378.731	-23892.146	-34353.273	-47204.459	-62671.670	17.109	20.435
45	-380.123	-23889.676	-34354.064	-47203.101	-62656.640	17.287	20.146
26	-378.354	-23892.807	-34351.090	-47202.015	-62663.825	17.315	20.341
91	-376.980	-23894.187	-34352.579	-47204.336	-62658.886	17.421	20.226
all	-378.666	-23891.253	-34354.806	-47203.592	-62665.984	19.886	19.886

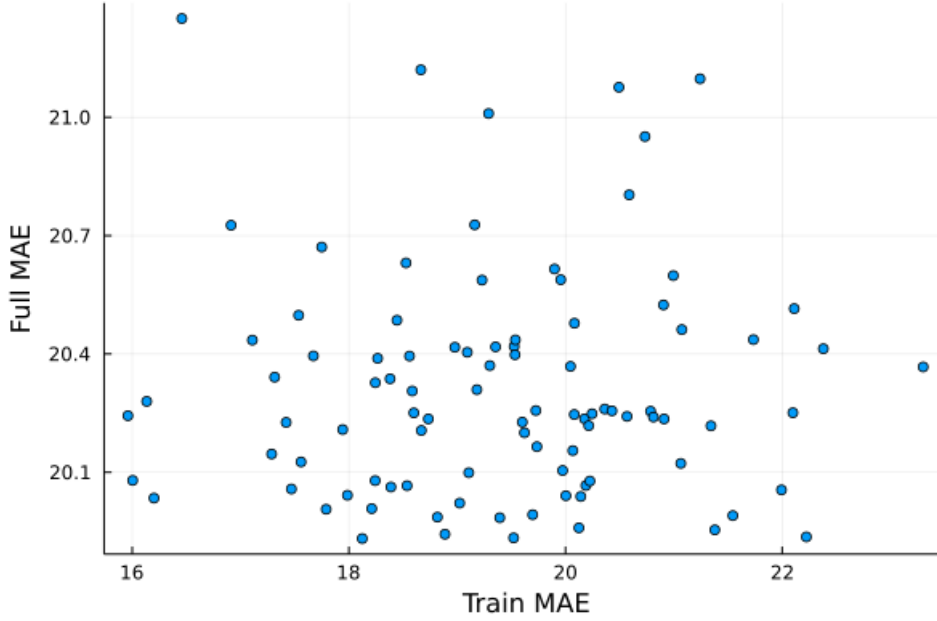


Figure 25: Scatter plot of training MAE against full MAE for 95 training sets.

6.4.3.3 Fitting reduced energies The fitting experiments with E^{red} as the target are run by varying the models, the set of features and the training labels. In total there are 50 instances, 10 of which with the lowest MAE are shown in Table 11. One set of features and one set of training labels from the lowest fitting MAE is chosen to compare the models, the results are shown in Table 12. Here the MAEs are computed from the original energies, since the predicted energies are actually the predicted reduced energies,

$$\text{MAE} := \frac{1}{N_{QM9}} \sum_{m \in W} |E_m - (\hat{E}_m + E_m^{\text{null}})| \quad (295)$$

The models' hyperparameters are fixed across all experiments. ROSEMI uses $n_s = 6$, KRR uses scaler $c = 1$, GAK uses $\sigma = 2048.$; and the neural network (NN)

architecture used here is composed of `nodes` = $[n_{mf}, 10, 1]$, where each entry represents the number of nodes in i th layer; ReLU is chosen as the activation function in the hidden layer (2nd layer); the NN is optimized using Adam with 0.01 momentum.

FCHL is also independently tested, it results in 9.3 kcal/mol MAE. The feature extraction, fitting, and prediction is done using the quantum machine learning (QML) package [34] in Python. The training set indices K is computed from the processed ACSF set of features with $n_{af} = 4$ and $n_{mf} = 18$. The hyperparameters used are: cutoff radius $r_{\text{cut}} = 8.0$ for both the FCHL feature extraction and the local kernel generation; and Gaussian scaler $\sigma = 32.0$ for the local kernel generation.

Table 11: 10 numerical experiments with the lowest **MAE** picked from 50 experiments, which is 5 model experiments for each selected index k . The MAEs are in kcal/mol. Null MAE refers to the MAE of the null model (in kcal/mol). k refers to the index of set of labels used for training. KRR refers to (72), GAK refers to (73), and LLS refers to (69). f is the type of atomic features, n_{af} is the number of atomic features, n_{mf} is the number of molecular features, t_s is the solver time to train the model in seconds, and t_p is the prediction time of N_{qm9} molecules in seconds.

MAE	Null train MAE	model	k	f	n_{af}	n_{mf}	t_s	t_p
11.626	15.960	GAK	38	ACSF	20	16	0.043	731.476
11.626	17.287	GAK	45	ACSF	20	16	0.044	676.670
12.188	17.315	GAK	26	ACSF	30	29	0.044	352.096
12.971	16.912	GAK	8	ACSF	40	37	0.043	401.476
13.227	16.201	GAK	48	ACSF	20	15	0.044	700.244
13.575	17.315	LLS	26	ACSF	30	29	0.002	0.011
13.582	16.134	GAK	31	ACSF	30	20	0.043	980.447
15.059	17.315	KRR	26	ACSF	30	29	0.045	8.874
15.328	16.912	KRR	8	ACSF	40	37	0.044	8.846
16.245	16.134	LLS	31	ACSF	30	20	0.001	0.006

Table 12: Model comparison under the same sets of training label and feature sets (the feature set in the first row of Table 11). ROSEMI refers to (266), and NN refers to (76).

MAE	model	t_s	t_p
19.776	ROSEMI	906.468	268.563
21.521	KRR	0.045	9.921
20.244	NN	16.238	0.008
18.507	LLS	0.001	0.004
11.626	GAK	0.043	731.476

6.4.3.4 Hyperparameter optimization (Hyperopt) To find the best model configuration, here we conducted numerical experiments using hyperparameters optimized by a derivative-free optimizer from [89, 90].

Hyperopt with box constraints Infeasible solutions exist as the outputs of unconstrained solvers, one such example is when $n_{mf} \leq 0$. Here infeasible solutions are treated in a straightforward manner by these options:

- Set a very high MAE for infeasible solutions.
- Round infeasible solutions to the boundary. This only applies for box-constraints.

The set of hyperparameters chosen to be optimized is the subset of hyperparameters explained in section 6.4.2, which are:

$$H := \{p_{af}, p_{mf}, n_s, \text{ftype}, \text{norm_af}, \text{norm_mf}, k, \text{model}, c\},$$

where $p_{af}, p_{mf} \in (0, 1] \cap \mathbb{R}$ are the percentage of the features selected,

$$n_{af} = p_{af} \bar{n}_{af},$$

where $\bar{n}_{af}$ is the maximum possible number of atomic feature selected, e.g., for ACSF $\bar{n}_{af} = 51$; this is also analogous for the percentage of molecular features p_{mf} . H is initialized by the set of hyperparameters which produced the lowest MAE in table 11 (1st row) with perturbation

$$H = \{0.392, 0.800, 3, \text{ACSF}, \text{true}, \text{true}, 38, \text{GAK}, 32\}$$

A series of experiments by setting infeasible solutions to have MAE = 100 is done, 30 latest hyperparameters found are shown in table 13. The second series of experiments is done by projecting the infeasible solutions into the boundaries of feasible solutions, the bounds are

$$H_{\text{bounds}} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & \text{Inf} & 3 & 1 & 1 & \text{Inf} & 5 & \text{Inf} \end{pmatrix}, \quad (296)$$

where each column corresponds to the each hyperparameter boundaries. 25 hyperparameters found are in Table 14.

Table 13: Sets of hyperparameters and their MAE found during the hyperparameter optimization process. **MAE** is in kcal/mol. Here infeasible points are set to have MAE = 100.

MAE	p_{af}	p_{mf}	n_s	<i>ftype</i>	<i>norm_af</i>	<i>norm_mf</i>	k	<i>model</i>	c
11.785	0.389	0.798	2	ACSF	true	true	38	GAK	33
11.785	0.393	0.789	3	ACSF	true	true	38	GAK	33
11.785	0.390	0.798	2	ACSF	true	true	38	GAK	33
11.785	0.390	0.798	3	ACSF	true	true	38	GAK	33
11.872	0.396	0.797	3	ACSF	true	true	38	GAK	32
11.881	0.393	0.802	3	ACSF	true	true	38	GAK	31
12.070	0.374	0.925	2	ACSF	true	true	38	GAK	33
12.572	0.390	0.798	2	ACSF	true	true	37	GAK	33
12.572	0.390	0.799	2	ACSF	true	true	37	GAK	33
12.572	0.389	0.798	3	ACSF	true	true	37	GAK	33
12.572	0.390	0.798	2	ACSF	true	true	37	GAK	33
12.583	0.391	0.798	3	ACSF	true	true	37	GAK	32
12.738	0.466	0.708	2	ACSF	true	true	39	GAK	31
17.959	0.503	0.989	3	ACSF	false	true	38	GAK	32
18.024	0.396	0.800	3	ACSF	false	true	38	GAK	32
18.408	0.392	0.800	4	ACSF	true	true	37	LLS	32
15.975	0.390	0.798	2	SOAP	true	true	38	GAK	32
15.975	0.389	0.799	3	SOAP	true	true	38	GAK	32
15.975	0.390	0.799	3	SOAP	true	true	38	GAK	32
15.975	0.387	0.798	4	SOAP	true	true	38	GAK	32
15.975	0.391	0.799	3	SOAP	true	true	38	GAK	32
16.002	0.390	0.799	2	SOAP	true	true	38	GAK	33
16.002	0.390	0.799	2	SOAP	true	true	38	GAK	33
16.492	0.390	0.799	2	SOAP	true	true	37	GAK	33
16.492	0.387	0.795	3	SOAP	true	true	37	GAK	33
16.492	0.388	0.798	2	SOAP	true	true	37	GAK	33
20.237	0.312	0.998	3	SOAP	false	true	38	GAK	33
20.240	0.403	0.782	4	SOAP	false	true	38	GAK	31
21.886	0.164	0.977	4	SOAP	true	false	38	LLS	32
25.816	0.250	0.867	3	SOAP	false	true	38	LLS	31

Table 14: Sets of hyperparameters and their MAE found during the hyperparameter optimization process. **MAE** is in kcal/mol. Here infeasible points are projected into the boundaries of feasible solutions.

MAE	p_{af}	p_{mf}	n_s	f type	norm_af	norm_mf	k	model	c
12.063	0.465	0.926	3	ACSF	true	true	0	GAK	32.0
12.580	0.392	0.800	3	ACSF	true	true	0	GAK	32.0
12.736	0.377	0.811	3	ACSF	true	true	0	GAK	32.0
12.869	0.392	0.800	3	ACSF	true	true	0	GAK	32.0
12.962	0.445	0.815	3	ACSF	true	true	0	GAK	32.0
12.977	0.366	0.773	3	ACSF	true	false	0	GAK	32.0
13.091	0.409	0.805	3	ACSF	true	true	0	GAK	32.0
13.192	0.376	0.795	3	ACSF	true	true	0	GAK	32.0
13.295	0.392	0.800	3	ACSF	true	true	0	GAK	32.0
13.313	0.093	1.000	3	ACSF	true	true	0	GAK	32.0
13.512	0.392	0.800	3	ACSF	true	true	0	GAK	32.0
13.628	0.418	0.827	3	ACSF	true	true	0	GAK	31.0
13.660	0.392	0.800	3	ACSF	true	true	0	GAK	32.0
13.728	0.393	0.799	3	ACSF	true	true	0	GAK	32.0
13.729	0.691	0.415	4	ACSF	true	true	0	GAK	33.0
13.768	0.392	0.801	3	ACSF	true	true	0	GAK	32.0
13.866	0.394	0.795	3	ACSF	true	true	0	GAK	32.0
13.869	0.392	0.800	3	ACSF	true	true	0	GAK	32.0
14.144	0.392	0.800	3	ACSF	true	true	0	GAK	32.0
14.383	0.394	0.793	3	ACSF	true	true	0	GAK	32.0
14.474	0.392	0.800	3	ACSF	true	true	0	GAK	32.0
14.813	0.391	0.805	3	ACSF	true	true	0	GAK	32.0
15.130	0.407	0.789	3	ACSF	true	true	0	GAK	32.0
19.037	0.340	0.785	3	ACSF	true	true	0	LLS	32.0
20.871	0.320	0.674	4	SOAP	false	true	0	GAK	33.0

Hyperopt with special constraints As explained in section 6.4.2, several variables have special constraints that are implied by the types, for example, the relationship $n_{mf} \leq n_{af}$. In particular, here we treat categorical (**Cat**) and ordered integer variables (**OInt**) using the mechanisms in section 2.3.1.1 and 2.3.1.2 respectively. The middle-point between the lower and upper bound of each variable is used as the initial starting point as shown in Table 15, this is done to promote more exploration. Table 16 shows the samples of hyperparameters found and their MAE.

Table 15: The augmented version of Table 9. The hyperparameters passed into the hyperparameter optimization routine with its data type, possible range, best found value, and starting value. Under the **type** column, **Int** means integer, **Cat** means categorical, and **OInt** means ordered integers. The hyperparameters in the **best** column give the current best found MAE = 11.626.

hyperparameter	type	range	best	start
$n_{mf} \leq n_{af}$	OInt	[1,50]	[16, 20]	[1/3, 1/3, 1/3]
n_s	Int	[1,10]	3	5.5
c	Real	[0, 20]	11	10.5
f _{type}	Cat	[1,3]	1	[1/3, 1/3, 1/3]
norm _{af}	Cat	[0,1]	1	[1/2, 1/2]
norm _{mf}	Cat	[0,1]	1	[1/2, 1/2]
model	Cat	[1,5]	3	[1/5, 1/5, 1/5, 1/5, 1/5]

Table 16: Sets of hyperparameters and their MAE found during the hyperparameter optimization process using the special mechanisms for **OInt** and **Cat**.

MAE	n_{mf}	n_{af}	n_s	f _{type}	norm _{af}	norm _{mf}	model	c
12.053	22	38	6	ACSF	1	0	GAK	10.356
12.053	22	38	1	ACSF	1	1	GAK	10.356
12.053	22	38	5	ACSF	1	0	GAK	10.356
15.552	21	34	6	ACSF	0	1	GAK	10.435
15.634	22	38	3	ACSF	1	0	LLS	10.356
15.655	22	38	6	ACSF	1	1	LLS	10.356
15.805	22	38	6	ACSF	0	0	GAK	10.356
15.805	22	38	10	ACSF	0	1	GAK	10.356
19.889	22	38	5	ACSF	0	0	ROSEMI	10.356
19.918	22	38	10	ACSF	0	0	ROSEMI	10.356
20.046	22	38	7	ACSF	1	0	ROSEMI	10.356
20.050	21	35	6	ACSF	1	0	ROSEMI	10.380
20.050	22	38	10	ACSF	1	0	ROSEMI	10.356
20.053	22	38	3	ACSF	1	1	ROSEMI	10.356
20.131	22	38	8	ACSF	1	1	ROSEMI	10.356
20.204	22	38	9	ACSF	1	1	ROSEMI	10.356
20.253	19	32	5	ACSF	1	1	NN	10.162
21.240	14	31	6	SOAP	0	1	ROSEMI	10.584
20.295	27	40	6	FCHL	1	1	NN	10.581
20.367	25	34	5	FCHL	0	0	GAK	10.269
20.367	22	38	5	FCHL	0	1	GAK	10.356
20.367	23	37	6	FCHL	0	0	GAK	10.322
20.367	22	38	1	FCHL	0	1	GAK	10.356
30.173	22	38	4	FCHL	1	0	GAK	10.356
99.999	22	38	1	FCHL	1	0	LLS	10.356
NaN	22	38	5	FCHL	1	0	ROSEMI	10.356
NaN	22	38	4	FCHL	1	1	ROSEMI	10.356
NaN	22	38	5	FCHL	0	0	ROSEMI	10.356
NaN	19	33	6	FCHL	1	1	ROSEMI	10.271
NaN	21	37	5	FCHL	1	1	ROSEMI	10.378

7 Delta machine learning (Δ ML) and optimal selection of molecules

In this chapter, we address the paths to find the optimal combination of techniques with the spirit of reaching chemical accuracy using only 100 molecules for training. Pipeline-wise, the initial procedure is composed of a way to select the most relevant 100 molecules from all of the 134k molecules; the discussion is mainly based on a sampling algorithm: farthest point sampling (FPS) [91]; in particular the stochastic adaptation: `usequence` [89]. The fitting itself is split into three main procedures: pre-fitting with **delta machine learning** (Δ ML), the main procedure of fitting with feature-model combination, and post-fitting optimization; lastly, hyperparameter optimization as an additional component that overarches most of the computational parts.

The Δ ML procedure has the purpose of reducing the fitting load. Although here the Δ ML for pre-fitting itself is technically a fitting mechanism, albeit simpler. An expansion of the many-body in terms of graph representation of molecules, dubbed as dressed [atom, bond, angle, torsion] (DA, DB, DN, DT) (see Section 2.1.1), analogous to "bonds, angles, machine learning" (BAML) [92] and "universal force fields" (UFF) [93], are used as the main sets of features for the Δ ML pre-fitting procedure. Typically, Δ ML mechanism is used to learn the correction (i.e., difference) between two levels of accuracy of the same quantum property [94, 95, 96], this implies at least two sets of quantum property targets computed from *ab initio* methods are used. However in this work, we make use only one level of accuracy of a quantum property directly from the QM9 dataset, since loosely speaking, using multiple sets of targets from many sources contradicts the spirit of this work, and more importantly it violates one of the QM9 challenge constraint [19]. Therefore here we use the Δ ML to learn the levels of quantum property recursively derived from only a level of quantum property, without additional *ab initio* calculation or resources.

7.1 Delta machine learning (Δ ML)

The crux of Δ ML is to take out as much low-level energy as possible before learning the rest of the high-level energy, i.e., to reduce the workload of the machine to learn the complicated relation between the molecular geometries and the high-level energies. As described by [95], the general form of the Δ ML can be formalised as

$$E^l = E^{l-1} + \text{ML}^l(\theta^l, F^l), \quad E^0 = \text{ML}^0(\theta^0, F^0) \quad (297)$$

where ML^l is a function to learn and predict $E_t^l := E_t - E^{l-1}$, which is the difference between two energy levels; where E_t is the target energy, θ^l is the set of parameters and F^l is the set of features. This recipe implies a multi-level recursive learning scheme for different levels of energies. Given a set of energy levels $l = 1, \dots, n$, we call all levels $l < n$ as **baselines**. In general it is expected that the lowest level of energy will have the largest magnitude, with decaying magnitude of energy as we traverse up the stairs of energy levels (for more details on the baseline feature extractions refer to Section 2.1.1).

7.2 Data selection technique

7.2.1 Unsupervised selection technique

One of the biggest hurdle in the QM9 challenge is that the training set size is limited to 100, which means less than 0.1% of the total 134k is used. However it is allowed to hand-pick the data points of the training set such that it gives better model generalization for the rest of the test-set (often also called as out-of-sample set). Here we employ heuristics to select the relevant data points for the training set, which is an adaptation of the farthest point strategy (FPS) by [91].

In essence, FPS is a sampling technique that preserves the uniformity of the sparse samples of the global set. An algorithm dubbed `usequence` [89], is an adaptation of the FPS with additional stochastic feature.

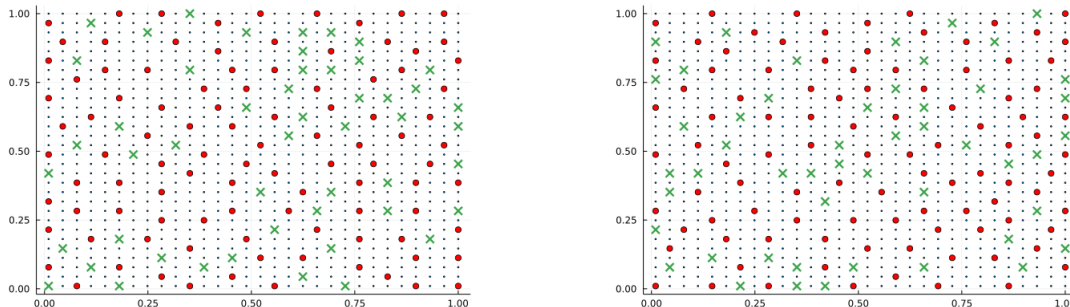


Figure 26: A grid of 900 points in 2D space, red dots are the first 100 points selected by usequence, green crosses are the next 50 points selected. Two sets (left and right image) of selection are generated from the same grid.

The `usequence` works on the molecular feature level. We can transform atomic feature vectors of a molecule into molecular feature vector in many ways. One way is to compute the sums of the atomic features sorted by the atomic types: given the i th molecule which contains number of atoms n_{atom} and a set of atom types $A = \{\text{H}, \text{C}, \text{O}, \dots\}$ with cardinality n_{type} , a partition of the molecular feature is computed by

$$s_j^i = \sum_{l \in M_j} x_l^i \quad (298)$$

where M_j is a set of indices which contains the row positions of atom type $j \in A$ in the x^l matrix. These partitions are then concatenated into the molecular feature vector of the i th molecule:

$$X_i := [s_1^i, \dots, s_{n_{\text{type}}}^i]. \quad (299)$$

7.2.2 Supervised selection technique

Aside from unsupervised way of training-set selection using the `usequence`, here we also conduct the training-set selection experiments in a supervised manner, in which the energy is included as a component of the objective function to be minimized. The goal is to minimize an objective function where the training indices act as the decision variables. Generally, this can be categorized as a combinatorial (or discrete) optimization. Here we formulate the problem as

$$\begin{aligned} \min \quad & f(x) \\ \text{subject to} \quad & c(x) = 0 \end{aligned} \quad (300)$$

where

$$f(x) = \frac{1}{N_{\text{test}}} \sum_k \left| \left[D(\neg x) K D(x) \left(D(x) K D(x) \right)^{-1} \left(D(x) E \right) \right]_k - \left[D(\neg x) E \right]_k \right| \quad (301)$$

is the nonlinear objective function to be minimized, we can also notice that in fact $f(x)$ is equivalent to the MAE.

$$c(x) = \sum_i x_i - 100, \quad x_i \in \{0, 1\}, \quad i \in [1, 2, \dots, N_{\text{QM9}}] \quad (302)$$

is the linear equality constraint to be fulfilled and x is a boolean vector of the training indices sorted by the label of the data where $x_i = 1$ if the data with label i is selected and $x_i = 0$ otherwise; and N_{QM9} is the size of the QM9 dataset.

$$D(x) \in \{0, 1\}^{N_{\text{QM9}} \times N_{\text{QM9}}}, \quad D(x)_{ii} := x_i \quad (303)$$

is a matrix that selects the rows and columns of K , where

$$K := K(H) \quad (304)$$

is the kernel matrix from Section 2.2.2 formed by the fixed hyperparameters H , where H should ideally be the best found hyperparameters.

