Master's Thesis

MACHINE LEARNING - BASED EXCITED STATE MOLECULAR DYNAMICS

Kicheol Kim

Department of Chemistry

Graduate School of UNIST

2019

MACHINE LEARNING - BASED EXCITED STATE MOLECULAR DYNAMICS

Kicheol Kim

Department of Chemistry

Graduate School of UNIST

MACHINE LEARNING - BASED EXCITED STATE MOLECULAR DYNAMICS

A thesis/dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Kicheol Kim

06/07/2019 of submission

Approved by

Advisor

Seung Kyu Min

MACHINE LEARNING - BASED EXCITED STATE MOLECULAR DYNAMICS

Kicheol Kim

This certifies that the thesis/dissertation of Kicheol Kim is approved.

06/07/2019

signature

Advisor: Seung Kyu Min

signature

Kwang Soo Kim: Thesis Committee Member #1

signature

Geunsik Lee: Thesis Committee Member #2

Abstract

We present a new methodology for excited state molecular dynamics (ESMD) (or nonadiabatic molecular dynamics) based on machine learning (ML) technique. The most time consuming process in conventional on-the-fly ESMD simulations is an electronic structure calculation including analytic gradients of potential energy surfaces (PESs). Our study proposes that we can bypass this by exploiting ML.

We consider ensemble density functional theory, especially state-interaction state-averaged spin-restricted ensemble-referenced Kohn-Sham (SI-SA-REKS, or SSR for brevity) method as electronic structure calculation for the ML model since SSR(2,2) provides two diabatic electronic states, namely perfectly spin-paired singlet (PPS) and open-shell singlet (OSS), and their analytic gradients as well as interstate couplings (Δ^{SA}).

In this study, we exploit the SchNetPack ML python library for ML procedure. For compatibility, the decoherence-induced surface hopping based on exact factorization (DISH-XF) program is implemented in Python (pyDISH-XF) for nuclear dynamics. Some part of pyDISH-XF is written in C programming language to minimize the slowdown. We investigate ESMD of an ethylene molecule to benchmark pyDISH-XF. The performance of pyDISH-XF is better than UNI-xMD (a reference program is Fortran90). For overall ML-based ESMD, we focus on photo-induced cis-trans isomerization of *trans*-Penta-2,4-dieniminium cation (PSB3), which is a typical model molecule for rhodopsin. The SchNet model is trained with 7500 and 50000 PSB3 geometries getting from previous ESMD studies. We get the poor result with 7500 training set, while the result with 50000 training set is reliable. Mean absolute error (MAE) energy and force of PPS is evaluated 0.01001 eV and 0.01750 eV/Å, MAE energy and force of OSS is evaluated 0.00888 eV and 0.01785 eV/Å, and MAE energy and force of Δ^{SA} is evaluated 0.00948 eV and 0.03720 eV/Å.

We investigate ESMD of PSB3 with ML and compare the result with conventional ESMD of PSB3 with SSR method. The result is comparable to the conventional result. Furthermore, it takes 21 minutes using 1 cpu thread, while the conventional result takes 1.7 days using 2 gpu for propagating one trajectory.

Contents

List of Figures

List of Tables

Explanation of Terms and Abbreviations

ESMD	Excited state molecular dynamics
BO	Born-Oppenheimer
MQC	Mixed quantum-classical
DFT	Density functional theory
ML	Machine learning
DISH-XF	Decoherence-induced surface hopping based on exact factorization
SI-SA-REKS, or SSR	State-interaction state-averaged spin-restricted ensemble-referenced Kohn-Sham
eDFT	Ensemble DFT
PSB3	*Trans*-Penta-2,4-dieniminium Cation
PPS	Perfectly spin-paired singlet
OSS	Open-shell singlet
pyDISH-XF	Python-based DISH-XF program
FSSH	Fewest switch surface hopping
ACSF	Atom-centered symmetry function
DTNN	Deep tensor neural network
cfconv	Continuous-filter convolutional
CI	Conical intersection
FON	Fractional occupation number
KS	Kohn-Sham
MAE	Mean absolute error

I Introduction

Excited state molecular dynamics (ESMD) plays on the important role in photodynamic processes. Born-Oppenheimer (BO) approximation is known as a fundamental tool in chemical reactions. The BO approximation separates the motion of electrons and the motion of nuclei because electrons move much faster than nuclei. However, in ESMD, molecules pass through nonadiabatic coupling regions, where electron-nuclear correlation becomes strong, BO approximation fails to describe nonadiabatic behaviors between multiple BO electronic states. One of the most promising tools to describe the molecular dynamics in nonadiabatic coupling regions more precisely is mixed quantum-classical (MQC) method; nuclei obey classical dynamics while electrons obey quantum mechanics. One of the notable MQC methods is surface hopping technique that nuclei hop from one BO adiabatic surface to another BO adiabatic surface, if calculated hopping probability between two states satisfies some condition. Although many variants of surface hopping were introduced, most of them fails to describe quantum decoherence.