We want to emphasize that (300 - 304) are only the formal mathematical representation, in the program implementation it is not desirable to compute $f(x)$ in that particular way due to inefficient (time and memory wise) K and $D(\cdot)$ computation. Slicing the rows and columns of K by x at the same time as it is being computed is the preferred way, this implies black-box optimization problem.

Selecting 100 training data from 130k data points that minimizes (300) is untractable, due to the very large number of possible combinations and due to the nature of discrete optimization where "descend directions" are not available (unlike continuous optimization). Therefore we rely on a heuristic, we devise an algorithm based on the Tabu search (Algorithm 4). Essentially Tabu search eliminates penalized candidates, where the penalty scheme itself can be in many different forms depending on problem context [97].

Algorithm 4 Tabu search for training-set optimization

Input:

- a vector $p^{(0)}$ of initial penalty values of all data points.
- a set of training sets $P^{(0)}$ selected during the process of generating $p^{(0)}$.
- hyperparameters:
 - size n of best training sets to be selected from the global set $P^{(0)}$.
 - numbers m of training sets x to be replaced on each iteration.
- reset criteria, e.g., number of iterations, rate of decay of objective value, etc.
- stopping criteria, e.g., number of iterations, objective value threshold, etc

Output: best obtained training set x^* and its corresponding objective value f^*

- 1: Pick $P \subset P^{(0)}$ where $|P| = n$ that have the lowest objective values.
 - 2: **while** stopping criteria are not met **do**
 - 3: **while** reset criteria are not met **do**
 - 4: Pick $x \in P$ randomly.
 - 5: Replace m numbers of data points $s \in x$ that have high penalty values with $s \notin x$ that have low penalty values.
 - 6: Evaluate $f(x)$ and update the penalty value $p(s) = \frac{g(s)+f^*}{h(s)+c}$, $\forall s \in x$ where $g(s)$ is the sum of all objective values of the simulations where s is involved, f^* is the currently best found objective value, $h(s)$ is the total number of simulations where s is involved, and c is a small positive constant, here we pick $c = 1$.
 - 7: **end while**
 - 8: Reinitialize P , and/or change the values of m and/or n .
 - 9: **end while**
-

7.3 Numerical experiments

Experiment scenarios consist mostly of the enumeration of combination of features, models, energy levels, and training set selections in order to bring down the errors to be as low as possible. The complete results are enumerated in the appendices: in Table 32 for randomly selected training molecules and Table 34 for pre-selected training molecules by the `usequence` algorithm.

7.3.1 Setup

7.3.1.1 Dataset The QM9 dataset [22] is used for the benchmarks, in particular the pre-cleaned version which was released by [98]. QM9 dataset contains 134k organic molecules in ground states composed of `atom_types = [H,C,N,O,F]`, with several properties computed from quantum chemistry, one of which is the internal energy at 0 K U_0 property is the target of our interest.

The dataset is further cleaned, 3054 uncharacterized molecules and 11 molecules with geometries that are difficult to converge are removed [98], this procedure results in **130823** molecules.

7.3.1.2 Hyperparameter space Throughout all of the scenarios of the numerical experiments, the hyperparameter space tested will be mostly the same, although

it is possible that there will be some additional or removal of the hyperparameters' components depending on the scenario which will be described exclusively within the section that correspond to the said scenario.

The hyperparameters that are commonly used throughout the numerical experiments:

- `ntrain` = 100. The number of molecules included in the training set; in particular, the main interest is in `ntrain` = 100 (as described in the QM9 challenge). Unless specified within the relevant section, `ntrain` = 100 is fixed. All of the MAEs shown in the plots are the MAEs of the out-of-sample sets (test sets) rather than the training sets', unless specified otherwise.
- `feature` $\in$ {ACSF, SOAP, FCHL19, MBDF, CMBDF}. The type of atomic features used. ACSF refers to the atomic centered symmetry function [3]. SOAP refers to the smooth overlap of atomic positions [4]. FCHL19 refers to Faber-Christensen-Huang-Lilienfeld [17]. MBDF refers to many-body-density-functional [6], and CMBDF is its convolutional variant [30].
 - ACSF is extracted using `DescriptorZoo` Julia package using the default hyperparameters available in the package [31, 32], this results in atomic feature vectors where each vector has 51 entries.
 - SOAP is extracted using `DScribe` Python package [33], the hyperparameters used are (`rcut`, `nmax`, `lmax`, `sigma`) = (6, 3, 3, 0.1) [6], this leads to each atomic vector having 480 entries.
 - FCHL19 is extracted using the development branch of `QML` Python package [34], the hyperparameters used are (`rcut`, `nRs2`, `nRs3`) = (6., 12, 10) [6], this gives each atomic feature vector 360 entries.
 - MBDF and CMBDF are extracted using the default parameters embedded in the program package [30], results in atomic feature vector with 40 entries.
- `model` $\in$ {GK, DPK}. GK refers to Gaussian kernel function in (70), here $\sigma = 32$ is chosen as default, unless stated otherwise. DPK refers to the linear kernel in (71).
- `solver` $\in$ {CGLS, direct}. Direct refers to an ensemble of row reduction techniques embedded in the backslash operator "`\`" of Julia programming language [99]; while CGLS refers to conjugate gradient for linear least squares problem [86, 36, 37]. For both solver, regularization variable $\lambda = 1e - 8$ is used as the default value, and CGLS is limited to 500 iterations.
- `elevel` $\in$ {A, AB, ABN, ABNT} or equivalently `elevel` $\in$ {DA, DB, DN, DT}. The energy levels after reduction by the baselines. DA refers to only dressed atoms as baseline, DB refers to both dressed atoms and dressed bonds as baselines, DN refers to baselines up to dressed angles, while DT refers to baselines up to dressed torsions.

7.3.1.3 Training set selection schemes There are two selection schemes compared in this series of experiments: random selection and selection by `usequence`. The random selection is straightforward: first one random seed is selected, e.g., `Random.seed!(603)` in Julia programming language, then generate a set of

unique 100 integers (indices) within the range of $[1, 130823] \cap \mathbb{Z}$. Table 17 shows the baseline MAEs of the random indices, while Table 18 is `usequence`'s.

For `usequence`, given each molecular features set it outputs stochastic sets of indices, here we generate 20 sets of those. Since currently we have 3 types of features (ACSF, SOAP, FCHL19), therefore we have 60 different sets of indices. The (training) MAE given each set is computed on both DA and DB level (see Table 30 in the appendices), the MAEs are sorted and then one of the set of indices which correspond to $1 < \text{rank} \leq 5$ is picked; the main reason is we want to avoid overfitting, and the first ranked set has the largest chance to overfit; since there are no validation scheme used in the training process, thus we want to pick the ones between the best and the middle-ground given the training MAEs.

This scheme will be applicable for the rest of the numerical experiments unless it is defined explicitly in the section.

Table 17: Baseline MAEs from randomly selected training set.

Elevel	solver	MAEtrain	MAEtest
DA	direct	18.020	20.264
DA	CGLS	18.020	20.264
DB	direct	10.861	13.966
DB	CGLS	10.861	13.966

Table 18: Baseline MAEs of the 2nd ranked training set from `usequence`.

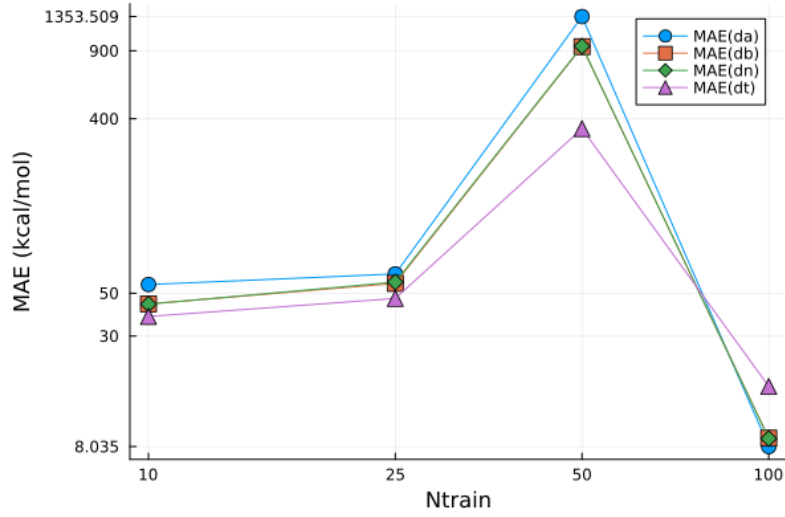
Elevel	solver	MAEtrain	MAEtest
DA	direct	15.143	21.467
DA	CGLS	15.143	21.467
DB	direct	9.808	12.962
DB	CGLS	9.808	12.962

7.3.2 Preliminary experiments of baselines, training-set selection, and features

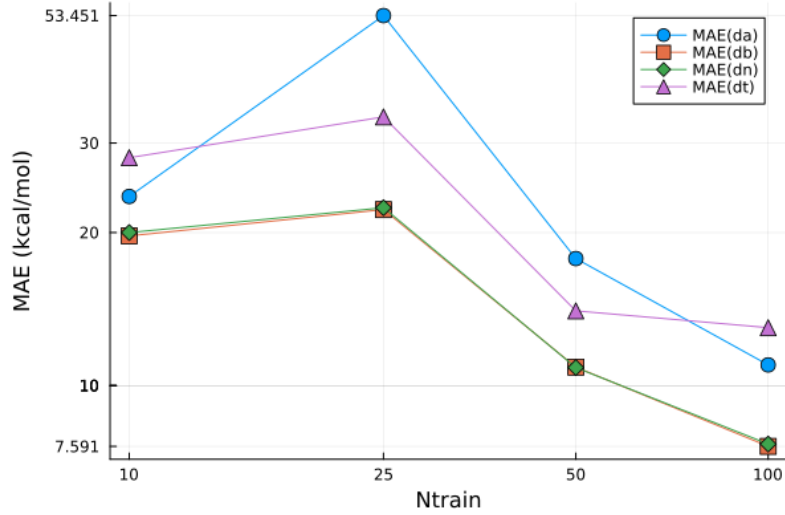
Here we see the effects when varying several hyperparameters, i.e., queries that output the row-subsets of Table 32 and Table 34. The queries in interest are:

- The effects of baselines: Here we want to see the effects of `elevel` on fixed `{feature, model, solver}`, for both random training set selection and `usequence`.

Table query: We query the row that corresponds to the lowest test MAE from each Table 32 and 34, then we fix the `{feature, model, solver}` given by the selected row for all `ntrain` for each table, this enumerates `{elevel, ntrain}` and set selection scheme. This gives two plots, one plot for each selection scheme; where each plot contains four curves, each curve corresponds to one `elevel` (Figure 27).



(a) Baselines on randomly selected training set. The lowest MAE found is on [FCHL19, GK, direct, DA] parameters, which is 8.035. Here the lowest MAEs of [DB, DN, DT] by setting [FCHL19, GK, direct] as the [feature, model, solver] are [8.878, 8.832, 16.452].



(b) Baselines on training set selected by usequence. The lowest MAE found is on [ACSF, DPK, cgl, DB], which is 7.591. Here the lowest MAEs of [DA, DN, DT] are [10.979, 7.673, 12.998] respectively on the mentioned [feature, model, solver].

Figure 27: Baselines curves (**DA, DB, DN, DT**) on (27a) randomly selected data, and (27b) a set selected by usequence. Here we can see that on both cases, DB as the highest level baseline performs the best; while DN is slightly worse than DB in performance, it is practically indistinguishable; DT on the other hand visibly increases the MAE compared to the other baselines, possibly due to higher variance.

- The effects of training-set selection: Here we want to see the effects of the training set selection schemes on a fixed {feature, model, solver, elevel}.

Table query: Table 32 and 34 are joined, then we query the row that corresponds to the lowest MAE, we call this the `minrow`; for each Table we query rows that contain the {feature, model, solver, elevel} of `minrow`, this enumerates `ntrain` and the selection scheme. This outputs one plot with two curves, each curve corresponds to one selection scheme. Ad-

ditionally, one curve is obtained by querying the `minrow` of Table 32 only and fixing `{feature, model, solver, elevel}` for all `ntrain` (Figure 28).

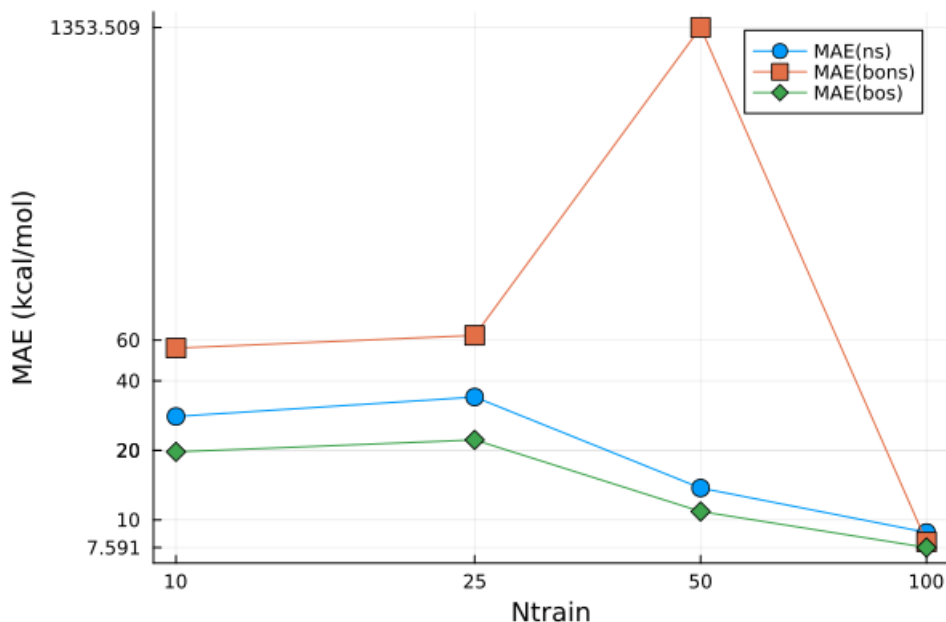


Figure 28: Comparison of random training set selection and usequence. **MAE(bos)** curve is the "best of selection", i.e., contains a point with the lowest MAE from the usequence selection, the `[feature, model, solver, elevel]` is `[ACSF, DPK, cgls, DB]`, the lowest MAE here is 7.591; **MAE(bons)** curve is the "best of non-selection", where the hyperparameters are `[FCHL19, GK, direct, DA]` i.e., from randomly selected training set, the lowest MAE is 8.035; while **MAE(ns)** curve is generated using the same `[feature, model, solver, elevel]` as **MAE(bos)**, it yields the lowest MAE of 8.823. Here we can see that usequence performs better than randomly selecting training set under the same hyperparameters and against the best performing randomly selected training set.

- The effects of the sets of features: Here we want to compare the best performances of each `feature` from all possible scenarios.

Table query: Table 32 and 34 are joined, then for each `feature`, we query the `minrow` then fix `{model, solver, elevel}` and the selection scheme given by the `minrow` for all `ntrain`. This procedure outputs one plot with three curves, each curve corresponds to each `feature` (Figure 29).

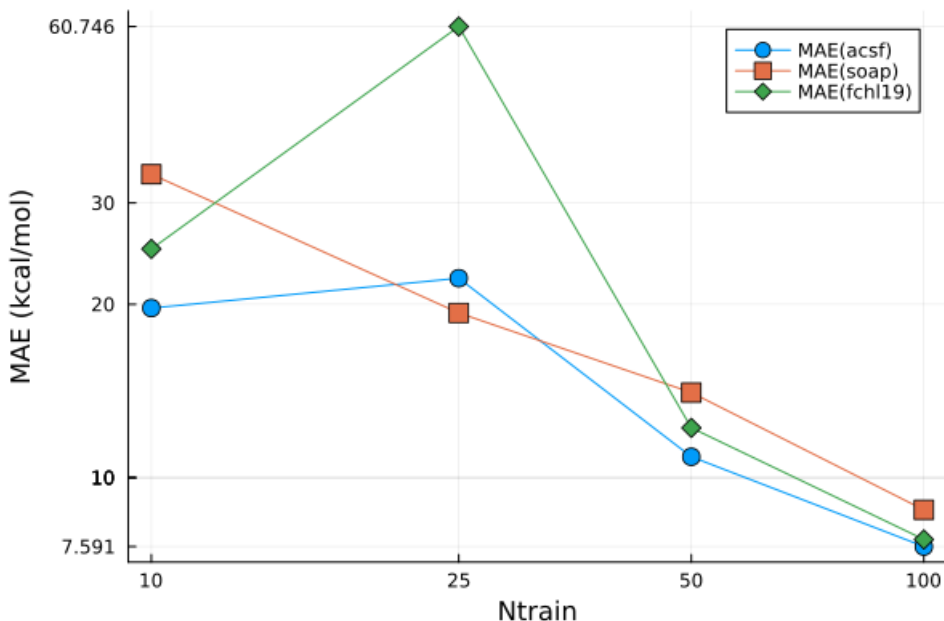


Figure 29: Comparison of the curves where each curve contain the best point of each feature (ACSF, SOAP, and FCHL19), both curves of ACSF and FCHL19 are obtained from usequence, while SOAP is from the randomly selected training set (SOAP performs best there). The **MAE(ACSF)** curve’s [model, solver, elevel] is [DPK, cgls, DB]; while **MAE(SOAP)** is under [DPK, direct, DA]; and **MAE(FCHL19)** is under [GK, cgls, DB]. The lowest MAEs are 7.591, 8.791, and 7.808 for ACSF, SOAP, and FCHL19 respectively.

7.3.3 Baselines with feature selection

From section 7.3.2, it can be observed that in general DB level of baseline performs the best, even though DN and DT are higher level of baselines. This is possibly due to higher variance., i.e., underdetermined system of linear equation (a wide matrix in contrast to tall and thin matrix which DA and DB possess). Here we want to see whether feature selection, or specifically, dimensionality reduction of DB, DN and/or DT would yield improvements to the fitting schemes.

PCA is used here to reduce the feature sets’ dimensions. Two groups of PCA are tested:

- PCA1: reduces the feature dimensions of [DN, DT] into [5, 5].
- PCA2: reduces the feature dimensions of [DB, DN, DT] into [5, 5, 5].

We already know from previous experiments that the global minima of the MAE is obtained from DB level of energy, therefore with PCA1 we want to see whether there are any improvements by fixing DB while transforming the higher level of baselines (DN and DT). As shown in Table 19, for DN level it is virtually indistinguishable, while DT’s improvement is noticeable, even though the global minima is still held by the non-PCA DB. By using PCA2 in general the MAEs of all levels increase when compared to PCA1, although PCA2 noticeably improves DT when compared to DT without PCA. This variance in results suggests hyperparameter optimization on the baseline space.

Table 19: The lowest MAE of each elevel and PCA mode. **None** means the original feature matrix is retained, **PCA1** means the dimensions of DN and DT are reduced to 5, and **PCA2** means the dimensions of DB, DN, and DT are reduced to 5.

PCA mode	feature	model	solver	elevel	MAE test
none	ACSF	DPK	cgl	DB	7.591
none	ACSF	DPK	cgl	DN	7.673
none	ACSF	DPK	cgl	DT	12.998
PCA1	ACSF	DPK	cgl	DN	7.673
PCA1	ACSF	DPK	cgl	DT	7.790
PCA2	FCHL19	GK	cgl	DB	7.624
PCA2	FCHL19	GK	cgl	DN	10.253
PCA2	FCHL19	GK	cgl	DT	10.920

7.3.4 Dressed atom hybridization

Here we want to see if a more specific atomic classification would yield better accuracy. In particular, we try the **orbital hybridization** for atomic classification. The only thing that we need to know about orbital hybridization in the context of atomic classification based on graph representation is that each atom is re-grouped into different classes depending on how it is bonded within the molecule.

Let p be the number of lone pairs of atom i , and q be the number of connectivity, i.e., the number of atoms atom i is connected to (by covalent bond), then

$$\text{Hybrid}(i) = \begin{cases} s & \text{if } p + q = 1, \\ sp & \text{if } p + q = 2, \\ sp^2 & \text{if } p + q = 3, \\ sp^3 & \text{if } p + q = 4, \\ sp^3d & \text{if } p + q = 5, \\ sp^3d^2 & \text{if } p + q = 6, \end{cases} \quad (305)$$

we are only interested in s up to sp^3 hybrid classes due to the limited atom types $\{H, C, N, O, F\}$ that exist in QM9 dataset; additionally, class s corresponds to standard atom type, i.e., equivalent to no hybridization. The hybridization results in atom types $\{H, C_{sp}, C_{sp^2}, C_{sp^3}, N_{sp}, N_{sp^2}, N_{sp^3}, O_{sp^2}, O_{sp^3}, F_{sp^3}\}$, notice that the hybridization is sparse for atom type F ($[sp, sp^2]$ do not exist for F in the QM9 molecules).

The comparison of baseline MAE of standard dressed atom (SDA) and hybridized dressed atom (HDA) is shown in Table 20, just to make things clear here the next level of dressed features (DB, DN, DT) are not hybridized. Table 21 shows the summary of full resolution fitting comparison between SDA and HDA; here a full batch fitting experiments which was done in section 7.3.2 is repeated with HDA in place of DA; we can see that on DA level, the best MAE found is improved by 0.9 kcal/mol while the global minimum (best from all) is slightly improved by 0.07 kcal/mol.

Table 20: The baseline MAE comparison of **standard DA** against **hybridized DA** of all **elevel**. **Standard DA** means the dressed atom level is fitted by atom types {H, C, N, O ,F}. **Hybridized DA** means the dressed atom level is fitted by {H, C_{sp}, C_{sp2}, C_{sp3}, N_{sp}, N_{sp2}, N_{sp3}, O_{sp2}, O_{sp3}, F_{sp3}}.

elevel	standard DA		hybridized DA	
	MAE train	MAE test	MAE train	MAE test
DA	15.143	21.467	9.832	12.784
DB	9.808	12.962	9.313	11.868
DN	9.731	12.938	9.331	11.887
DT	6.590	16.130	6.478	15.921

Table 21: Full fitting result comparison between standard and hybridized dressed atom types. The query "best on DA" means the MAEs are obtained from the lowest MAE of the full fitting where the baseline is only dressed atom **elevel**. The query "best from all" means the global minimum from all possible experiments. **SDA MAE test** means the full fitting test MAE where the dressed atom baseline uses standard dressed {H, C, N, O ,F} atom types. **HDA MAE test** means the full fitting test MAE where the dressed atom baseline uses {H, C_{sp}, C_{sp2}, C_{sp3}, N_{sp}, N_{sp2}, N_{sp3}, O_{sp2}, O_{sp3}, F_{sp3}} atom types.

query	SDA MAE test	HDA MAE test
best on DA	8.677	7.765
best from all	7.591	7.528

7.3.5 Benchmark: baseline learning curves

From section 7.3.2 and 7.3.3, in general DN and DT fell short when compared to DB on `ntrain` = 100, therefore we want to see whether they are still useful for experiments other than the QM9 challenge. First, we try to fit only the baselines with `ntrain` ≥ 100, using the combinations of the baselines encoded in the powerset of {*b*, *n*, *t*}, for example:

- *b* means only DB is involved, i.e., $E_t := E - E^{\text{DA}} - E^{\text{DB}}$
- *bt* means $E_t := E - E^{\text{DA}} - E^{\text{DB}} - E^{\text{DT}}$
- *bnt* means $E_t := E - E^{\text{DA}} - E^{\text{DB}} - E^{\text{DN}} - E^{\text{DT}}$
- The rest follows trivially

Notice that DA is always included by default. As shown in Table 35 and implied by Section 7.3.2, *b* baseline yields the lowest total MAE for the case of `ntrain` = 100. On the `ntrain` = 1000 regime, the full baseline fidelity of *bnt* yields the lowest total MAE. For 10k and 100k in general baselines that include *t* performs better.

Another batch of numerical experiments are done by fixing 100k data points as test set and the rest as training sets tested at different sizes `Ntrain`. The delta-level are increased from the dressed atom up to dressed torsion [*A*, *AB*, *ABN*, *ABNT*], with variations in the data selection methods [*rand*, *sid57*] that correspond to random selection and the identifier of one of the selected training sets generated by *usequence* respectively (*sid57* set was generated by the molecular

features computed from the atomic features of CMBDF); and the presence of orbital hybridization classes `[true, false]`.

Generally, features with higher resolution have higher variance and lower bias, and the opposite is also true. Moreover, features with high variance tend to have high error with less data points included for training and becomes more accurate with higher number of data points included for training, while features with low variance does not become more accurate with addition of training data points; as shown in Figure 30, `ABN` and `ABNT` curves have lower MAEs (higher accuracy) as `Ntrain` increases, while `A` and `AB` are generally stagnant.

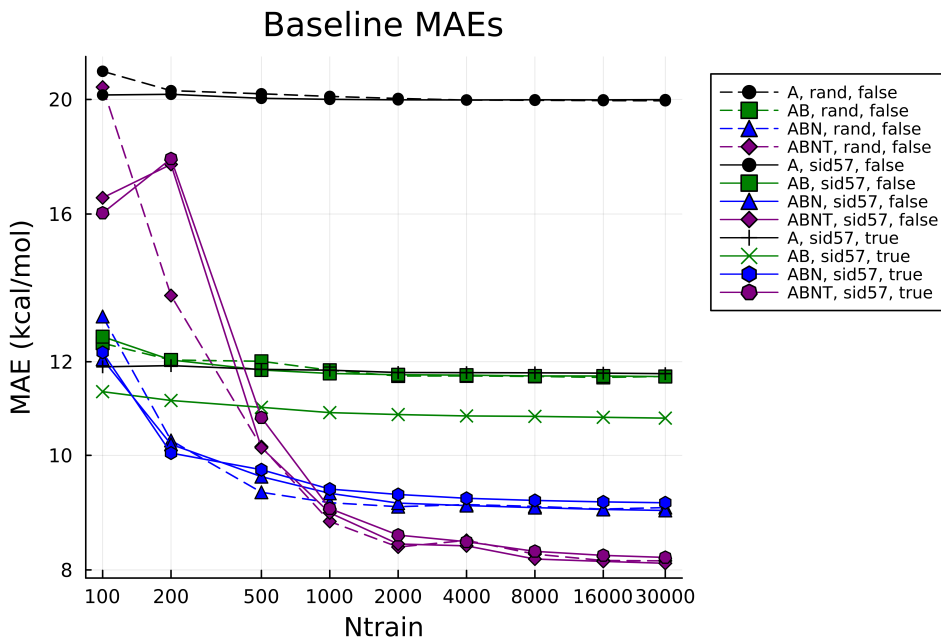


Figure 30: MAE of the dressed features in increasing delta-levels `[A, AB, ABN, ABNT]`, with variations of training set selection `[rand, sid57]`, and the presence of hybridization `[true, false]`.

7.3.6 Benchmark: learning curves of higher resolution features

In this section we do a benchmark to see what the learning curves look like when `Ntrain` ≥ 100 , sort of a sanity check to see whether the prediction accuracy can actually go down to 1 kcal/mol or below. Similar to section 7.3.5 we fix 100k data as test set and the rest as the training data where they are reorganized into many different sizes `Ntrain`.

For the setup, we enumerate the hyperparameters:

- `elevel` $\in \{\text{DA, DB, DN, DT}\}$,
- `model` $\in \{\text{GK, DPK}\}$,
- `hybrid` $\in \{\text{true, false}\}$,

then we keep some hyperparameters fixed: $\sigma = 32$, `solver = direct`, and `hydrogen = true`.

First we repeat the benchmark between `rand` and `usequence` selection method. The result is shown in Figure 31, the curves are generated using the lowest MAE from all of the possible hyperparameters for each combination of atomic feature, data selection method, and `Ntrain`. We can observe that for lower `Ntrain`, proper

training data selection is more relevant, while on larger `Ntrain` typically there are almost no difference in accuracy between `rand` and `usequence`, this is expected since the distribution of the training data becomes more dense hence the accuracy is more robust with respect to the composition of the training data, in contrast to lower `Ntrain` where the MAE is more sensitive to the composition of training data. We can also observe that only ACSF and FCHL19 go below the chemical accuracy (the red horizontal line) for $N_{train} \leq 30k$. On the point of interest, i.e., $N_{train} = 100$ (even though this benchmark does not necessarily fulfill the QM9 challenge since the test set size is 100k), CMBDF has the lowest MAE at 3.72 kcal/mol. Figure 32 shows the curves of minimum MAE obtained from all of the possible hyperparameters with the addition of the selection method for each atomic feature set, here MBDF is also included to see whether there are any significant differences when compared to CMBDF.

We did a rerun with $\sigma \in \{1, 2, \dots, 32\}$ to see whether the slope of the curves become steeper, which indeed it is the case for CMBDF in particular. The experiment is limited to $N_{train} \leq 1000$ and 100k test molecules. The result is in Figure (31). Additionally, with this training and tuning scheme, chemical accuracy is achieved with **3750** training molecules (0.989 kcal/mol accuracy), where the optimal hyperparameters use CMBDF and `usequence`.

The timing for the computation of each kernel that depends on the atomic feature is also a relevant part, since the timing is proportional to the size of the atomic feature data. Table 22 shows the average kernel computation time for each atomic feature on the largest kernel size with $100k \times 30k$ entries, i.e., $100k \times 30k \times 8e-6 = 24k \text{ MegaByte} = 24 \text{ GigaByte}$ worth of `Float64` data. Additionally, the mean total time of kernel computation, training, and evaluation combined for $N_{train} \leq 1000$ is reported in Table 33.

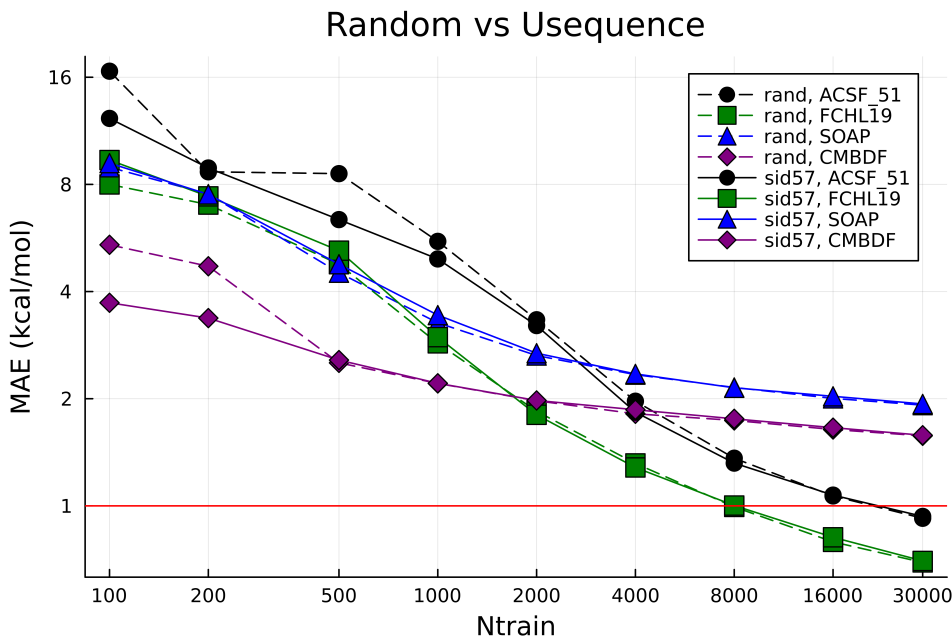


Figure 31: Benchmark of `rand` vs `usequence` (represented by sequence `sid57`) for larger values of `Ntrain`.

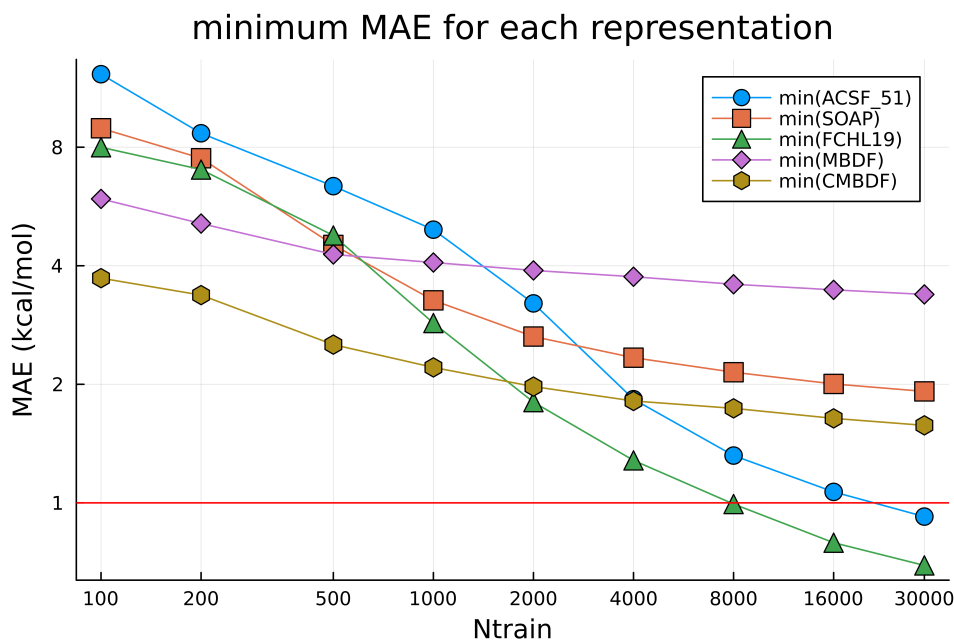


Figure 32: The comparison of minimum MAEs obtained from each atomic feature.

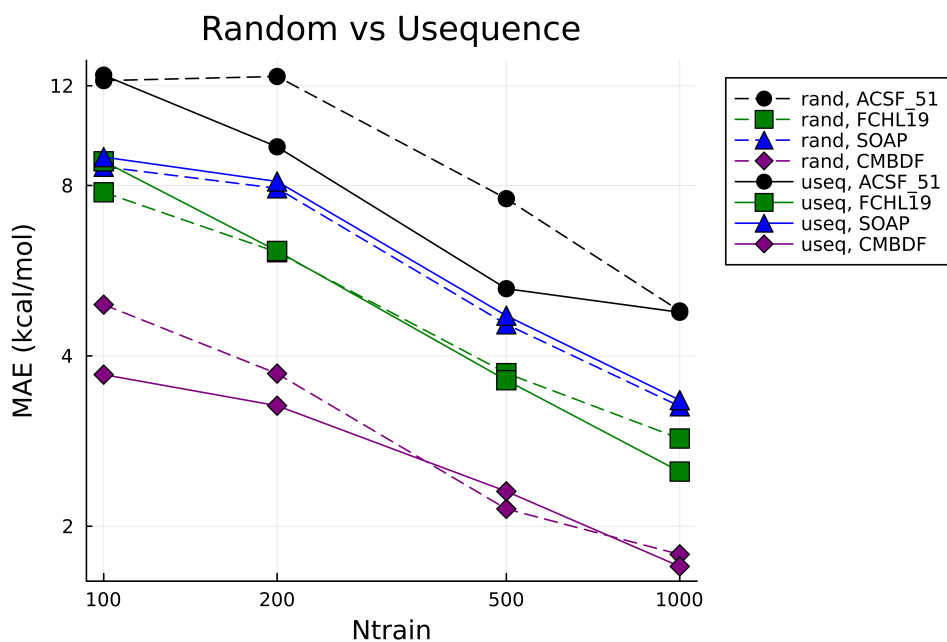


Figure 33: Benchmark of `rand` vs `usequence` (represented by sequence `sid57`) for `ntrain` up to 1k. MAEs are evaluated on 100k test molecules.

Table 22: Computation time of $100k \times 30k$ kernel for each atomic feature in seconds

feature	timing GK	timing DPK
ACSF	824.960	691.597
SOAP	8223.429	7943.422
FCHL19	5827.073	6223.928
MBDF	352.116	454.746
CMBDF	766.063	697.997

Table 23: Mean of the total time (in seconds) of the kernel model computation, training, and evaluation using 100k test molecules for each feature on each `Ntrain`.

ntrain	Feature			
	ACSF	FCHL19	SOAP	CMBDF
100	2.542	16.766	21.027	1.837
200	4.123	32.680	41.401	3.321
500	10.719	79.640	101.847	8.362
1000	20.149	155.616	207.597	16.796

7.3.7 Hyperparameter tuning intensification for `ntrain` = 100

With respect to the QM9 challenge, the main goal is reaching chemical accuracy (or lower) with only 100 training data ($< 0.1\%$ of total data), we want to find the most optimal configuration for the fitting procedure. Here we attempt to optimize the hyperparameters that are listed in section 7.3.1.2 with the addition of a few extra parameters, by setting the MAE as the objective function in a minimization problem.

We assume a black box mixed integer optimizer. In order to feed the hyperparameters as input variables of the optimizer, we need to encode the variables into numerical form (a list of real and integer values). Special consideration need to be done for categorical variables, for example, `feature_name` is encoded as categorical type hyperparameter which is a (block) vector of binaries, if we choose MBDF from `list_feature = {"ACSF", "SOAP", "FCHL19", "MBDF", "CMBDF"}`, then

```
encode("MBDF", list_feature) = [0,0,0,0,1],
```

the value of the vector is 1 only at the position of the selected element of the list while the rest are 0, for simplicity here we just write the position of the 1 value, e.g., from the example before then

$$[0,0,0,0,1] \implies 6.$$

The hyperparameter space is:

1. `Edb_switch` $\in \{0,1\}$, the choices of whether to include DB level of energy,
2. `Edn_switch` $\in \{0,1\}$, same but for DN,
3. `Edt_switch` $\in \{0,1\}$, same but for DT;
4. `Edb_PCA_switch` $\in \{0,1\}$, the choices of whether to apply PCA for DB feature,
5. `Edn_PCA_switch` $\in \{0,1\}$, same but for DN,
6. `Edt_PCA_switch` $\in \{0,1\}$, same but for DT;
7. `num_Edb_f` $\in \{1,2,..,10\}$, size of DB feature outputted by PCA, this variable will only be activated (effective) if `Edb_switch` = 1 and `Edb_PCA_switch` = 1,
8. `num_Edn_f` $\in \{1,2,..,10\}$, same but for DN,

9. `num_Edt_f` $\in \{1, 2, \dots, 10\}$, same but for DT;
10. `fmol_PCA_switch` $\in \{0, 1\}$, the choices of whether to apply PCA for molecular feature,
11. `fatom_PCA_switch` $\in \{0, 1\}$, the choices of whether to apply PCA for atomic feature;
12. `num_fmole` $\in \{1, 2, \dots, 50\}$, the numbers of molecular feature, this variable will only be effective only if `fmol_PCA_switch` = 1;
13. `num_fatom` $\in \{1, 2, \dots, 50\}$, the numbers of atomic feature, this variable will only be effective only if `fatom_PCA_switch` = 1;
14. `n_basis` $\in \{1, 2, \dots, 10\}$, the numbers of basis feature used for the ROSEMI kernel, this variable will only be effective if `model` = "ROSEMI" (see `model` parameter);
15. `feature_name` $\in \{\text{"ACSF", "SOAP", "FCHL19", "MBDF", "CMBDF"}\}$, categorical variable, the choices of atomic feature to be used as the last level of delta learning;
16. `normalize_atom` $\in \{0, 1\}$, the choices of whether to normalize the atomic feature;
17. `normalize_mol` $\in \{0, 1\}$, the choices of whether to normalize the molecular feature;
18. `model` $\in \{\text{"ROSEMI", "KRR", "NN", "LLS", "GK", "DPK"}\}$, categorical variable, the choices of machine learning models. **ROSEMI** refers to robust shepard model for interpolation [100], **KRR** refers to global Gaussian kernel, **NN** refers to neural networks, **LLS** refers to linear least squares with global rectangular matrix; while **GK** and **DPK** are local Gaussian and linear kernel respectively;
19. `const` $:= 1 + 2 \log_2(\sigma) \in \{1, 2, \dots, 20\}$, this variable will only be effective if `model` = "KRR" or `model` = "GK";
20. `solver` $\in \{\text{"CGLS", "direct"}\}$, categorical variable, solver type to solve the linear model;

We utilize a black box mixed integer optimizer, **matrix adaptation trust-region strategy** (MATRS) [89, 90] for the hypertuning. The hyperparameter optimization gives varying results depending on how close the objective function is to the chemical accuracy target — in general, the closer it is, the less gain the hyperparameter optimization gives. The summary of the hypertuning results is in Table 24.

Chronologically, the first run of the hyperparameter optimization did not include MBDF and CMBDF, as both of them were not available yet; accuracy gain of 0.5 kcal/mol were obtained, and a total of 8689 iterations were done and the minima were found in the 4085th iteration. The next experiments were done when MBDF was available; a total of 3829 iterations were done, and the minima were found in the 500th iteration; accuracy gain of 0.5kcal/mol were also obtained. Then finally with CMBDF, the lowest point were found at the 115th iteration from a total of 9824

iterations; this gives only 0.05kcal/mol accuracy gain. For each set of experiments, the starting point used is the best training set generated from `usequence`, and we fix it for all of the iterations, i.e., no change or no new training set generation during the optimization process. For more details of the hyperparameter tuning results, see Appendix C.2.

Additionally, the CPU timing for each run of prediction of 130k molecules takes around **2.9 seconds** on a 64 cores CPU using the current best found hyperparameters.

Table 24: The MAE of the initial point and the minima found after hypertuning. Additionally the total iterations and the iteration in which the minimizer is found for each set of computational experiments is also reported.

	Set 1 (ACSF)		Set 2 (MBDF)		Set 3 (CMBDF)	
	start	minima	start	minima	start	minima
MAE	7.59	7.09	5.78	5.29	3.72	3.67
total iteration	8689		3829		9824	
minimizer iteration	4085		500		115	

7.3.8 Optimal training set selection

In the previous sections we have seen the performance of `usequence` when used as a one-shot unsupervised data selection technique (completely detached from the computation of the objective function). Here we combine both `usequence` and supervised data selection technique in Section 7.2.2.

We use the Tabu search described in Algorithm (4) for the supervised selection process. In particular for the algorithm’s configuration, the input pairs $p^{(0)}$ and $P^{(0)}$ can be obtained by setting $p^{(0)} := f(x^{(0)})$ for all available $x^{(0)}$ and storing all $x^{(0)}$ into $P^{(0)}$; where $x^{(0)}$ and $f(x^{(0)})$ can be obtained from simulation data or by spawning many random training sets and evaluating (301) for each training set (it must also comply to (302) by only selecting 100 data points for each training set). The `usequence` is used to generate $P^{(0)}$ where $|P^{(0)}| \geq 1000$ instead of randomly selecting the data points, in other words the training sets from `usequence` are the initial starting points for the optimization process. The reset mechanism is useful to avoid being trapped in local minima, usually this is done after there are no gains within some numbers of iterations. In the numerical experiments, we sample 10k data points as the test set to compute the objective function (301) instead of the whole 130k, this significantly increases the computation speed by a factor of 50 while only sacrificing the accuracy in the 0.01 kcal/mol range.

Each iteration of the Tabu search takes only **0.1 second** on average in a serial process, and only $3 \times 130k$ entries of `Float64` need to be kept track of by the algorithm that is equivalent to **3.2MB** (independent of the memory requirement for the models). The initial (data) points include the current best point so far and points with high penalties. Points with high penalties are included in the hope that the algorithm would learn to exclude them earlier. After around 100k iterations, the current best found point is $\text{MAE} := f(x) = 3.214$ kcal/mol, and it gives $\text{MAE} = \mathbf{3.25}$ kcal/mol when evaluated on 130k data points as test set (Figure 34). Figure 36 shows the information of the 100 selected molecules as the final solution of the optimization process. Figure 35 shows the atomization energies’ distributions of the 100 selected molecules and all of the QM9 molecules.