Another problem of existing ab initio-based molecular dynamics simulations is a short time scale. From ab initio to semi-empirical quantum chemistry methods, the quantum mechanical methods have struggled getting efficiency and accuracy. Hartree-Fock approximation is fundamental ab initio method. A many-body electronic wave function is expanded in terms of electronic orbitals, $|\Phi_0\rangle = |\phi_1\phi_2...\phi_n\rangle$. Fock operator is defined, $\hat{f} = \hat{h}(1) + \sum_{j=1}^{N/2}(2\hat{J}_j(1) - \hat{K}_j(1))$, where $\hat{h}(1) = -\frac{1}{2}\nabla_1^2 - \sum_I \frac{Z_I}{r_{1I}}$ is one electron hamiltonian, $\hat{J}_j(1) = \int dr_2 \phi_j^*(2)\frac{1}{r_{12}}\phi_j(2)$ is coulomb operator, and $\hat{K}_j = \int dr_2 \phi_j^*(2)\frac{1}{r_{12}}\phi_i(2)$ is exchange operator. Then, solving eigenfunction equation, $\hat{f}\phi_i = \epsilon_i\phi_i$. The orbital energy of the electronic orbital ϕ_i is ϵ_i. The Hartree-Fock method has defect neglected electron correlation. For considering electron correlation energy, configuration interaction, Møller-Plesset perturbation theory, and density functional theory (DFT) is suggested. However, time scale cannot exceed femtosecond which is very short time to describe realistic molecular dynamics. Machine learning (ML) is a rising methodology overcoming time scale problems.

ML is a task of creating a model that learns from arbitrary data and predicts uneducated values. In the field of machine learning, various kinds of models exist; support vector machine, decision tree learning, and so on. Among them, a deep neural network is in the spotlight. Artificial neural networks are systems that mimic a human brain. Artificial neural networks are networked with artificial neurons, $y = f(\sum_i \mathbf{w}_i\mathbf{x}_i + \mathbf{b}_i)$, like that the neurons are connected by synapses to form the brain structures, where $\mathbf{w}$ is weights, $\mathbf{x}$ is features, $\mathbf{b}$ is bias, and f is activation function. Usually, nonlinear function (ex. sigmoid function) is used by activation function for making network more complicated. Deep neural network is neural network which has hidden layers between an input layer and an output layer as shown in Fig. 1. Layer is a group of artificial neurons. Supervised ML trains the model from a set of examples and labels, where an example is an instance of a feature, a label is a thing to be predicted. During training the model, a loss function is defined. A square error, $l = (y - y')^2$, is commonly used for a

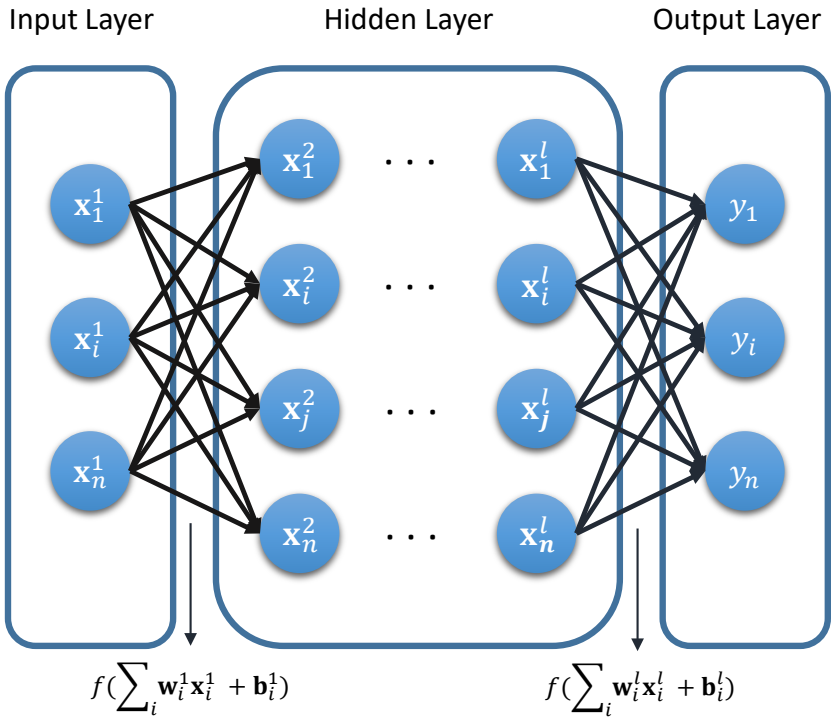

Figure 1: Deep neural network structure.

loss function, where y is a label, y' is a prediction. Then, gradients of weights and losses are calculated, and weights are adjusted to negative gradients for decreasing loss. After iterating those processes until a loss function minimizes, training is done.

ML has strengths in situations where it is difficult to formalize algorithms, such as reading pictures and judging what they are. In this respect, ML can be applied to quantum mechanical computation. This is because the result of quantum mechanics can be obtained through machine learning even if the exact formula is not utilized. There are many attempts to solve quantum mechanics problem through ML in various ways, such as learning to solve the Schrodinger equation [1] and the nonadiabatic excited-state dynamics [2].