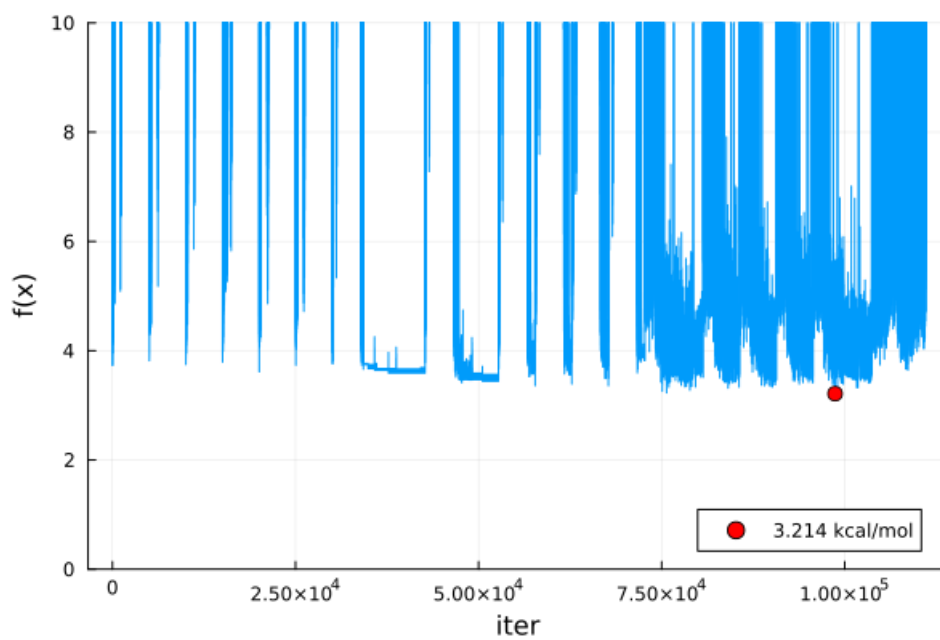


Figure 34: Objective values found after around 100k iterations of Tabu search.

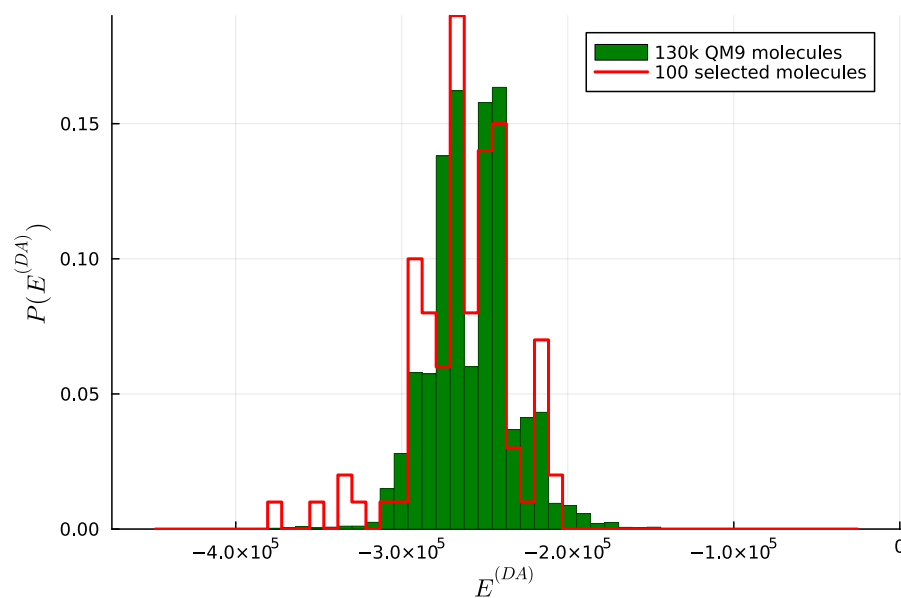


Figure 35: The atomization energies' $E^{(DA)}$ (in kcal/mol) distributions $P(E^{(DA)})$ of the 100 selected molecules and all of the 130k QM9 molecules. The distribution of each set is normalized such that the sum of the bins' height is 1.

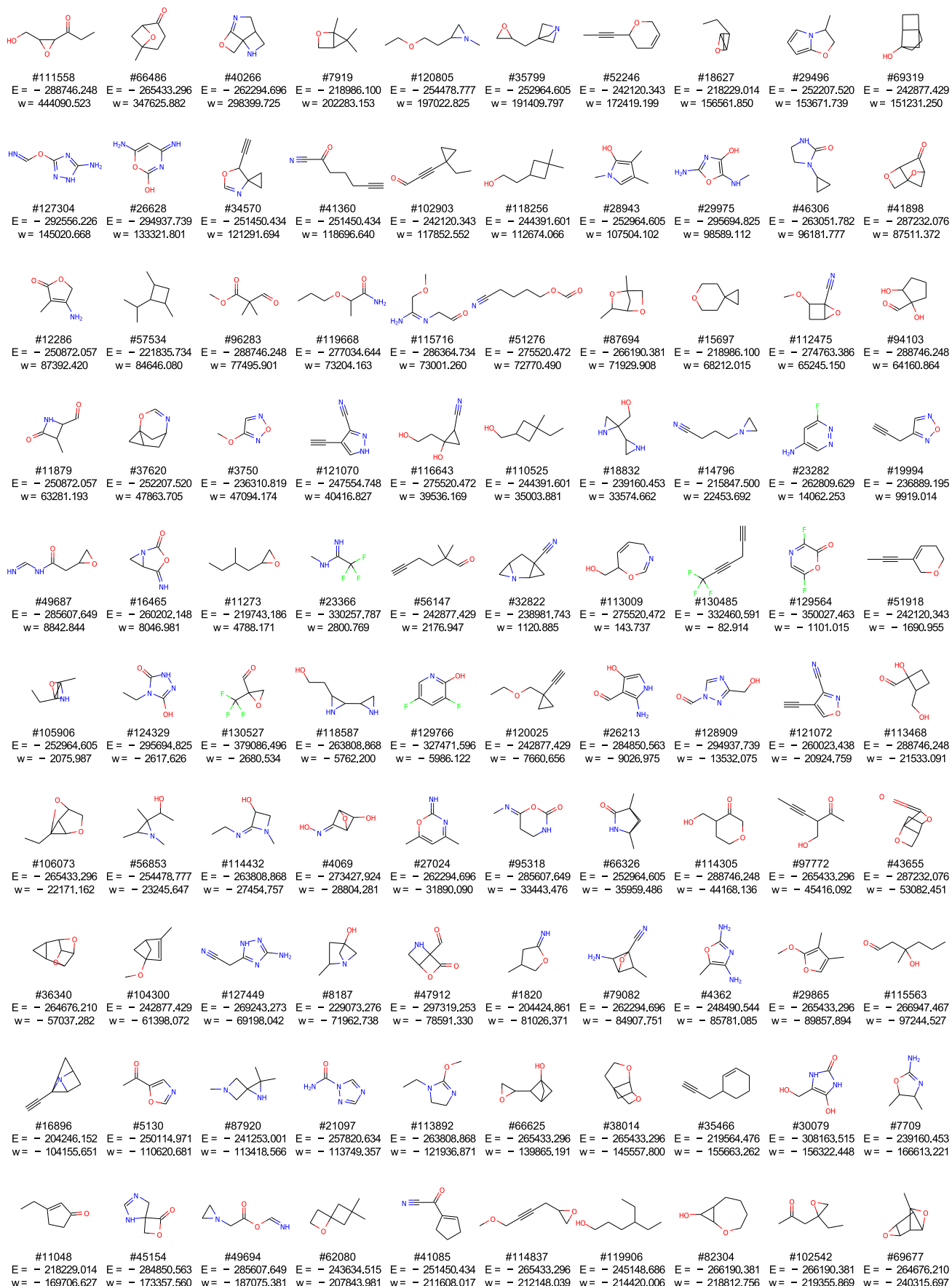


Figure 36: The molecular graphs of the 100 selected molecules that correspond to the training set with the lowest test MAE after optimization processes. The molecules are placed in a row-major ordering based on the sorted (descending) weights θ of the model computed from training (see (74)). The text under the molecules are: the label (index) of each molecule relative to the preprocessed QM9 dataset; the atomization energy $E := E^{(DA)}$ (in kcal/mol); and the corresponding weight $w \in \theta$ (in kcal/mol).

7.3.9 Optimal level of delta learning

In this section, we want to see the extent of delta learning in helping the higher resolution of atomic features in reducing the prediction errors: we hypothesized that atomic features with higher resolution, e.g., the ones that contain higher n -body terms would depend less on the dressed features, and vice versa.

We fix 100 training molecules (the previously generated training set [sid57](#)) and the rest of 130k molecules for prediction. The atomic features for benchmark are ACSF, SOAP, FCHL19, MBDF, CMBDF, CM, and BOB. We have already seen ACSF up to CMBDF perform in the previous experiments, in fact, those are atomic features that contain some variations of 3-body terms, while CM and BOB contain variations of 2-body terms only. This suggests that CM and BOB may need higher level of dressed feature to enhance the accuracy. As in [Table 25](#), indeed, lower resolution atomic features get better gain of accuracy from higher level of dressed feature. BOB and CM gain accuracy up to dressed angle (DN) level of dressed feature; while the rest of the atomic features that contain 3-body terms either can receive benefits up to dressed bond (DB), or dressed atom (DA) is already sufficient; while dressed torsion (DT) is useful only when used independently (see [Figure 30](#)).

Table 25: The minimum prediction MAE of 130k molecules (trained using molecules in [Figure 36](#)) for each level of dressed fragment [Dressed Atom (DA), Dressed Bond (DB), Dressed Angle (DN), Dressed Torsion (DT)] and atomic features [ACSF, SOAP, FCHL19, MBDF, CMBDF, CM, BOB]. The bold numbers represent the minimum MAE for each atomic features from all of the dressed features.

	ACSF	SOAP	FCHL19	MBDF	CMBDF	CM	BOB
DA	10.937	10.199	8.620	5.664	3.250	21.625	20.200
DB	8.050	9.687	9.013	9.142	5.261	13.423	12.721
DN	10.152	10.479	9.866	10.880	8.524	12.452	11.917
DT	38.099	38.500	38.481	38.391	37.544	38.696	38.762

7.4 Explainable AI (X-AI): principal components of kernel feature

In this section we wish to see the distribution of the most optimally selected data in the feature space. In particular, the 2D visualization of the feature space serves as the basis of the analysis.

We fix the current best configuration from all of the previous experiments, in terms of the hyperparameters generated in [section 7.3.7](#). Here we used the best training set obtained from the Tabu search that is listed in [Figure 36](#). This configuration gives 3.25 kcal/mol as we have seen in [section 7.3.9](#).

First, the Gaussian kernel is computed given the CMBDF atomic features (the hyperparameter configuration used is in [C.2](#)), then we perform a PCA on the kernel where it only returns two of the most important feature vectors that corresponds to the two largest eigenvalues: 99.124 and 0.495. The first eigenvalue dominates the second component by a large margin, this implies that only the first principal component is significant, however for visualization purpose we include the second principal component. For the 2D visualization, we project the Gaussian kernel using the eigenvectors that correspond to the mentioned eigenvalues, this outputs a coordinate matrix with 2 columns, where each column corresponds to one of the axis (dimension), and each row corresponds to one data point, then we re-scale each

column into $[0, 1]$; this results in the distribution of the data points as shown in Figure 37.

We observe diagonal linear pattern in Figure 37, this may imply a pattern in the chemical space, in particular, in terms of the atomic composition of the molecules. To bring the pattern to light, we sample 3 linear equations f_1, f_2, f_3 , where $f_i : \mathbb{R} \rightarrow \mathbb{R}$, and then classify each molecule into a cluster $i \in \{1, 2, 3\}$, based on the proximity criteria

$$|f_i(x) - y| \leq \delta y,$$

where x is the first principal component value PC1 of a molecule, y is the second principal component value PC2 of a molecule, and $0 < \delta y < 1$ is the proximity threshold, here we use $\delta y = 0.07$. Once the clusters are formed, for each cluster we sort the molecules based on the magnitude of PC2.

By observing the clusters, we can generalize a pattern based on the number of atoms within each molecule. In Table 26, it is visible that the frequency distribution of the number of atoms `n_atom` shifts from one cluster to another. In particular, it appears that the distributional peak shifts by 2 values of `n_atom` when moving from a cluster to its next. If we assume the clusters as parallel lines where each cluster is $y_i = ax + b_i$ (similar to f_i) and the only difference between the clusters are the values of b_i , then the clusters with lower values of b_i have distributional peak that are located on lower values of `n_atom`, and vice versa.

A pattern that relates to the atomic composition also emerges, as seen in Figure 38. For example, at the second column at the bottom left, the two lowest molecules are almost isomorphic. Another one is at the rightmost columns, it is visible that the molecules contain more than one Fluorine atoms.

Table 26: The frequency of `n_atom` in clusters $i = 1, 2, 3$.

<code>n_atom</code>	$i = 1$	$i = 2$	$i = 3$
9	0	0	3
10	0	0	20
11	0	0	98
12	0	20	377
13	0	103	1243
14	3	505	3125
15	58	1576	6278
16	220	4118	9331
17	734	9178	7018
18	2291	15059	35
19	5801	12173	0
20	11817	8	0
21	11557	0	0
22	0	0	0
23	0	0	0

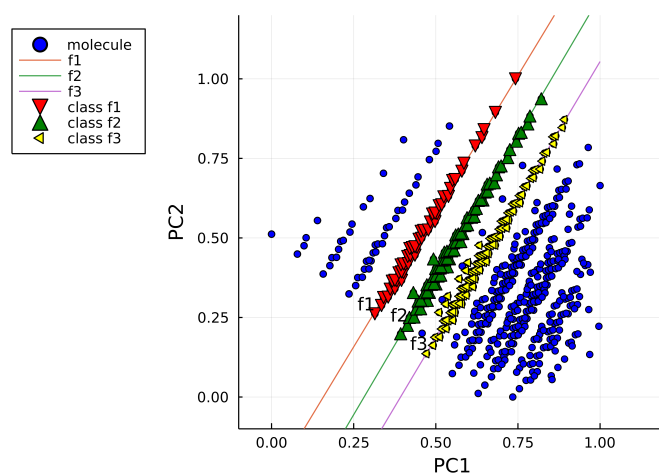


Figure 37: Superpositioned lines f_1, f_2, f_3 and their corresponding clusters on top of the principal components' coordinates of the kernel.

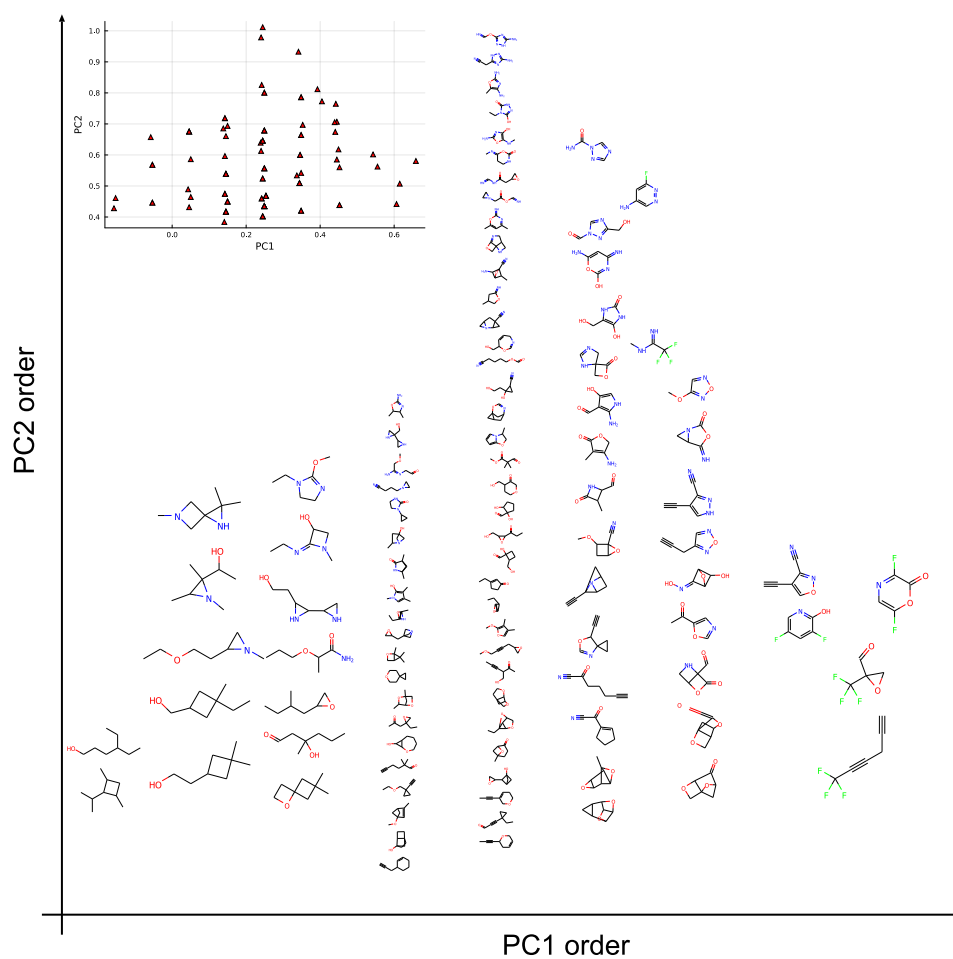


Figure 38: Molecular graphs of the 100 training molecules placed based on the ordering of transformed (**PC1**, **PC2**) values. A subplot on the top left shows the precise (**PC1**, **PC2**) coordinates of the training sets after (**PC1**, **PC2**) coordinates transformation.

8 High-performance code in Julia

Most parts (specifically, Chapters 2, 5, 6, and 7) of the mathematical modelling of this thesis is written in Julia programming language [99]. One of the major advantage of Julia for scientific computing is that it is easier to write customizable performant code – even a simple loop without vectorization is fast. Therefore Julia end up as the language of choice for the majority of this thesis.

In this Chapter we present (non-exhaustive) examples of performance increase that were found during the implementation of Julia code related to this thesis. For more comprehensive list of performance guide for Julia, the reader is referred to [Julia performance tips documentation](#).

8.1 Minimizing memory allocation

One of the largest performance bottleneck in any programming languages is the memory allocation. In Julia, memory allocation management is a built-in feature in which the programmer can control. Knowing when to allocate or not to allocate memory is the very first step to minimize the execution time of program blocks. We present a simple case of timing differences between when the computation uses more memory allocations compared to when the computation uses less memory allocations. We investigate the computation of Gaussian kernel function (70) as a case study.

First, we implement the kernel computation in a vectorized way, which is typically preferred in Matlab-esque programming languages:

```
"""
computes  $\exp(-||u-v||^2_2/c)$ 
u and v are vectors; c is a scalar
"""
function kernel_fun_1(u, v, c)
    s = (u - v).^2 # here s is a vector
    return exp(-sum(s)/c)
end
```

Next, we implement the kernel in a natural `for-loop` as a separate function:

```
function kernel_fun_2(u, v, c)
    s = 0. # here s is a scalar
    for i in eachindex(u)
        s += (u[i]-v[i])^2
    end
    return exp(-s/c)
end
```

We proceed with some test variables:

```
u = rand(100)
v = rand(100)
c = 1024.
```

Running the functions using the variables, gives the output shown in Figure 39. It is interesting to see that natural `for-loop` is preferred in this case over vectorization, as shown by the execution time output, where for a very simple case of two vectors with size 100 already gives a $\sim 1.7\times$ speed up and $1000\times$ memory efficiency. Since

in this case, vectorization leads to vector memory allocation in contrast to a scalar.

```
julia> @btime kernel_fun_1(u, v, c)
169.889 ns (3 allocations: 1.77 KiB)
0.9837710889033627

julia> @btime kernel_fun_2(u, v, c)
100.847 ns (1 allocation: 16 bytes)
0.9837710889033627
```

Figure 39: Timing comparison of kernel function that allocates more to the memory against the kernel function that allocates less memory. Here with the `@btime` macro for each Julia command, the first line shows the command entry, the second line shows the execution time and the size of memory allocations, and the third line shows the command's output.

If we upscale the computation, for example, by computing an entry of K in (73), i.e., for between a pair of matrices U and V only (corresponds to a pair of molecules), then the efficiency gap becomes increasingly much larger. This can be implemented as the following function:

```
"""
U and V are real matrices with row size equal to the size of lU and lV
respectively,
and uniform column sizes.
lU and lV are vectors of booleans.
"""
function kernel_atom(U, V, lU, lV, c)
    s = 0.
    for i ∈ eachindex(lU)
        for j ∈ eachindex(lV)
            if lU[i] == lV[j] # equivalent to the Kronecker delta
                d = kernel_fun_1(@view(U[i, :]), @view(V[j, :]), c)
                s += d
            end
        end
    end
    return s
end
```

Now, we declare the input data variables:

```
lU = rand{Bool, 100}; lV = rand{Bool, 100} # lists of booleans
rowsizes = length.([lU, lV])
U = rand{Float64, rowsizes[1], 200} # matrix of float64
V = rand{Float64, rowsizes[2], 200} # matrix of float64
```

We call `kernel_atom` function first using `kernel_fun_1` as the inner kernel function and then next using `kernel_fun_2` replacing `kernel_fun_1` as the inner kernel function, the output is as shown in Figure 40. We see that the efficiency gap increases, by $\sim 2\times$ execution time efficiency and the memory allocation size gap grows by over a factor of 10^6 . This case is rooted only on one inner function memory allocation, if the functions are more complicated and the memory allocation is unmanaged then surely the code efficiency gap would be much larger. Therefore memory allocation management remains one of the most critical part of code efficiency optimization.

```

kernel_atom

julia> @btime kernel_atom(U, V, lU, lV, c)
1.824 ms (18129 allocations: 17.46 MiB)
4902.418783167212

kernel_atom

julia> @btime kernel_atom(U, V, lU, lV, c)
864.900 μs (1 allocation: 16 bytes)
4902.418783167212

```

Figure 40: Upscaled timing comparison. The first command calls `kernel_atom` using `kernel_fun_1` as inner function, the second command calls `kernel_atom` using `kernel_fun_2`.

8.2 Multi-threading

To maximize the usage load of the processor so that the computation time is minimized, we can make use of multi-threading. With multi-threading activated, processes can be distributed to many or all threads available in a processor, therefore, depending on the code, instead of waiting for the previous process to be finished, the next processes can fill up the empty threads and be executed there, hence the speed up. We continue to use (73) as an example, now to compute the whole matrix K . The function to compute K can be implemented as the following:

```

"""
Computes K (a matrix) from Gaussian function entries, given:
Us and Vs are lists of real matrices,
lUs and lVs are lists of boolean vectors,
"""
function gaussian_kernel(Us, Vs, lUs, lVs, c; threading=true)
    if threading
        rowids = eachindex(lUs); colids = eachindex(lVs)
        iterids = Iterators.product(rowids, colids)
        K = ThreadsX.map((tuple_id) -> kernel_atom(Us[tuple_id[1]], F2[tuple_id[2]], lUs[tuple_id[1]], lVs[tuple_id[2]], c), iterids)
    else
        nm1 = length(lUs); nm2 = length(lVs)
        K = zeros(nm1, nm2)
        for j ∈ eachindex(lVs) # col
            for i ∈ eachindex(lUs) # row
                K[i, j] = kernel_atom(Us[i], Vs[j], lUs[i], lVs[j], c)
            end
        end
    end
    return K
end

```

It can be noticed that the multi-threading activation uses a built in `map` function from `ThreadsX` [101]. With the aforementioned multi-threading implementation, each pair of i and j data in (73) is now assigned to a thread until a maximum number of available threads, . Before running anything, we need to open Julia with multiple threads, for example, by calling in the console:

```
julia --threads 8
```

This will allocate 8 threads. Then check if we have correctly activated 8 threads, such as in Figure 41.

```
julia> Threads.nthreads()
8
```

Figure 41: Command to check the number of active threads.

As the threads are ready, we declare the input data variables:

```
Us = [rand(rowsizes[1], 200) for i in 1:100] # list of real matrices
Vs = [rand(rowsizes[2], 200) for i in 1:100] # list of real matrices
lUs = [rand{Bool, 100} for i in 1:100] # list of boolean vectors
lVs = [rand{Bool, 100} for i in 1:100] # list of boolean vectors
```

Note that we reuse the `rowsize` vector and `c` scalar from Section 8.1. First, we call the function using standard double `for-loop`, then we call the function using multi-threading mode by setting `threading=true`, the output is as shown in Figure 42. The output shows that with multi-threading with only 8 threads, the execution time is already $\sim 6.4\times$ faster than usual; it can be noticed that there is additional memory allocation overhead when using multi-threads, although it is still within tolerance, and it is expected since more threads are used.

```
julia> @btime gaussian_kernel(Us, Vs, lUs, lVs, c; threading=false);
8.910 s (2 allocations: 78.17 KiB)

julia> @btime gaussian_kernel(Us, Vs, lUs, lVs, c; threading=true);
1.398 s (434 allocations: 906.17 KiB)
```

Figure 42: Efficiency comparison between standard double loop and multi-threading.

8.3 Automatic differentiation (AD)

8.3.1 Background and simple example

AD is a tool to compute derivative of a function, by taking advantage of the storage of intermediate computations to propagate values between the primitive operations, and then obtaining the derivative of each primitive operation accumulated in some mechanism. AD is different to finite differences, AD gives exact derivatives while finite differences gives approximate derivatives; and it is also well known that, for example, for a function $f : \mathbb{R}^N \mapsto \mathbb{R}$, finite differences have $2NF + 2$ computation cost, where F is the cost of evaluating f , while AD only requires small multiple of F to compute both f and its derivatives. AD is also different to symbolic differentiation since AD is *automatic*, i.e., one AD algorithm can compute derivatives of any differentiable functions without (almost) any other prior assumptions (with some limitations, typically depending how the functions and the automatic differentiation routines are written in the program code); while in contrast for symbolic differentiation, it is rigid that it only works for differentiation of proper mathematical functions, moreover symbolic differentiation suffers from bloated expression.

Currently there are two main classes of AD: Forward mode AD, and reverse mode AD. In particular for machine learning and optimization, reverse AD is more desirable since its computational cost scales on the number of output rather than input, and it is very common to have much larger number of input compared to output for machine learning and optimization applications. Therefore for this section we focus on reverse AD.

Conceptually, reverse AD starts with a **forward sweep** to compute the function value while storing the intermediate values, then the **backward sweep** uses of the intermediate values to compute the derivative, from the output all the way to the input. Consider a simple example, evaluating $f(x)$ and its gradient at $x = [2, 1]$:

$$f(x) = (x_1 + x_2)(x_1 + 1), \quad f([2, 1]) = 9. \quad (306)$$

Manually, we can compute the gradient and obtain

$$g(x) := \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right]^T = [2x_1 + x_2 + 1, x_1 + 1]^T, \quad g([2, 1]) = [6, 3]^T. \quad (307)$$

Now we proceed to compute (307) from (306) by reverse AD. Reverse AD uses computational graph to store intermediate values and to accumulate the gradient, the computational graph for computing the gradient of (306) is in Figure 43.

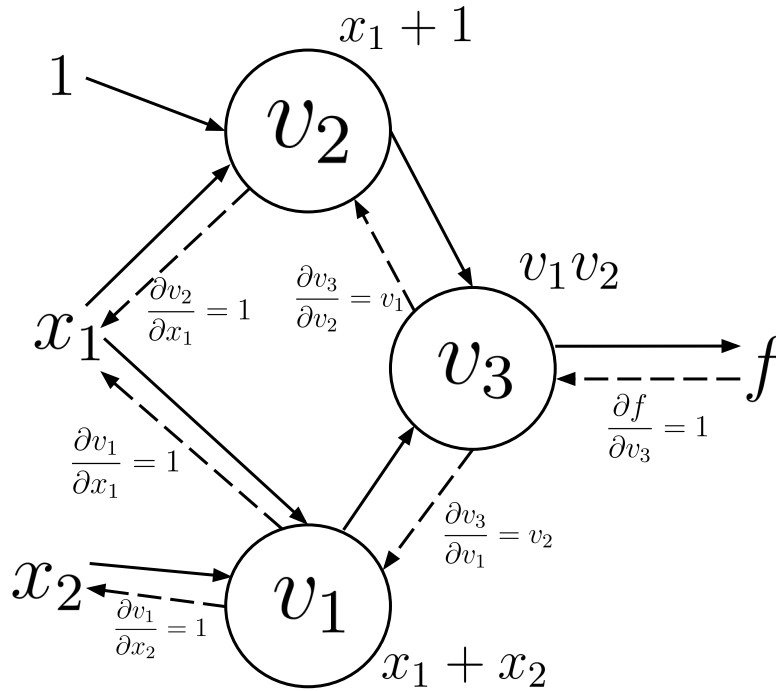


Figure 43: Computational graph for computing (306) and (307).

The forward sweep computes $f(x)$ while also storing the intermediate values v_i as the following:

$$\begin{aligned} v_1 &:= x_1 + x_2 = 3, \\ v_2 &:= x_1 + 1 = 3, \\ v_3 &:= v_1 v_2 = 9, \\ f &:= v_3 = 9. \end{aligned} \quad (308)$$

The backward sweep uses the intermediate values v_i and chain rule to compute $g(x)$ by following the path of the computational graph starting from the output f propagated back to the input $[x_1, x_2]$. First we compute $\frac{\partial f}{\partial x_1}$:

$$\begin{aligned} \frac{\partial f}{\partial x_1} &= \frac{\partial f}{\partial v_3} \frac{\partial v_3}{\partial v_1} \frac{\partial v_1}{\partial x_1} + \frac{\partial f}{\partial v_3} \frac{\partial v_3}{\partial v_2} \frac{\partial v_2}{\partial x_1}, \\ \frac{\partial f}{\partial x_1} &= 1 \cdot v_2 \cdot x_2 + 1 \cdot v_1 \cdot 1 = 6. \end{aligned} \quad (309)$$

Notice that since x_1 has two branches, then the total value of the partial derivative with respect to x_1 is the sum of the derivatives with respect to x_1 propagated from the two branches. Next we compute $\frac{\partial f}{\partial x_2}$:

$$\begin{aligned}\frac{\partial f}{\partial x_2} &= \frac{\partial f}{\partial v_3} \frac{\partial v_3}{\partial v_1} \frac{\partial v_1}{\partial x_2}, \\ \frac{\partial f}{\partial x_2} &= 1 \cdot v_2 \cdot 1 = 3.\end{aligned}\tag{310}$$

Therefore reverse AD also gives $g([2, 1]) = [6, 3]^T$.

If we count the number of operations in total in evaluation $f(x)$ and $g(x)$, the forward sweep has $F = 3$ operations, and the backward sweep has 4 operations, a total of only 7 operations which is just in the order of $2F$ with an extra small constant. The multiplication with output node $\frac{\partial f}{\partial v_n}$ is not counted since $\frac{\partial f}{\partial v_n} = 1$ for any function. Indeed, reverse AD does not scale with the numbers of input variables N . If we compare this with finite differences, then to compute the same gradient, it would require $4F + 2$ operations.

8.3.2 Implementation example in Julia

In Julia, AD is fast, since computational graphs in Julia are implemented as a first class data structure, on top of inherent Julia's computational efficiency. AD is implemented in various libraries in Julia, e.g., [ReverseDiff](#) [102], [Zygote](#) [103], [Enzyme](#) [104], and many others. Here we give an example of computational efficiency boost by AD implementation in Julia.

Suppose we want to minimize a least squares objective function given the matrix K computed from (73) and the energy vector E to obtain the minimizer vector θ

$$\min_{\theta} \sum_i \left(\sum_j K_{ij} \theta_j - E_i \right)^2, \tag{311}$$

this is implemented in Julia as:

```
"""
computes sum of residuals: sum_i(sum_j K_{ij}\theta_j - E_i)^2, given:
K, a real matrix of data
\theta, a real vector of coefficients
E, a real vector of target labels
"""
function dummy_fobj(K, \theta, E)
    res = 0.
    for i in eachindex(E)
        res += (K[i, :] * \theta - E[i])^2
    end
    return res
end
```

We use [ReverseDiff](#) to call reverse mode routine to compute the gradient:

```
"""
Reverse AD w.r.t \theta
"""
g!(\nabla f, \theta) = ReverseDiff.gradient!(\nabla f, \theta -> dummy_fobj(K, \theta, E), \theta)
```

Then declare the input data variables:

```
K = rand(100,100) # data matrix
E = rand(100) # target vector
init_θ = rand(100) # initial guess of coefficients
```

The objective function is minimized using BFGS implementation of `Optim.jl` [105]. We compare the optimization performance when using finite differences to compute the gradient (default mode if no gradient is supplied) against when the optimization routine is supplied by gradient from `ReverseDiff`, the results are in Figure 44. Both finite difference and reverse AD give minimum objective function value of 0. The computation time of reverse AD is faster by a factor of ~ 34 compared to finite difference. Additionally, it turns out that reverse AD is also more efficient in terms of memory allocation by also a factor of ~ 34 .

The illustrated `ReverseDiff` usage is in fact the simplest way of reverse AD usage. It is possible to optimize the efficiency of `ReverseDiff` usage, for example, by pre-compiling the gradient tapes once by supplying the type and shape of the input and output so that the tapes do not need to be re-compiled each time the gradient is computed; then surely reverse AD would be even more efficient.

```
julia> @btime optimize(θ->dummy_fobj(K,θ,E), init_θ, BFGS())
1.555 s (4369286 allocations: 3.56 GiB)
* Status: success

* Candidate solution
  Final objective value: 1.639190e-27
```

(a) Optimization performance with finite difference.

```
julia> @btime optimize(θ->dummy_fobj(K,θ,E), g!, init_θ, BFGS())
46.271 ms (529368 allocations: 106.56 MiB)
* Status: success

* Candidate solution
  Final objective value: 7.848709e-24
```

(b) Optimization performance with reverse AD.

Figure 44: Performance comparison of optimization between finite difference and reverse AD for computing the gradient.

Conclusion

We started from the development of pair-potential functions in the form of rational potential (RATPOT) variants to predict high accuracy diatomic H_xO_y data in Chapter 4. Benchmark was carried out between RATPOT variants against CHIPR. For most of the diatomic datasets, RATPOT reached median chemical accuracy of 1 kcal/mol, coupled with training efficiency in terms of computational time and number of parameters needed to be optimized in some benchmark samples. Additionally, a linear version of the RATPOT was also devised, which is more efficient for training since it is not reliant to non-linear optimization algorithm to be trained.

From pair-potential functions, we proceeded to Hydrogen clusters in Chapter 5. The proposed ansatz contains bonding potential functions that mimic the bonding strength of atoms with additional degrees of freedom for optimization flexibility. The bonding features were designed such that it include basis functions where each basis function captures different characteristics of atom and its neighbourhood. H_3 , H_4 , and H_5 molecules were used for the benchmark, where chemical accuracy was reached for H_4 data. The bonding potential functions were implemented in a way that they are compatible with reverse mode automatic differentiation; this resulted in more efficient training and optimization compared to the usage of finite differences during the computation of gradients.

We investigated machine learning approaches for large dataset containing larger equilibrium geometries (QM9 dataset) in Chapter 6, where only 100 out of 130k molecules with known energies are permissible to be used for training (the QM9 challenge). Semi-supervised method based on Shepard interpolation, named ROSEMI, was proposed. ROSEMI was tailored for equilibrium geometries where the information of target energies were minimum. One of the main advantage for training is that ROSEMI could make use data points without targets to gauge the interpolation deviation. While ROSEMI was designed for equilibrium geometries, nevertheless it was tested with diatomic datasets of H_xO_y , in which it reached chemical accuracy for some datasets. For the benchmark on QM9 dataset with only 100 molecules for training, chemical accuracy was not yet reached; improvements would have been needed to reach chemical accuracy, such as designing a way to allow for atomic level of features in contrast to molecular level features; since for energies, as it turned out atomic features performed better.

We continued our attempts to solve the QM9 challenge in Chapter 7. Here we focused on applying Δ ML procedures to reduce the magnitude of the energies such that the fitting processes became easier. Unsupervised and supervised data selection techniques were employed to select the most relevant 100 molecule for training that results in the lowest mean absolute error. In particular for unsupervised data selection we used the `usequence` which is a modified farthest point sampling algorithm, while for supervised selection we rely on an adaptation of Tabu search. We also applied intensive hyperparameter tuning wrapped in a mixed integer black-box optimization problem, combining feature extraction and selection techniques, various models including ROSEMI, and the models' hyperparameters. Overall, currently our combination of methods led to 3.25 kcal/mol accuracy, which is close to the chemical accuracy target for the solving QM9 challenge; and additionally with lower resolution parameter tuning, chemical accuracy was achieved with 3750 training molecules for the QM9 dataset.

Appendices

A General implementation information

The source code for Chapters 2, 6, and 7 is available in <https://github.com/berylgithub/qm>. The source code for Chapters 3, 4, and 5 is available in <https://github.com/berylgithub/pes>.

Most of the code are implemented in `Julia`, `Python`, and some `Octave` (with Matlab compatible syntax). Performance critical code are written in `Julia`, (data processing, model evaluation, and optimization), with the exception of most feature extraction routines which are written in `Python` (currently most of the molecular feature extraction libraries are written in `Python`). `Octave` is used to write the interface for hyperparameter optimization (Section 2.3) that bridges the simulations in `Julia` with the solver in Matlab.

Two sets of hardware are used for the main computations:

- AMD Ryzen 9 5900X 12-Core Processor with 32GB of RAM. Mainly used for the numerical experiments of Chapters 3, 4, and 5.
- Single node of VSC-5 (without GPU utilization). Mainly used for processing and numerical experiments of larger datasets involving Chapters 2, 6, and 7.

B Semi supervised model (Chapter 6)

B.1 Program structure for experiments in Section 6.4

The main program required for this section is divided into two main subprograms: feature extraction, data setup and fitter. The feature extraction and data setup subprogram is compatible with any types of model, while the fitter subprogram is specific for the Shepard model. Each subprogram is defined by:

- Feature extraction refers to any operations included in extracting the raw array of features (usually on the atomic level) given the geometrical information of the molecules (e.g., coordinates, nuclear charges, etc), this is usually only done once for each set of molecular geometries.
- Data setup includes PCA and data selection. In general, processes that are independent of θ and the indexing from set T are included into the data setup.
- The fitter subprogram computes everything which depend on θ or set T .

B.1.1 Feature extraction

Most of the feature extraction techniques for molecular geometries require similar set of inputs, e.g., the coordinates and nuclear charges; additionally with the list of hyperparameters. In most cases, each feature extraction technique is bundled within different libraries (and often on different programming languages); and usually each library wraps the molecular information in a different way, hence it is required to transform the raw molecular data into the the data structure demanded by the library. The abstraction of the feature extraction subprogram is

1. `mol_class` $\leftarrow$ `transform_data( $x, Z, \text{mol\_info}$ )`, where x is a set of molecular coordinates, Z is the set of nuclear charges and `mol_info` is a list of additional molecular information (which is usually not mandatory), such as the molecule name, number of atoms, etc.
2. $f \leftarrow \text{extract_feature}(\text{mol_class}, \text{params})$, where `mol_class` is an object that contains all of the required molecular information for feature extraction, and `params` is a list of the required hyperparameters, e.g., ACSF requires the cutoff radius, level of symmetry, periodicity of the atomic system, etc.
3. `save(feature\_file, f)`, where `feature\_file` is a string describing the harddrive path to store the array f , this will be used for the `data\_setup` process.

B.1.2 Data setup subprogram

The data setup function abstraction is

$$(F, T, D, \phi, \phi') \leftarrow \text{data_setup}(\text{indices}, n_{af}, n_{mf}, n_s, N, \text{feature_file}, \text{feature_mode}),$$

where `indices` is the set of labels which will be included, in case of full QM9 dataset then `indices` = $1 : N_{\text{QM9}}$; `feature\_file` is a string which indicates the file path of a file containing the atomic feature set computed on all of the QM9 data points; `feature\_mode` is a list of boolean which indicates the inclusion of sums of squares or binomial features, e.g., `feature\_mode` = `[true, false]` means only the sums of squares will be added to the molecular features. The flow of the data setup subprogram is:

1. $F \leftarrow \text{PCA_atom}(\text{feature_file}, n_{af})$
2. $F \leftarrow \text{extract_mol_feature}(F, \text{feature_mode})$
3. $F \leftarrow \text{PCA_mol}(F, n_{mf})$
4. $(\phi, \phi') \leftarrow \text{extract_bspline}(F, n_s)$
5. $(T, D) \leftarrow \text{get_centers}(F, N)$

B.1.3 Fitter subprogram

Given the set of files from section (B.1.2), the fitter function abstraction is simply

$$(\theta, \text{info}) \leftarrow \text{fitter}(\text{data_dir}, \text{t_limit}, \text{batchsize}) \quad (312)$$

where `data\_dir` is a string that refers to the path which contains the files resulted from the data setup subprogram, `t\_limit` is the solver time limit (in second), `batchsize` is an integer representing the size of mini-batch of several variables, in some cases the total memory usage of the system is over 8GB (typical Random Access Memory (RAM) size of modern computers) hence mini-batching is necessary. On the output side, `info` contains the fitting errors, list of timings, and list of fitting hyperparameters (e.g., n_{af}, n_{mf} , etc). The processes executed in the fitter are:

1. Compute variables for fitting:

- (a) $K \leftarrow \text{select_centers}(T)$
- (b) $U \leftarrow T \setminus K$
- (c) $W \leftarrow \{1, 2, \dots, n_{\text{data}}\} \setminus K$
- (d) $S_K \leftarrow \text{compute_SK}(D, K, U)$
- (e) $\gamma \leftarrow \text{compute_gamma}(D, S_K, K, U)$
- (f) $\alpha \leftarrow \gamma. - 1$
- (g) $\phi \leftarrow \text{compute_phi}(\Phi, \Phi', F, K, U)$

2. Fitting:

- (a) $b \leftarrow \text{compute_b}(E, \gamma, \alpha, K)$, where E is a vector of energies.
- (b) $\text{op} \leftarrow \text{LinearOperators}(y, Au, A^\top v)$
- (c) $\theta \leftarrow \text{CGLS}(\text{op}, b, \text{itmax}, \text{t_limit})$

3. Energy prediction is computed in mini-batches in terms of the row numbers, i.e., for any vectors or matrices, the maximum row size is `batchsize`; e.g., S_K vector for prediction has N_{QM9} row size by default, however when computed in mini-batches, S_K has `batchsize` rows for each mini-batch loop, where the total number of loops is `ceil( $N_{\text{QM9}}$ /batchsize)`.

- (a) $S_K \leftarrow \text{compute_SK}(D, K, W)$
- (b) $\phi \leftarrow \text{compute_phi}(\Phi, \Phi', F, K, W)$
- (c) $V_K \leftarrow \text{predict}(\theta, E, D, \phi, S_K, K, W)$

4. Compute the errors:

- (a) $\text{MAE} \leftarrow \text{get_MAE}(V_K, E)$
- (b) $\text{MAD} \leftarrow \text{get_MAD}(V_K, E, D, \theta, \phi, S_K, K, U)$, where MAD is a vector of length n_U . Note that the variables used to compute the MAD are the ones from step 1 (for fitting), not the ones from step 3.

B.2 Additional results for experiments in Section 6.3

Table 27: Hyperparameters of ROSEMI found after optimization for pair potential data.

Dataset	objective value	computation time (s)	force	n	c_s	p_{tr}
OH ¹	0.003	787.160	false	1	6	0.6
OH ²	0.004	802.194	true	1	6	0.5
OH ³	0.004	660.965	false	1	8	0.5
OH ⁴	0.021	598.250	false	3	1	0.3
OH ⁵	0.015	591.931	false	2	4	0.5
O ₂ ¹	0.002	700.700	false	2	8	0.5
O ₂ ²	0.002	665.585	false	1	10	0.5
O ₂ ³	0.002	669.035	false	2	6	0.5
H ₂ ¹	0.003	1974.593	false	1	5	0.6
OH+ ¹	0.009	287.298	true	1	7	0.4
H ₂ + ¹	0.049	751.684	false	1	5	0.6
OH+ ²	0.005	835.773	true	2	5	0.5
O ₂ + ¹	0.004	1117.320	true	1	6	0.7
OH ⁶	0.001	444.995	false	3	3	0.5

B.3 Additional results for experiments in Section B.1

Table 28: The feature sets which are used to generate the sets of training labels

k	f	n_{af}	n_{mf}	Mean train MAE	Std train MAE	Mean combined MAE	Std combined MAE
1-5	SOAP	20	10	18.364	1.458	20.365	0.296
6-10	ACSF	40	37	19.417	1.655	20.521	0.420
11-15	ACSF	40	30	20.914	0.860	20.224	0.193
16-20	ACSF	40	20	19.756	1.137	20.115	0.205
21-25	ACSF	30	25	20.314	0.315	20.369	0.346
26-30	ACSF	30	29	18.410	1.170	20.338	0.215
31-35	ACSF	30	20	19.391	2.209	20.295	0.196
36-40	ACSF	20	16	18.885	1.800	20.224	0.155
41-45	ACSF	20	16	19.565	2.107	20.134	0.142
46-50	ACSF	20	15	19.012	1.886	20.388	0.468
51-55	SOAP	75	72	20.114	1.521	20.156	0.206
56-60	SOAP	75	50	19.044	1.361	20.479	0.086
61-65	SOAP	75	25	19.351	0.601	20.319	0.178
66-70	SOAP	50	48	19.760	1.513	20.560	0.351
71-75	SOAP	50	45	19.955	1.973	20.178	0.153
76-80	SOAP	50	25	19.327	1.307	20.378	0.419
81-85	SOAP	25	22	18.933	2.011	20.442	0.468
86-90	SOAP	25	23	20.110	0.595	20.215	0.250
91-95	SOAP	25	22	17.677	1.157	20.222	0.224

Table 29: The molecular labels from 10 sets which result in the lowest atomization training MAE.

$k = 38$	$k = 95$	$k = 31$	$k = 48$	$k = 83$	$k = 8$	$k = 1$	$k = 45$	$k = 26$	$k = 91$
95178	80056	8298	58768	80368	61182	57345	58000	109079	71281
29069	101483	23686	23821	21951	133215	23785	99841	109455	47253
130049	81831	85743	101692	128149	121095	97934	118703	24927	123121
100842	123043	26524	124027	121875	75876	66699	825	131233	133311
132549	124471	124024	133355	96178	126818	5648	130229	23772	127756
52422	133437	82915	37794	37940	24457	120963	72552	133653	635
99118	35013	122995	128305	99977	29717	77829	56137	122533	83165
125747	14633	130107	126263	40	116729	130734	33311	23910	25312
132240	21166	35099	263	2603	62472	1884	7125	126564	128742
122757	41	97474	10199	122517	21575	130219	62183	66334	100178
4369	57362	133402	23902	129880	21868	32471	124477	117715	26643
9349	122592	151	38450	117718	14984	25270	113900	41287	101578
63712	95105	97808	10067	51902	83254	187	165	3933	125428
47245	122161	36755	42080	56600	84194	122852	19603	122726	21812
695	118440	72556	53069	132211	121448	35457	117520	71475	60127
67489	94425	123178	62308	105982	1750	49178	20452	23610	132141
27535	31552	20217	125168	7125	17578	129002	28073	133073	71152
78263	169	55208	23757	77823	39927	106017	23368	133295	132243
34115	48146	101558	111105	47200	57611	80898	70514	133447	96178
33741	74619	99978	394	34122	97455	122285	45109	35140	2680
21100	6028	16237	1532	119847	123396	126452	456	125	133383
14741	44216	126768	21339	78955	58014	31023	122106	25168	52428
43498	2823	131915	62119	574	3869	57087	21461	25316	34949
102520	56489	57964	119133	99845	132472	83255	102345	37550	128823
126043	2521	2480	100017	124068	121469	2875	52825	275	9391
1378	66798	130001	119281	26616	16016	26360	61016	116057	126606
46227	72807	55429	97942	38625	35398	72080	1200	32172	65
2687	110881	9683	22154	131459	127694	122658	4790	7397	482
14603	58151	108678	17640	58375	52948	3861	71547	92005	126771
684	86714	30946	101678	9282	14551	87118	104319	52428	117408
121974	2551	3950	38816	102061	46471	13948	77091	40775	100875
96711	23881	55400	3018	109682	104571	102186	2100	131558	36249
115203	86269	37221	56237	33524	37707	125445	122640	14653	63761
81070	129192	22013	53425	6841	113687	102294	103946	36734	42624
3457	71872	77624	52437	6385	1313	79101	97383	109567	85921
127186	133082	122947	34909	25312	97836	132648	29981	7829	55175
13887	116929	1925	17172	275	51557	44825	48117	100354	111021
11556	271	11252	105754	64011	130078	52736	55620	406	13664
80801	116831	81013	124785	31058	27963	131841	39026	67944	92343
97541	118415	56291	90368	133205	4273	103691	19902	21414	747
83224	29549	21583	33497	4200	124023	9888	125625	122222	119927
564	17533	118016	19194	121593	97576	13837	33279	56367	18782
26372	110750	73307	122829	97495	88982	36656	2272	41894	122007
82796	685	61427	68682	119623	115042	34611	28375	100204	93129
38533	115076	7391	78111	72175	32155	21272	4223	3110	102339
99376	120323	55675	132359	97522	111463	96554	2080	33583	21735
59370	472	28815	129902	105899	113693	132244	120975	3719	124537
42900	132470	122620	43009	102498	19731	9218	99313	38606	66719
27403	47997	51838	6751	30692	100597	79340	36748	132264	74072
1414	89327	119895	102394	123066	99859	132822	117331	74674	133291
59758	121867	77590	10341	43055	28773	133424	114722	99567	133126
102208	13378	88837	71652	115670	1012	62528	114632	59433	60279
132885	16767	38733	88589	494	4836	20793	20973	9396	14752
130619	95590	80119	27570	19502	30346	99945	127056	30996	79556
53704	125540	112780	20771	57610	48036	9413	110720	2731	71795
17737	25648	81746	1495	52722	27836	129054	50831	22420	21141
108667	71359	93973	112122	1306	23656	2937	99177	46804	19845
23728	75117	68054	116361	95408	6781	22269	33499	15185	124485
2291	122487	130970	97920	125416	6216	101020	11970	80506	6994
117900	24025	133273	34928	133728	130104	82731	98907	84877	131775
41240	60385	6111	88775	122627	90260	76238	133502	132308	24246
36811	78433	46351	42005	9938	2166	11332	132034	8779	120611
81262	118463	126933	35233	37205	122255	34709	47813	50576	268
24519	127219	90675	117331	79300	42903	43904	104648	53406	1749
18280	39337	131905	132011	1395	3410	132485	102942	129731	93575
33676	37547	83702	101673	76300	55522	49734	38330	14241	113749
98982	117317	20702	2283	117006	126787	23270	34571	98780	65758
111511	3526	12646	126400	116162	33800	23669	81376	128868	10001
3125	98136	131763	121358	14541	117966	40587	634	133634	106993
73149	22027	122810	133031	106155	61436	59890	25692	42586	23913
116856	52530	543	11763	133120	950	84789	120559	101568	2829
55638	97350	28725	8562	112913	53798	124648	102241	129119	71359
66568	9996	9846	48152	22055	43516	52773	68491	38450	118900
21924	1457	76210	83481	100720	49583	123199	80381	13881	57378
62552	75928	117093	5569	85724	32832	56071	132974	76803	271
46082	62614	35250	132736	131158	25401	56062	21553	99613	119588
79786	129082	131421	123165	81499	108416	127432	11338	28194	95309
48159	99640	1659	86981	33010	37104	95604	3265	18056	125746
128542	84595	132370	131028	81410	49447	9586	58200	21228	15254
103920	55677	82806	34074	6705	114718	102941	1386	107274	108344
28261	9376	109642	114831	72427	64648	58974	40982	35682	6082
36824	117282	394	101598	2456	100528	112380	56044	86685	16147
6915	35915	3631	7292	129123	13347	132280	172	29787	8791
2531	116988	34991	61846	32825	84009	58967	121005	33339	48614
34923	104097	125555	85643	47989	77329	17327	93121	120600	49135
16548	39341	52686	132797	99893	36135	128642	19116	34152	118201
2565	133594	22298	55544	97145	23616	100463	79129	132457	124022
129844	48443	83213	2229	12868	54807	31602	116804	102844	55520
16224	36205	120498	55509	21374	131344	125132	32570	115507	122317
122072	27867	4623	43084	36806	47953	42437	61052	22177	69241
13257	130131	35381	71634	6911	39550	19062	96597	7992	133228
57820	130888	30660	133215	66210	126232	56270	24860	86077	90165
129689	126676	127832	100822	56322	83838	39555	68343	79904	12473
100240	17814	35726	68021	10226	31505	96278	42869	20523	99299
33383	45233	1855	8950	57618	58012	61267	57985	124953	126770
35465	111766	57380	128220	91203	126508	19987	92259	97829	116488
30805	131683	78097	126376	108938	61063	39548	112311	29744	29911
15817	68890	42312	101342	83843	123780	35694	116649	20752	52938
46276	104265	16832	22215	116692	120972	93410	54335	98280	84026
105833	36573	93851	13489	75434	1294516	117036	118309	70773	6739

C Δ ML (Chapter 7)

C.1 Additional results involving baseline features

Table 30: 60 baseline MAEs of the training indices, sorted by the last level of baseline MAE (DB train).

DA train	DA test	DB train	DB test
15.307	20.292	9.328	14.787
15.143	21.467	9.808	12.962
17.799	20.486	10.150	13.831
19.793	20.391	10.492	14.148
18.659	20.334	10.552	13.348
19.196	20.529	10.891	13.727
16.706	20.782	10.902	13.185
17.366	20.580	11.077	14.150
19.899	20.147	11.204	13.919
21.531	20.505	11.214	13.740
19.135	20.517	11.369	14.186
15.847	20.097	11.502	13.530
18.472	20.966	11.558	13.857
20.193	20.245	11.576	12.877
19.645	20.441	11.775	13.517
20.176	20.300	11.861	13.129
19.957	20.486	11.868	15.416
19.012	20.038	11.881	13.128
22.112	20.262	11.892	14.015
18.476	20.495	11.941	14.230
19.592	20.126	11.944	13.967
18.576	19.989	11.965	13.833
17.789	20.397	11.992	14.042
19.951	20.284	12.002	14.548
19.969	20.102	12.022	13.328
19.743	20.587	12.078	13.621
16.730	21.060	12.092	13.679
19.500	20.509	12.261	13.360
22.261	20.372	12.386	13.906
20.070	19.953	12.389	13.398
18.897	22.254	12.563	14.415
20.848	20.138	12.630	13.141
17.621	21.746	12.635	13.996
22.064	20.336	12.649	13.026
22.343	20.078	12.857	13.353
19.850	19.992	12.871	14.030
19.833	20.124	12.883	14.007
18.924	20.636	12.933	13.328
20.960	21.281	13.001	13.218
17.950	20.461	13.046	14.021
18.192	20.397	13.053	14.576
17.719	20.203	13.058	12.794
22.869	20.391	13.083	13.352
21.180	20.762	13.098	13.768
19.857	20.298	13.153	13.822
20.349	20.193	13.154	13.152
19.147	1477.395	13.164	1467.483
23.539	20.711	13.208	12.599
19.721	20.351	13.221	13.211
22.707	20.146	13.222	12.996
18.652	20.178	13.250	13.060
17.843	21.095	13.289	14.080
21.611	20.272	13.409	13.680
19.322	20.194	13.812	13.415
20.217	21.630	13.992	15.483
20.595	20.171	14.110	13.772
20.745	21.152	14.356	13.328
23.850	20.724	14.605	14.669
22.626	20.165	14.694	13.225
20.922	20.103	15.269	13.594

Table 31: Enumerated runs up to DB level of baseline with random data selection.

ntrain	feature	model	solver	elevel	MAEtrain	MAEtest
10	ACSF	GK	direct	DA	6.233e-13	32.573
25	ACSF	GK	direct	DA	3.885e-11	35.732
50	ACSF	GK	direct	DA	1.891e-10	42.436
100	ACSF	GK	direct	DA	1.894e-09	16.345
10	ACSF	GK	cgls	DA	2.982e-08	32.573
25	ACSF	GK	cgls	DA	5.043e-05	35.732
50	ACSF	GK	cgls	DA	0.646	42.260
100	ACSF	GK	cgls	DA	4.986	14.716
10	ACSF	DPK	direct	DA	4.975e-11	52.021
25	ACSF	DPK	direct	DA	3.379e-09	40.372
50	ACSF	DPK	direct	DA	2.039e-07	936.750
100	ACSF	DPK	direct	DA	9.420e-05	19.089
10	ACSF	DPK	cgls	DA	8.002e-07	52.021
25	ACSF	DPK	cgls	DA	0.631	54.071
50	ACSF	DPK	cgls	DA	4.344	31.303
100	ACSF	DPK	cgls	DA	6.505	12.688
10	ACSF	GK	direct	DB	3.174e-13	24.066
25	ACSF	GK	direct	DB	2.007e-11	44.785
50	ACSF	GK	direct	DB	9.592e-11	39.087
100	ACSF	GK	direct	DB	1.437e-09	12.513
10	ACSF	GK	cgls	DB	1.873e-08	24.066
25	ACSF	GK	cgls	DB	6.266e-05	44.785
50	ACSF	GK	cgls	DB	0.625	36.941
100	ACSF	GK	cgls	DB	4.198	12.537
10	ACSF	DPK	direct	DB	3.226e-11	28.044
25	ACSF	DPK	direct	DB	6.328e-09	34.037
50	ACSF	DPK	direct	DB	6.754e-08	184.163
100	ACSF	DPK	direct	DB	8.941e-05	14.418
10	ACSF	DPK	cgls	DB	1.207e-08	28.044
25	ACSF	DPK	cgls	DB	0.006	34.039
50	ACSF	DPK	cgls	DB	2.288	13.176
100	ACSF	DPK	cgls	DB	4.370	8.520
10	SOAP	GK	direct	DA	0.000	2.040e+06
25	SOAP	GK	direct	DA	0.000	3.267e+06
50	SOAP	GK	direct	DA	0.000	2.335e+06
100	SOAP	GK	direct	DA	0.000	11.808
10	SOAP	GK	cgls	DA	15.320	27.531
25	SOAP	GK	cgls	DA	20.921	21.391
50	SOAP	GK	cgls	DA	18.710	21.273
100	SOAP	GK	cgls	DA	18.019	20.264
10	SOAP	DPK	direct	DA	4.065e-11	38.866
25	SOAP	DPK	direct	DA	1.668e-10	32.059
50	SOAP	DPK	direct	DA	3.823e-10	43.585
100	SOAP	DPK	direct	DA	4.016e-09	9.344
10	SOAP	DPK	cgls	DA	9.149	26.766
25	SOAP	DPK	cgls	DA	15.478	20.542
50	SOAP	DPK	cgls	DA	15.777	18.440
100	SOAP	DPK	cgls	DA	13.057	16.391
10	SOAP	GK	direct	DB	2.688e-05	2.175e+06
25	SOAP	GK	direct	DB	7.436e-05	5.440e+06
50	SOAP	GK	direct	DB	0.000	1.074e+07
100	SOAP	GK	direct	DB	0.000	10.019
10	SOAP	GK	cgls	DB	9.191	17.974
25	SOAP	GK	cgls	DB	12.494	13.745
50	SOAP	GK	cgls	DB	10.583	13.967
100	SOAP	GK	cgls	DB	10.478	13.352
10	SOAP	DPK	direct	DB	1.484e-11	33.819
25	SOAP	DPK	direct	DB	1.498e-10	19.106
50	SOAP	DPK	direct	DB	2.100e-10	14.343
100	SOAP	DPK	direct	DB	1.973e-09	8.843
10	SOAP	DPK	cgls	DB	7.104	19.569
25	SOAP	DPK	cgls	DB	9.896	13.676
50	SOAP	DPK	cgls	DB	9.907	13.329
100	SOAP	DPK	cgls	DB	9.456	12.552
10	FCHL19	GK	direct	DA	4.983e-10	55.431
25	FCHL19	GK	direct	DA	2.674e-09	62.909
50	FCHL19	GK	direct	DA	1.576e-08	1353.509
100	FCHL19	GK	direct	DA	3.721e-08	8.035
10	FCHL19	GK	cgls	DA	0.003	55.408
25	FCHL19	GK	cgls	DA	0.474	63.822
50	FCHL19	GK	cgls	DA	1.914	406.800
100	FCHL19	GK	cgls	DA	2.962	8.708
10	FCHL19	DPK	direct	DA	1.714e-12	31.372
25	FCHL19	DPK	direct	DA	1.266e-11	35.113
50	FCHL19	DPK	direct	DA	2.535e-11	15.405
100	FCHL19	DPK	direct	DA	8.096e-11	8.601
10	FCHL19	DPK	cgls	DA	5.808e-09	31.372
25	FCHL19	DPK	cgls	DA	2.962e-05	35.113
50	FCHL19	DPK	cgls	DA	0.003	15.408
100	FCHL19	DPK	cgls	DA	1.113	8.336
10	FCHL19	GK	direct	DB	7.477e-10	44.018
25	FCHL19	GK	direct	DB	2.567e-09	56.209
50	FCHL19	GK	direct	DB	3.226e-08	945.465
100	FCHL19	GK	direct	DB	4.762e-08	8.878
10	FCHL19	GK	cgls	DB	0.003	43.995
25	FCHL19	GK	cgls	DB	0.152	50.503
50	FCHL19	GK	cgls	DB	1.199	621.318
100	FCHL19	GK	cgls	DB	2.767	8.934
10	FCHL19	DPK	direct	DB	3.342e-12	30.943
25	FCHL19	DPK	direct	DB	6.780e-12	21.137
50	FCHL19	DPK	direct	DB	2.714e-11	12.618
100	FCHL19	DPK	direct	DB	6.911e-11	9.770
10	FCHL19	DPK	cgls	DB	8.021e-09	30.943
25	FCHL19	DPK	cgls	DB	1.325e-05	21.137
50	FCHL19	DPK	cgls	DB	0.001	12.617
100	FCHL19	DPK	cgls	DB	1.139	9.555

Table 32: Enumerated runs of [DN,DT] baselines with random data selection.

ntrain	feature	model	solver	elevel	MAEtrain	MAEtest
10	ACSF	GK	direct	DN	8.340e-13	23.963
25	ACSF	GK	direct	DN	1.172e-11	44.535
50	ACSF	GK	direct	DN	1.034e-10	39.067
100	ACSF	GK	direct	DN	1.564e-09	12.519
10	ACSF	GK	cgl	DN	2.119e-08	23.963
25	ACSF	GK	cgl	DN	9.994e-06	44.535
50	ACSF	GK	cgl	DN	0.615	37.006
100	ACSF	GK	cgl	DN	4.188	12.518
10	ACSF	DPK	direct	DN	7.627e-11	27.917
25	ACSF	DPK	direct	DN	4.708e-09	33.747
50	ACSF	DPK	direct	DN	5.080e-08	190.144
100	ACSF	DPK	direct	DN	5.371e-05	14.392
10	ACSF	DPK	cgl	DN	7.221e-07	27.917
25	ACSF	DPK	cgl	DN	0.001	33.746
50	ACSF	DPK	cgl	DN	2.287	15.782
100	ACSF	DPK	cgl	DN	4.472	8.874
10	SOAP	GK	direct	DN	2.391e-05	2.171e+06
25	SOAP	GK	direct	DN	0.000	5.275e+06
50	SOAP	GK	direct	DN	0.000	1.055e+07
100	SOAP	GK	direct	DN	0.000	9.966
10	SOAP	GK	cgl	DN	9.227	17.810
25	SOAP	GK	cgl	DN	12.465	13.652
50	SOAP	GK	cgl	DN	10.545	13.891
100	SOAP	GK	cgl	DN	10.426	13.286
10	SOAP	DPK	direct	DN	2.149e-11	33.645
25	SOAP	DPK	direct	DN	6.369e-11	19.320
50	SOAP	DPK	direct	DN	6.392e-10	14.053
100	SOAP	DPK	direct	DN	2.100e-09	8.791
10	SOAP	DPK	cgl	DN	7.107	19.430
25	SOAP	DPK	cgl	DN	9.871	13.589
50	SOAP	DPK	cgl	DN	9.877	13.283
100	SOAP	DPK	cgl	DN	9.384	12.471
10	FCHL19	GK	direct	DN	9.307e-10	43.876
25	FCHL19	GK	direct	DN	1.185e-09	57.109
50	FCHL19	GK	direct	DN	3.032e-08	949.965
100	FCHL19	GK	direct	DN	3.357e-08	8.832
10	FCHL19	GK	cgl	DN	0.003	43.853
25	FCHL19	GK	cgl	DN	0.178	53.869
50	FCHL19	GK	cgl	DN	1.195	619.905
100	FCHL19	GK	cgl	DN	2.753	8.804
10	FCHL19	DPK	direct	DN	2.399e-12	30.819
25	FCHL19	DPK	direct	DN	9.766e-12	21.222
50	FCHL19	DPK	direct	DN	2.305e-11	12.652
100	FCHL19	DPK	direct	DN	7.208e-11	9.744
10	FCHL19	DPK	cgl	DN	6.292e-09	30.819
25	FCHL19	DPK	cgl	DN	4.217e-06	21.222
50	FCHL19	DPK	cgl	DN	0.001	12.652
100	FCHL19	DPK	cgl	DN	1.152	9.511
10	ACSF	GK	direct	DT	6.423e-13	27.167
25	ACSF	GK	direct	DT	3.755e-12	25.068
50	ACSF	GK	direct	DT	9.869e-11	41.067
100	ACSF	GK	direct	DT	1.354e-09	18.331
10	ACSF	GK	cgl	DT	1.494e-08	27.167
25	ACSF	GK	cgl	DT	1.451e-05	25.068
50	ACSF	GK	cgl	DT	0.304	41.765
100	ACSF	GK	cgl	DT	3.083	17.658
10	ACSF	DPK	direct	DT	8.861e-12	26.545
25	ACSF	DPK	direct	DT	4.553e-10	76.725
50	ACSF	DPK	direct	DT	4.237e-08	540.173
100	ACSF	DPK	direct	DT	4.798e-05	25.196
10	ACSF	DPK	cgl	DT	2.667e-06	26.545
25	ACSF	DPK	cgl	DT	0.002	76.723
50	ACSF	DPK	cgl	DT	2.664	81.268
100	ACSF	DPK	cgl	DT	4.414	15.925
10	SOAP	GK	direct	DT	6.880e-05	1.428e+06
25	SOAP	GK	direct	DT	0.000	1.130e+06
50	SOAP	GK	direct	DT	4.330e-05	1.623e+06
100	SOAP	GK	direct	DT	0.000	17.401
10	SOAP	GK	cgl	DT	6.601	23.740
25	SOAP	GK	cgl	DT	6.772	18.053
50	SOAP	GK	cgl	DT	5.982	18.111
100	SOAP	GK	cgl	DT	5.867	18.593
10	SOAP	DPK	direct	DT	2.177e-11	27.434
25	SOAP	DPK	direct	DT	1.432e-10	22.640
50	SOAP	DPK	direct	DT	8.332e-10	50.345
100	SOAP	DPK	direct	DT	2.602e-09	16.500
10	SOAP	DPK	cgl	DT	4.852	24.833
25	SOAP	DPK	cgl	DT	5.712	18.257
50	SOAP	DPK	cgl	DT	5.241	18.165
100	SOAP	DPK	cgl	DT	5.541	18.365
10	FCHL19	GK	direct	DT	1.571e-10	37.883
25	FCHL19	GK	direct	DT	1.700e-09	46.964
50	FCHL19	GK	direct	DT	3.351e-08	355.542
100	FCHL19	GK	direct	DT	5.479e-08	16.452
10	FCHL19	GK	cgl	DT	0.008	37.821
25	FCHL19	GK	cgl	DT	0.103	48.474
50	FCHL19	GK	cgl	DT	1.337	411.182
100	FCHL19	GK	cgl	DT	2.945	16.751
10	FCHL19	DPK	direct	DT	1.671e-12	28.380
25	FCHL19	DPK	direct	DT	5.537e-12	18.713
50	FCHL19	DPK	direct	DT	4.449e-12	20.701
100	FCHL19	DPK	direct	DT	1.260e-10	16.823
10	FCHL19	DPK	cgl	DT	3.837e-09	28.380
25	FCHL19	DPK	cgl	DT	2.419e-05	18.713
50	FCHL19	DPK	cgl	DT	0.000	20.701
100	FCHL19	DPK	cgl	DT	1.766	16.786

Table 33: Enumerated runs of the 2nd ranked training set from `usequence` up to DB.

ntrain	feature	model	solver	elevel	MAEtrain	MAEtest
10	ACSF	GK	direct	DA	1.308e-12	32.179
25	ACSF	GK	direct	DA	3.256e-12	25.735
50	ACSF	GK	direct	DA	5.601e-11	28.103
100	ACSF	GK	direct	DA	1.239e-09	17.360
10	ACSF	GK	cgl	DA	4.578e-09	32.179
25	ACSF	GK	cgl	DA	1.549e-06	25.735
50	ACSF	GK	cgl	DA	0.424	28.682
100	ACSF	GK	cgl	DA	4.892	14.214
10	ACSF	DPK	direct	DA	5.460e-11	23.553
25	ACSF	DPK	direct	DA	1.569e-09	54.891
50	ACSF	DPK	direct	DA	7.250e-08	18.925
100	ACSF	DPK	direct	DA	4.288e-05	11.931
10	ACSF	DPK	cgl	DA	5.664e-09	23.553
25	ACSF	DPK	cgl	DA	0.228	53.451
50	ACSF	DPK	cgl	DA	3.061	17.777
100	ACSF	DPK	cgl	DA	5.879	10.980
10	ACSF	GK	direct	DB	1.477e-13	21.822
25	ACSF	GK	direct	DB	4.539e-12	17.701
50	ACSF	GK	direct	DB	9.395e-11	21.133
100	ACSF	GK	direct	DB	9.922e-10	12.513
10	ACSF	GK	cgl	DB	3.005e-09	21.822
25	ACSF	GK	cgl	DB	3.997e-05	17.701
50	ACSF	GK	cgl	DB	0.202	21.029
100	ACSF	GK	cgl	DB	3.872	10.172
10	ACSF	DPK	direct	DB	2.529e-11	19.703
25	ACSF	DPK	direct	DB	1.301e-09	23.257
50	ACSF	DPK	direct	DB	1.764e-08	13.042
100	ACSF	DPK	direct	DB	3.548e-05	17.031
10	ACSF	DPK	cgl	DB	4.376e-07	19.703
25	ACSF	DPK	cgl	DB	0.389	21.560
50	ACSF	DPK	cgl	DB	2.839	10.846
100	ACSF	DPK	cgl	DB	4.432	7.566
10	SOAP	GK	direct	DA	0.000	3.619e+06
25	SOAP	GK	direct	DA	0.001	4.984e+06
50	SOAP	GK	direct	DA	0.000	17.143
100	SOAP	GK	direct	DA	0.000	13.268
10	SOAP	GK	cgl	DA	9.153	22.505
25	SOAP	GK	cgl	DA	16.147	21.340
50	SOAP	GK	cgl	DA	15.000	21.082
100	SOAP	GK	cgl	DA	15.143	21.466
10	SOAP	DPK	direct	DA	1.272e-11	32.276
25	SOAP	DPK	direct	DA	3.394e-10	26.876
50	SOAP	DPK	direct	DA	3.825e-10	12.742
100	SOAP	DPK	direct	DA	2.697e-09	9.877
10	SOAP	DPK	cgl	DA	8.139	22.282
25	SOAP	DPK	cgl	DA	10.522	18.359
50	SOAP	DPK	cgl	DA	11.034	18.793
100	SOAP	DPK	cgl	DA	11.709	18.324
10	SOAP	GK	direct	DB	0.000	2.892e+06
25	SOAP	GK	direct	DB	0.000	1.265e+06
50	SOAP	GK	direct	DB	0.000	11.207
100	SOAP	GK	direct	DB	0.000	9.764
10	SOAP	GK	cgl	DB	9.539	14.491
25	SOAP	GK	cgl	DB	9.007	13.263
50	SOAP	GK	cgl	DB	8.997	12.933
100	SOAP	GK	cgl	DB	9.647	13.014
10	SOAP	DPK	direct	DB	1.200e-11	23.605
25	SOAP	DPK	direct	DB	1.112e-10	17.083
50	SOAP	DPK	direct	DB	1.959e-09	16.481
100	SOAP	DPK	direct	DB	3.052e-09	8.839
10	SOAP	DPK	cgl	DB	7.703	13.582
25	SOAP	DPK	cgl	DB	6.573	13.185
50	SOAP	DPK	cgl	DB	7.725	12.187
100	SOAP	DPK	cgl	DB	8.387	11.930
10	FCHL19	GK	direct	DA	2.315e-10	32.041
25	FCHL19	GK	direct	DA	4.008e-09	23.537
50	FCHL19	GK	direct	DA	1.185e-08	13.854
100	FCHL19	GK	direct	DA	5.206e-08	9.424
10	FCHL19	GK	cgl	DA	0.002	32.039
25	FCHL19	GK	cgl	DA	0.242	23.199
50	FCHL19	GK	cgl	DA	0.759	12.091
100	FCHL19	GK	cgl	DA	3.451	8.993
10	FCHL19	DPK	direct	DA	1.697e-12	23.323
25	FCHL19	DPK	direct	DA	2.224e-11	17.183
50	FCHL19	DPK	direct	DA	3.061e-11	13.293
100	FCHL19	DPK	direct	DA	1.393e-10	10.136
10	FCHL19	DPK	cgl	DA	1.946e-09	23.323
25	FCHL19	DPK	cgl	DA	2.751e-07	17.183
50	FCHL19	DPK	cgl	DA	0.001	13.293
100	FCHL19	DPK	cgl	DA	1.555	8.677
10	FCHL19	GK	direct	DB	1.034e-10	24.959
25	FCHL19	GK	direct	DB	3.580e-09	61.710
50	FCHL19	GK	direct	DB	1.731e-08	13.418
100	FCHL19	GK	direct	DB	6.413e-08	9.395
10	FCHL19	GK	cgl	DB	0.001	24.956
25	FCHL19	GK	cgl	DB	0.281	60.758
50	FCHL19	GK	cgl	DB	0.850	12.411
100	FCHL19	GK	cgl	DB	3.080	7.791
10	FCHL19	DPK	direct	DB	6.310e-13	19.192
25	FCHL19	DPK	direct	DB	1.874e-12	13.940
50	FCHL19	DPK	direct	DB	4.469e-12	13.179
100	FCHL19	DPK	direct	DB	1.629e-10	10.780
10	FCHL19	DPK	cgl	DB	2.004e-09	19.192
25	FCHL19	DPK	cgl	DB	6.208e-07	13.940
50	FCHL19	DPK	cgl	DB	0.003	13.182
100	FCHL19	DPK	cgl	DB	1.766	8.741

Table 34: Enumerated runs of the 2nd ranked training set from `usequence` on [DN,DT].

ntrain	feature	model	solver	elevel	MAEtrain	MAEtest
10	ACSF	GK	direct	DN	7.258e-13	21.599
25	ACSF	GK	direct	DN	3.916e-12	17.859
50	ACSF	GK	direct	DN	9.331e-11	21.039
100	ACSF	GK	direct	DN	1.100e-09	12.514
10	ACSF	GK	cgl	DN	1.260e-07	21.599
25	ACSF	GK	cgl	DN	1.301e-05	17.859
50	ACSF	GK	cgl	DN	0.153	20.933
100	ACSF	GK	cgl	DN	3.866	10.229
10	ACSF	DPK	direct	DN	1.665e-11	20.003
25	ACSF	DPK	direct	DN	1.538e-09	23.533
50	ACSF	DPK	direct	DN	4.584e-08	13.220
100	ACSF	DPK	direct	DN	3.911e-05	17.218
10	ACSF	DPK	cgl	DN	8.688e-08	20.003
25	ACSF	DPK	cgl	DN	0.336	22.386
50	ACSF	DPK	cgl	DN	2.871	10.849
100	ACSF	DPK	cgl	DN	4.461	7.673
10	SOAP	GK	direct	DN	0.000	2.932e+06
25	SOAP	GK	direct	DN	0.000	1.287e+06
50	SOAP	GK	direct	DN	0.000	11.237
100	SOAP	GK	direct	DN	0.000	9.812
10	SOAP	GK	cgl	DN	9.498	14.329
25	SOAP	GK	cgl	DN	8.817	13.275
50	SOAP	GK	cgl	DN	8.896	12.940
100	SOAP	GK	cgl	DN	9.589	13.036
10	SOAP	DPK	direct	DN	1.564e-11	23.933
25	SOAP	DPK	direct	DN	1.376e-10	17.346
50	SOAP	DPK	direct	DN	3.805e-10	16.618
100	SOAP	DPK	direct	DN	3.616e-09	8.876
10	SOAP	DPK	cgl	DN	7.793	13.565
25	SOAP	DPK	cgl	DN	6.602	13.321
50	SOAP	DPK	cgl	DN	7.869	12.292
100	SOAP	DPK	cgl	DN	8.457	12.013
10	FCHL19	GK	direct	DN	1.394e-10	25.124
25	FCHL19	GK	direct	DN	1.343e-09	62.136
50	FCHL19	GK	direct	DN	1.013e-08	13.489
100	FCHL19	GK	direct	DN	6.116e-08	9.432
10	FCHL19	GK	cgl	DN	0.002	25.121
25	FCHL19	GK	cgl	DN	0.279	61.197
50	FCHL19	GK	cgl	DN	0.859	12.483
100	FCHL19	GK	cgl	DN	3.107	7.838
10	FCHL19	DPK	direct	DN	4.365e-13	19.346
25	FCHL19	DPK	direct	DN	1.169e-12	13.985
50	FCHL19	DPK	direct	DN	4.976e-12	13.221
100	FCHL19	DPK	direct	DN	1.447e-10	10.792
10	FCHL19	DPK	cgl	DN	1.577e-09	19.346
25	FCHL19	DPK	cgl	DN	3.352e-06	13.985
50	FCHL19	DPK	cgl	DN	0.003	13.225
100	FCHL19	DPK	cgl	DN	1.796	8.679
10	ACSF	GK	direct	DT	5.440e-14	15.899
25	ACSF	GK	direct	DT	1.130e-12	24.969
50	ACSF	GK	direct	DT	3.074e-11	19.195
100	ACSF	GK	direct	DT	9.363e-10	16.236
10	ACSF	GK	cgl	DT	2.393e-07	15.899
25	ACSF	GK	cgl	DT	2.414e-05	24.969
50	ACSF	GK	cgl	DT	0.241	18.648
100	ACSF	GK	cgl	DT	3.112	14.448
10	ACSF	DPK	direct	DT	8.900e-11	28.081
25	ACSF	DPK	direct	DT	5.540e-10	36.378
50	ACSF	DPK	direct	DT	3.690e-08	26.385
100	ACSF	DPK	direct	DT	0.000	24.878
10	ACSF	DPK	cgl	DT	4.440e-08	28.081
25	ACSF	DPK	cgl	DT	0.156	33.778
50	ACSF	DPK	cgl	DT	2.770	14.026
100	ACSF	DPK	cgl	DT	4.453	12.998
10	SOAP	GK	direct	DT	0.000	1.775e+06
25	SOAP	GK	direct	DT	0.000	1.547e+06
50	SOAP	GK	direct	DT	0.000	15.503
100	SOAP	GK	direct	DT	0.000	14.365
10	SOAP	GK	cgl	DT	7.595	16.807
25	SOAP	GK	cgl	DT	7.877	16.639
50	SOAP	GK	cgl	DT	6.571	16.208
100	SOAP	GK	cgl	DT	6.765	16.112
10	SOAP	DPK	direct	DT	4.024e-11	23.575
25	SOAP	DPK	direct	DT	5.084e-10	20.785
50	SOAP	DPK	direct	DT	7.029e-10	16.389
100	SOAP	DPK	direct	DT	3.369e-09	13.474
10	SOAP	DPK	cgl	DT	6.276	15.732
25	SOAP	DPK	cgl	DT	6.494	15.467
50	SOAP	DPK	cgl	DT	5.724	15.740
100	SOAP	DPK	cgl	DT	6.164	15.421
10	FCHL19	GK	direct	DT	6.067e-10	23.145
25	FCHL19	GK	direct	DT	7.564e-09	80.695
50	FCHL19	GK	direct	DT	1.740e-08	17.098
100	FCHL19	GK	direct	DT	8.434e-08	14.547
10	FCHL19	GK	cgl	DT	0.001	23.142
25	FCHL19	GK	cgl	DT	0.142	79.207
50	FCHL19	GK	cgl	DT	0.745	16.214
100	FCHL19	GK	cgl	DT	3.327	13.327
10	FCHL19	DPK	direct	DT	5.392e-13	22.061
25	FCHL19	DPK	direct	DT	3.175e-12	17.924
50	FCHL19	DPK	direct	DT	6.245e-12	16.570
100	FCHL19	DPK	direct	DT	2.127e-10	15.092
10	FCHL19	DPK	cgl	DT	2.883e-09	22.061
25	FCHL19	DPK	cgl	DT	3.059e-05	17.924
50	FCHL19	DPK	cgl	DT	0.003	16.569
100	FCHL19	DPK	cgl	DT	2.043	13.435

Table 35: Experiments of baselines that include `ntrain` > 100.

ntrain	baseline	MAEtrain	MAEtest	MAEtot
50	b	10.758	1470.428	1469.870
100	b	12.177	14.500	14.498
1000	b	13.084	12.901	12.902
10000	b	13.140	12.908	12.926
100000	b	12.928	13.086	12.965
50	n	18.022	1475.407	1474.850
100	n	17.658	20.738	20.735
1000	n	19.503	19.835	19.833
10000	n	19.926	19.705	19.722
100000	n	19.648	19.753	19.673
50	t	4.980	1476.746	1476.183
100	t	6.387	53.087	53.051
1000	t	10.754	11.707	11.700
10000	t	10.729	10.754	10.752
100000	t	10.677	10.737	10.691
50	bn	10.438	1470.394	1469.836
100	bn	12.133	14.455	14.454
1000	bn	13.054	12.861	12.862
10000	bn	13.118	12.876	12.894
100000	bn	12.904	13.062	12.941
50	bt	5.109	1478.982	1478.419
100	bt	6.043	56.152	56.113
1000	bt	10.898	11.592	11.587
10000	bt	10.813	10.811	10.811
100000	bt	10.775	10.855	10.793
50	nt	4.974	1477.087	1476.525
100	nt	6.389	52.346	52.311
1000	nt	10.664	11.626	11.618
10000	nt	10.644	10.661	10.660
100000	nt	10.588	10.636	10.600
50	bnt	5.033	1479.100	1478.537
100	bnt	6.027	55.716	55.678
1000	bnt	10.896	11.588	11.582
10000	bnt	10.812	10.810	10.810
100000	bnt	10.773	10.853	10.792

C.2 Hyperparameter tuning detailed results (Section 7.3.7)

Table 36: The initial points and the points that corresponds to the minimal objective value found for each set of experiments for the hyperparameter tuning intensification, alongside the information of the number of iterations, the lowest points found, and the objective values.

Hyperparameter	Set 1		Set 2		Set 3	
	start	lowest	start	lowest	start	lowest
Edb_switch	1	1	0	0	0	0
Edn_switch	0	0	0	1	0	1
Edt_switch	0	0	0	0	0	0
Edb_PCA_switch	0	1	0	0	0	1
Edn_PCA_switch	0	0	0	1	0	0
Edt_PCA_switch	0	0	0	1	0	0
num_Edb_f	10	6	10	10	10	10
num_Edn_f	10	5	10	5	10	10
num_Edt_f	10	5	10	5	10	10
fmol_PCA_switch	0	0	0	1	0	0
fatom_PCA_switch	0	1	0	0	0	0
num_fmol	50	36	50	31	50	8
num_fatom	50	36	50	34	50	44
n_basis	3	1	3	10	3	3
feature_name	1	3	4	4	5	5
normalize_atom	0	1	0	0	0	1
normalize_mol	0	0	0	1	0	0
model	6	5	5	5	5	5
const	11	8	11	13	11	11
solver	-	-	2	2	2	2
f(x)	7.59	7.09	5.78	5.29	3.72	3.67
total_iter	8689		3829		9824	
lowest_found_at	4085		500		115	

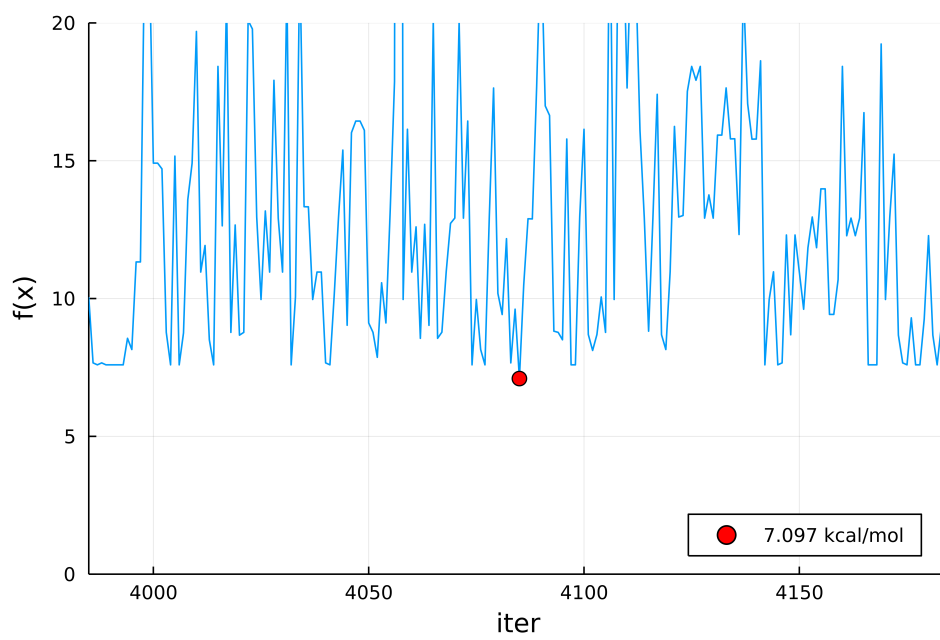


Figure 45: Sample of **Set 1** $f(x)$.

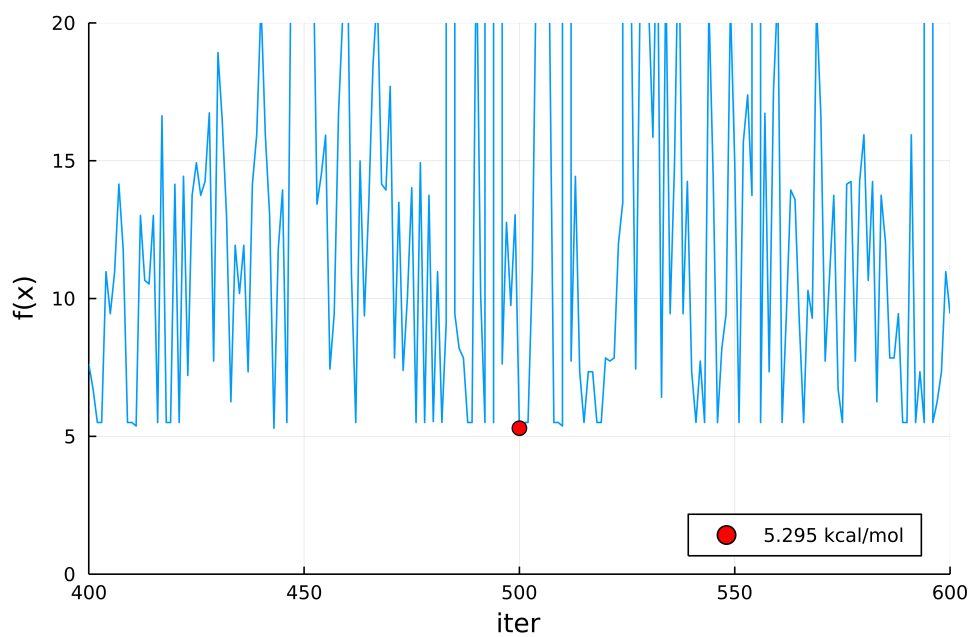


Figure 46: Sample of **Set 2** $f(x)$.

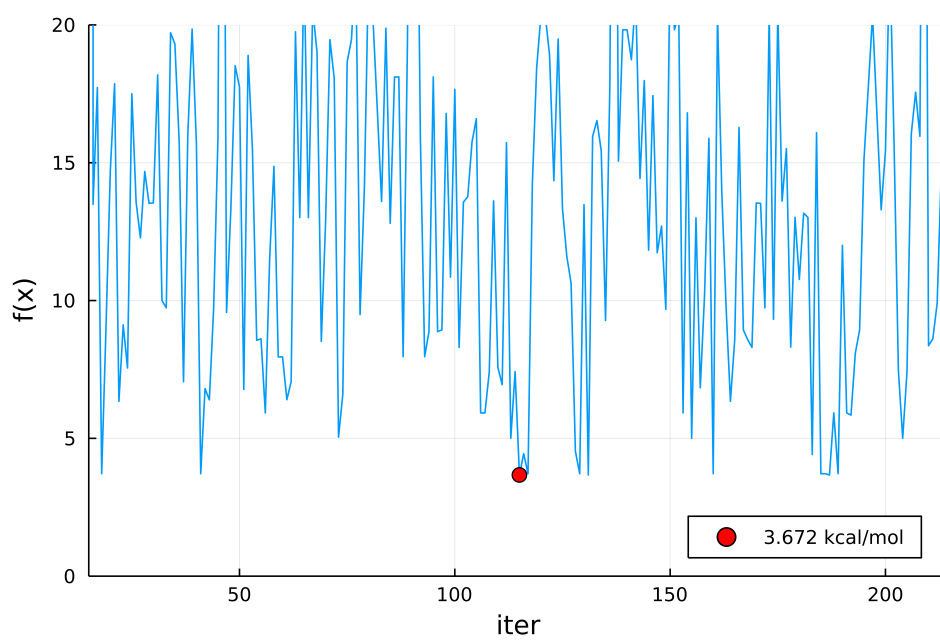


Figure 47: Sample of **Set 3** $f(x)$, one of the lowest MAE ever found, that is 3.67 kcal/mol.

References

- [1] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, “Fast and accurate modeling of molecular atomization energies with machine learning,” *Physical Review Letter*, vol. 108, p. 058301, Jan 2012.
- [2] B. Cheng, “The QM9 challenge: learning quantum chemistry from small dataset.” University website, 2020. Available at https://www.cst.cam.ac.uk/files/project_suggestions_from_bingqing.pdf.
- [3] J. Behler, “Atom-centered symmetry functions for constructing high-dimensional neural network potentials,” *The Journal of Chemical Physics*, vol. 134, p. 074106, 02 2011.
- [4] A. P. Bartók, R. Kondor, and G. Csányi, “On representing chemical environments,” *Physical Review B*, vol. 87, p. 184115, May 2013.
- [5] F. A. Faber, A. S. Christensen, B. Huang, and O. A. von Lilienfeld, “Alchemical and structural distribution based representation for universal quantum machine learning,” *The Journal of Chemical Physics*, vol. 148, p. 241717, 03 2018.
- [6] D. Khan, S. Heinen, and O. A. von Lilienfeld, “Kernel based quantum machine learning at record rate: Many-body distribution functionals as compact representations,” *The Journal of Chemical Physics*, vol. 159, p. 034106, 07 2023.
- [7] K. T. Schütt, S. Chmiela, O. A. von Lilienfeld, A. Tkatchenko, K. Tsuda, and K.-R. Müller, *Machine Learning Meets Quantum Physics*. Springer, 2020. Part of the book series: Lecture Notes in Physics (LNP, volume 968).
- [8] K.-R. Müller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, “An introduction to kernel-based learning algorithms,” *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181–201, 2001.
- [9] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, pp. 5–32, Oct 2001.
- [10] J. Behler and M. Parrinello, “Generalized neural-network representation of high-dimensional potential-energy surfaces,” *Physical Review Letters*, vol. 98, p. 146401, Apr 2007.
- [11] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, “Molecular graph convolutions: moving beyond fingerprints,” *Journal of Computer-Aided Molecular Design*, vol. 30, pp. 595–608, Aug 2016.
- [12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML’17, p. 1263–1272, JMLR.org, 2017.
- [13] J. E. Jones and S. Chapman, “On the determination of molecular fields.—I. From the variation of the viscosity of a gas with temperature, journal = Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character,” vol. 106, no. 738, pp. 441–462, 1924.

- [14] U. K. Deiters and A. Neumaier, "Computer simulation of the characteristic curves of pure fluids," *Journal of Chemical & Engineering Data*, vol. 61, no. 8, pp. 2720–2728, 2016.
- [15] U. K. Deiters and R. J. Sadus, "Two-body interatomic potentials for he, ne, ar, kr, and xe from ab initio data," *The Journal of Chemical Physics*, vol. 150, no. 13, p. 134504, 2019.
- [16] C. M. R. Rocha and A. J. C. Varandas, "A general code for fitting global potential energy surfaces via CHIPR method: Direct-Fit Diatomic and tetratomic molecules," *Computer Physics Communications*, vol. 258, p. 107556, 2021.
- [17] A. S. Christensen, L. A. Bratholm, F. A. Faber, and O. Anatole von Lilienfeld, "FCHL revisited: Faster and more accurate quantum machine learning," *The Journal of Chemical Physics*, vol. 152, p. 044107, 01 2020.
- [18] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nature Communications*, vol. 8, p. 13890, Jan 2017.
- [19] O. A. von Lilienfeld, "The QM9 challenge: Who will be the first to reach 1kcal/mol @ N=100?." Twitter, 2018. Available at <https://twitter.com/ProfvLilienfeld/status/1073179005854121984>.
- [20] O. A. von Lilienfeld, K.-R. Müller, and A. Tkatchenko, "Exploring chemical compound space with quantum-based machine learning," *Nature Reviews Chemistry*, vol. 4, pp. 347–358, Jul 2020.
- [21] O. A. von Lilienfeld and K. Burke, "Retrospective on a decade of machine learning for chemical discovery," *Nature Communications*, vol. 11, p. 4895, Sep 2020.
- [22] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific Data*, vol. 1, p. 140022, Aug 2014.
- [23] T. E. Sharp, "Potential-energy curves for molecular hydrogen and its ions," *Atomic Data and Nuclear Data Tables*, vol. 2, p. 119–169, 1970.
- [24] E. W. E. Wigner, "'Über die Struktur der zweiatomigen Molekelspektren nach der Quantenmechanik'," *Zeitschrift für Physik*, vol. 51, pp. 859–886, 1928.
- [25] M. Born and R. Oppenheimer, "Zur quantentheorie der molekeln," *Annalen der Physik*, vol. 389, no. 20, pp. 457–484, 1927.
- [26] "Energy Conversion Table." <http://wild.life.nctu.edu.tw/class/common/energy-unit-conv-table.html>. Accessed: 2021-09-09.
- [27] D. Weininger, "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules," *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, 1988.
- [28] S. Matsuoka, "mojaie/moleculargraph.jl: v0.16.1," Apr. 2024.

- [29] J. Fairbanks, M. Besançon, S. Simon, J. Hoffman, N. Eubank, and S. Karpinski, “Juliagraphs/graphs.jl: an optimized graphs package for the julia programming language,” 2021.
- [30] D. Khan, “Convolutional MBDF via fast fourier transforms.” Github, 2023. Accessed: 08-04-2024. Available at <https://github.com/dkhan42/cMBDF>.
- [31] B. Onat, C. Ortner, and J. R. Kermode, “Sensitivity and dimensionality of atomic environment representations used for machine learning interatomic potentials,” *The Journal of Chemical Physics*, vol. 153, p. 144106, 10 2020.
- [32] B. Onat, “DescriptorZoo Julia Package to provide single interface for supported descriptors and registry..” Github, 2020. Accessed: 08-04-2024. Available at <https://github.com/DescriptorZoo/DescriptorZoo.jl>.
- [33] L. Himanen, M. O. J. Jäger, E. V. Morooka, F. Federici Canova, Y. S. Ranawat, D. Z. Gao, P. Rinke, and A. S. Foster, “DScribe: Library of descriptors for machine learning in materials science,” *Computer Physics Communications*, vol. 247, p. 106949, 2020.
- [34] A. S. Christensen, F. A. Faber, B. Huang, L. A. Bratholm, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, “qmlcode/qml: Release v0.3.1,” June 2017. Available at <https://doi.org/10.5281/zenodo.817332>.
- [35] K. Yang, K. Swanson, W. Jin, C. Coley, P. Eiden, H. Gao, A. Guzman-Perez, T. Hopper, B. Kelley, M. Mathea, A. Palmer, V. Settels, T. Jaakkola, K. Jensen, and R. Barzilay, “Analyzing learned molecular representations for property prediction,” *Journal of Chemical Information and Modeling*, vol. 59, no. 8, pp. 3370–3388, 2019. PMID: 31361484.
- [36] M. R. Hestenes and E. Stiefel, “Methods of conjugate gradients for solving linear systems,” *Journal of Research of the National Bureau of Standards*, vol. 49, pp. 409–435, 1952.
- [37] Å. Björck, T. Elfving, and Z. Strakos, “Stability of conjugate gradient and lanczos methods for linear least squares problems,” *SIAM Journal on Matrix Analysis and Applications*, vol. 19, no. 3, pp. 720–736, 1998.
- [38] T. Hofmann, B. Schölkopf, and A. J. Smola, “Kernel methods in machine learning,” *The Annals of Statistics*, vol. 36, no. 3, pp. 1171 – 1220, 2008.
- [39] G. V. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.
- [40] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.
- [41] S. Peyerimhoff, J. N. Murrell, S. Carter, S. C. Farantos, P. Huxley, and A. J. C. Varandas, *Molecular Potential Energy Functions*, vol. 89. Wiley; 1st edition, 1985.
- [42] B. J. Braams and J. M. Bowman, “Permutationally invariant potential energy surfaces in high dimensionality,” *International Reviews in Physical Chemistry*, vol. 28, no. 4, pp. 577–606, 2009.

- [43] S. Zou, J. M. B., and A. Brown, “Full-dimensionality quantum calculations of acetylene–vinylidene isomerization,” *The Journal of Chemical Physics*, vol. 118, no. 22, pp. 10012–10023, 2003.
- [44] A. J. C. Varandas, “Combined-hyperbolic-inverse-power-representation of potential energy surfaces: A preliminary assessment for H_3 and HO_2 ,” *The Journal of Chemical Physics*, vol. 138, no. 5, p. 054120, 2013.
- [45] D. V. Coelho and J. Brandão, “A full dimensional potential for H_2O_2 (X^1A) covering all dissociation channels,” *Phys. Chem. Chem. Phys.*, vol. 19, p. 1378–1388, 2017.
- [46] F. G. D. Xavier, M. M. González, and A. J. C. Varandas, “Global Potential Energy Surface for HO_2^+ Using the CHIPR Method,” *The Journal of Physical Chemistry A*, vol. 123, no. 8, pp. 1613–1621, 2019. PMID: 30707025.
- [47] F. G. D. Xavier and A. J. C. Varandas, “Modelling adiabatic cusps in HO_2^+ via 2×2 diabatic matrix,” *Molecular Physics*, vol. 119, no. 10, p. e1904157, 2021.
- [48] J. Brandão and C. Rio, “Double-valued potential energy surface for H_2O derived from accurate ab initio data and including long-range interactions,” *The Journal of Chemical Physics*, vol. 119, pp. 3148–3159, 08 2003.
- [49] J. Brandão, C. Mogo, and B. Silva, “Potential energy surface for $\text{H}_2\text{O}(^3A'')$ from accurate ab initio data with inclusion of long-range interactions,” *The Journal of chemical physics*, vol. 121, pp. 8861–8, 11 2004.
- [50] J. Brandão, C. M. A. Rio, and J. Tennyson, “A modified potential for HO_2 with spectroscopic accuracy,” *The Journal of Chemical Physics*, vol. 130, no. 13, p. 134309, 2009.
- [51] R. J. Le Roy, “*Specialist Periodical Report, Molecular Spectroscopy*,” *The Chemical Society*, p. 113, 1973.
- [52] J. Brandão and C. M. A. Rio, “Long-range interactions within the H_2O molecule,” *Chemical Physics Letters*, vol. 372, no. 5, p. 866–872, 2003.
- [53] A. J. C. Varandas, J. Brandao, and L. A. M. Quintales, “A realistic hydroperoxo($x2a$) potential energy surface from the double many-body expansion method,” *The Journal of Physical Chemistry*, vol. 92, no. 13, p. 3732–3742, 1988.
- [54] A. Varandas and A. Pais, “A realistic double many-body expansion (dmbe) potential energy surface for ground-state o_3 from a multiproperty fit to ab initio calculations, and to experimental spectroscopic, inelastic scattering, and kinetic isotope thermal rate data,” *Molecular Physics*, vol. 65, no. 4, p. 843–860, 1988.
- [55] B. R. L. Galvão, S. P. J. Rodrigues, and A. J. C. Varandas, “Energy-switching potential energy surface for the water molecule revisited: A highly accurate singled-sheeted form,” *The Journal of Chemical Physics*, vol. 129, no. 4, p. 044302, 2008.

- [56] J. Chen, X. Xu, X. Xu, and D. H. Zhang, “A global potential energy surface for the $\text{H}_2 + \text{OH} \leftarrow \text{H}_2\text{O} + \text{H}$ reaction using neural networks,” *The Journal of Chemical Physics*, vol. 138, no. 15, p. 154301, 2013.
- [57] R. A. Buckingham, “The classical equation of state of gaseous helium, neon and argon,” *Proc. R. Soc. London, Ser. A*, vol. 168, pp. 264–283, 1938.
- [58] R. K. Pathak and A. J. Thakkar, “Very short-range interatomic potentials,” *The Journal of Chemical Physics*, vol. 87, no. 4, pp. 2186–2190, 1987.
- [59] U. Hager-Roiser, “The potential energy surface of hydrogen clusters,” Master’s thesis, University of Vienna, 2020.
- [60] W. Kołos and L. Wolniewicz, “Potential-Energy Curves for the $X^1\Sigma_g^+$, $b^3\Sigma_u^+$, and $C^1\Pi_u$ States of the Hydrogen Molecule,” *The Journal of Chemical Physics*, vol. 43, no. 7, p. 2429–2441, 1965.
- [61] L. Wolniewicz, “Relativistic energies of the ground state of the hydrogen molecule,” *The Journal of Chemical Physics*, vol. 99, no. 3, pp. 1851–1868, 1993.
- [62] L. Bytautas, N. Matsunaga, and K. Ruedenberg, “Accurate ab initio potential energy curve of O_2 . II. Core-valence correlations, relativistic contributions, and vibration-rotation spectrum,” *The Journal of Chemical Physics*, vol. 132, no. 7, p. 074307, 2010.
- [63] A. J. C. V. F. George D. Xavier, M. Martínez-González, “Accurate ab initio potential for HO_2^+ : CBS extrapolated energies and direct-fit diatomic curves,” *Chemical Physics Letters*, vol. 691, pp. 421–430, 2018.
- [64] A. Pradhan, “Private communication (to be published).” 1995.
- [65] B. D. Shizgal, “An analysis of O-H interaction potentials, O-H and O-D collision cross sections, and vibrational states,” *Planetary and Space Science*, vol. 47, pp. 163–174, Jan-Feb 1999. ISI Document Delivery No.: 179JTTimes Cited: 3Cited Reference Count: 29.
- [66] A. Nemukhin and B. Grigorenko, “Ab initio potential functions for the ionic states of OH,” *Chemical Physics Letters*, vol. 276, no. 3, pp. 171–176, 1997.
- [67] E. F. van Dishoeck, S. R. Langhoff, and A. Dalgarno, “The low-lying $^2\Sigma^-$ states of OH,” *The Journal of Chemical Physics*, vol. 78, no. 7, pp. 4552–4561, 1983.
- [68] R. Hodges, “Collision cross sections and diffusion parameters for H and D in atomic oxygen. [in upper earth and Venus atmospheres],” 1993.
- [69] S. Chu, M. Yoshimine, and B. Liu, “Ab initio study of the X^2 and A^{2+} states of OH. I. Potential curves and properties,” *The Journal of Chemical Physics*, vol. 61, no. 12, pp. 5389–5395, 1974.
- [70] H. Werner, P. Rosmus, and E. Reinsch, “Molecular properties from MCSCF-SCEP wave functions. I. Accurate dipole moment functions of OH, OH^- , and OH^+ ,” *The Journal of Chemical Physics*, vol. 79, no. 2, pp. 905–916, 1983.

- [71] E. F. van Dishoeck and A. Dalgarno, “Photodissociation processes in the OH molecule,” *The Journal of Chemical Physics*, vol. 79, no. 2, pp. 873–888, 1983.
- [72] D. Cooper, J. Yee, and A. Dalgarno, “Energy transfer in oxygen-hydrogen collisions,” *Planetary and Space Science*, vol. 32, no. 7, pp. 825–830, 1984.
- [73] W. G. Horner and D. Gilbert, “XXI. A new method of solving numerical equations of all orders, by continuous approximation,” *Philosophical Transactions of the Royal Society of London*, vol. 109, pp. 308–335, 1819.
- [74] J. J. Moré, “The Levenberg-Marquardt algorithm: Implementation and theory,” in *Numerical Analysis* (G. A. Watson, ed.), (Berlin, Heidelberg), pp. 105–116, Springer Berlin Heidelberg, 1978.
- [75] M. A. Branch, T. F. Coleman, and Y. Li, “A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems,” *SIAM Journal on Scientific Computing*, vol. 21, no. 1, pp. 1–23, 1999.
- [76] C. Voglis and I. E. Lagaris, “A Rectangular Trust Region Dogleg Approach for Unconstrained and Bound Constrained Nonlinear Optimization,” *WSEAS International Conference on Applied Mathematics, Corfu, Greece*, 2004.
- [77] J. M. White, “LsqFit.jl.” <https://github.com/JuliaNLSolvers/LsqFit.jl>, 2012.
- [78] D. Shepard, “A two-dimensional interpolation function for irregularly-spaced data,” in *Proceedings of the 1968 23rd ACM National Conference*, ACM ’68, (New York, NY, USA), p. 517–524, Association for Computing Machinery, 1968.
- [79] D. H. McLain, “Drawing contours from arbitrary data points,” *Comput. J.*, vol. 17, pp. 318–324, 1974.
- [80] K. C. Thompson, M. J. T. Jordan, and M. A. Collins, “Polyatomic molecular potential energy surfaces by interpolation in local internal coordinates,” *The Journal of Chemical Physics*, vol. 108, pp. 8302–8316, 05 1998.
- [81] A. Pandey and B. Poirier, “An algorithm to find (and plug) “holes” in multi-dimensional surfaces,” *The Journal of Chemical Physics*, vol. 152, p. 214102, 06 2020.
- [82] R. Franke, “Scattered data interpolation: tests of some methods,” *Mathematics of Computation*, vol. 38, pp. 181–200, 1982.
- [83] R. Farwig, “Rate of convergence of shepard’s global interpolation formula,” *Mathematics of Computation*, vol. 46, pp. 577–590, 1986.
- [84] P. Lancaster and K. Salkauskas, “Surfaces generated by moving least squares methods,” *Mathematics of Computation*, vol. 37, pp. 141–158, 1981.
- [85] C. Crespos, M. A. Collins, E. Pijper, and G. J. Kroes, “Application of the modified Shepard interpolation method to the determination of the potential energy surface for a molecule–surface reaction: $H_2 + Pt(111)$,” *The Journal of Chemical Physics*, vol. 120, pp. 2392–2404, 02 2004.

- [86] A. Montoison and D. Orban, “Krylov.jl: A julia basket of hand-picked krylov methods,” *Journal of Open Source Software*, vol. 8, no. 89, p. 5187, 2023.
- [87] D. Orban and A. S. Siqueira, “LinearOperators.jl (v2.8.0).” Zenodo, 2019. Available at <https://doi.org/10.5281/zenodo.11206192>.
- [88] F. B. Carlson, “Hyperopt.jl.” Github, 2018. Accessed: 06-08-2024. Available at <https://github.com/baggepinnen/Hyperopt.jl>.
- [89] M. Kimiaei and A. Neumaier, “Heuristic methods for noisy derivative-free bound-constrained mixed-integer optimization,” *Optimization Online*, 2023.
- [90] M. Kimiaei, “MATRSv2.0,” February 2024. Available at <https://doi.org/10.5281/zenodo.10692874>.
- [91] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi, “The farthest point strategy for progressive image sampling,” *IEEE Transactions on Image Processing*, vol. 6, no. 9, pp. 1305–1315, 1997.
- [92] B. Huang and O. A. von Lilienfeld, “Communication: Understanding molecular representations in machine learning: The role of uniqueness and target similarity,” *The Journal of Chemical Physics*, vol. 145, p. 161102, 10 2016.
- [93] A. K. Rappe, C. J. Casewit, K. S. Colwell, W. A. I. Goddard, and W. M. Skiff, “UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations,” *Journal of the American Chemical Society*, vol. 114, no. 25, pp. 10024–10035, 1992.
- [94] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, “Big Data Meets Quantum Chemistry Approximations: The Δ -Machine Learning Approach,” *Journal of Chemical Theory and Computation*, vol. 11, no. 5, pp. 2087–2096, 2015. PMID: 26574412.
- [95] P. Zaspel, B. Huang, H. Harbrecht, and O. A. von Lilienfeld, “Boosting quantum machine learning models with a multilevel combination technique: Pople diagrams revisited,” *Journal of Chemical Theory and Computation*, vol. 15, no. 3, pp. 1546–1559, 2019. PMID: 30516999.
- [96] S. Wengert, G. Csányi, K. Reuter, and J. T. Margraf, “Data-efficient machine learning for molecular crystal structure prediction,” *Chemical Science*, vol. 12, pp. 4536–4546, 2021.
- [97] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Computers Operations Research*, vol. 13, no. 5, pp. 533–549, 1986. Applications of Integer Programming.
- [98] M. Rupp, “Quantum Mechanics / Machine Learning. A community resource.” Website. "Accessed: 08-04-2024".
- [99] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.
- [100] A. Neumaier and B. Aribowo, “ROSEMI: Robust Shepard model for interpolation.” In preparation, 2024.

- [101] T. Arakaki, “ThreadsX: Parallelized Base functions.” Github, 2022. Accessed: 30-07-2024. Available at <https://github.com/tkf/ThreadsX.jl>.
- [102] J. Revels, “ReverseDiff.jl.” Github, 2020. Accessed: 01-08-2024. Available at <https://github.com/JuliaDiff/ReverseDiff.jl>.
- [103] M. J. Innes, “Zygote.jl.” Github, 2018. Accessed: 01-08-2024. Available at <https://github.com/FluxML/Zygote.jl>.
- [104] W. Moses and C. Valentin, “The Enzyme High-Performance Automatic Differentiator of LLVM.” Github, 2020. Accessed: 01-08-2024. Available at <https://github.com/EnzymeAD/Enzyme.jl>.
- [105] P. K. Mogensen and A. N. Riseth, “Optim: A mathematical optimization package for Julia,” *Journal of Open Source Software*, vol. 3, no. 24, p. 615, 2018.