In this study, ML is used for computing electronic structures in MQC dynamics. In previous studies, a new MQC formalism is suggested, where the nuclear dynamics follows a decoherence-induced surface hopping based on exact factorization (DISH-XF), while electronic states are obtained from the state-interaction state-averaged spin-restricted ensemble-referenced Kohn-Sham (SI-SA-REKS, or SSR) method based on the ensemble DFT (eDFT) [3]. Then, they are applied to the ESMD of the *trans*-Penta-2,4-dieniminium Cation (PSB3). Instead of the direct (or on-the-fly) SSR electronic structure calculation, a ML model implemented in SchNet

architecture [4] can predict the ground and excited state energies as well as their gradients and nonadiabatic coupling vectors. The model consists of 3 independent models which predict each of perfectly spin-paired singlet (PPS), open-shell singlet (OSS), and interstate coupling parameter, Δ^{SA}, energy and its gradient from the same atomic structure. DISH-XF algorithm is implemented in python program (pyDISH-XF). For much faster computation, C language code is interfaced in pyDISH-XF. In section **II**, we explain DISH-XF formalism, SchNet architecture, and SI-SA-REKS methodology briefly. In section **III**, details of pyDISH-XF, suggested SchNet models to predict electronic properties, and combination of both pyDISH-XF and the model are given. In section **IV**, we explain the result so far; The verification of pyDISH-XF, training the ML model and ESMD of PSB3 with machine learning. In section **V**, we describe the conclusion of ongoing study.

II Theoretical Background

2.1 DISH-XF formalism

Tully's fewest switch surface hopping (FSSH) is the most popular trajectory-based MQC method because of its simplicity and practicality. [5] Because of its fewest switch algorithm, nuclei only hop in nonadiabatic coupling region. FSSH calculates trajectories by Newtonian equation of motion, $M\frac{d^2\mathbf{R}}{dt^2} = -\nabla E$, and propagates electronic wave function $\Phi(\mathbf{r},t)$ by the time-dependent Schrödinger equation for electrons, $i\hbar\frac{\partial}{\partial t}\Phi(\mathbf{r},t) = \hat{H}_{BO}\Phi(\mathbf{r},t)$. BO expansion of an electronic wave function for nuclear trajectory $\mathbf{R}$ is given as,

$$\hat{H}_{BO}\Phi(\mathbf{r},t;\mathbf{R}(t)) = \sum_i C_i(t)\phi_i(\mathbf{r};\mathbf{R}(t)), \tag{1}$$

where $\phi_i(\mathbf{r};\mathbf{R}(t))$ is the wave function of ith BO state. If density matrix $\rho_{ik} = C_i^*(t)C_k(t)$ is defined, the electronic equation of motion can be expressed by the density matrix,

$$\begin{aligned}\frac{d}{dt}\rho_{ik}(t) = &\frac{i}{\hbar}\{E_i(t) - E_k(t)\}\rho_{ik}(t) \\ &- \sum_j \{\sigma_{ij}(t)\rho_{jk}(t) - \rho_{ij}(t)\sigma_{jk}(t)\}\end{aligned} \tag{2}$$

where E_i is the ith state BO energy, σ_{jk} is a nonadiabatic coupling matrix element between the jth and kth BO electronic states. However, FSSH lacks of quantum decoherence. To overcome this problem, DISH-XF method is suggested. [6]

In DISH-XF method, an additional electron-nuclear coupling term from exact factorization of a full molecular wave function [7–9] is introduced,

$$Q_{jk} = \sum_\nu \frac{i\hbar}{M_\nu}\frac{\nabla_\nu|\chi|}{|\chi|}|_{\mathbf{R}(t)} \cdot (\mathbf{f}_{\nu,j} - \mathbf{f}_{\nu,k}), \tag{3}$$

where M_ν is the mass of the νth nucleus, $\frac{\nabla_\nu|\chi|}{|\chi|}$ is a nuclear quantum momentum, and $\mathbf{f}_{\nu,j}$ is an electronic phase at jth BO electronic state. Then, the electronic equation of motion yields,

$$\begin{aligned}\frac{d}{dt}\rho_{ik}(t) = &\frac{i}{\hbar}\{E_i(t) - E_k(t)\}\rho_{ik}(t) \\ &- \sum_j \{\sigma_{ij}(t)\rho_{jk}(t) - \rho_{ij}(t)\sigma_{jk}(t)\} \\ &+ \sum_j \{Q_{ji}(t) + Q_{jk}(t)\}\rho_{ij}(t)\rho_{jk}(t).\end{aligned} \tag{4}$$

The nuclear quantum momenta are calculated from a set of auxiliary nuclear trajectories approximately. An auxiliary trajectory $\mathbf{R}_k(t')$ is generated when $\rho_{kk}(t')$ exceeds a threshold at time t' (i.e. a notable population at the kth state). Then, the nuclear quantum momentum is given as,

$$\frac{\nabla_\nu|\chi|}{|\chi|}|_{\mathbf{R}(t)} = -\frac{1}{2\sigma^2}\{\mathbf{R}_k(t) - \langle\mathbf{R}_k(t)\rangle\} \tag{5}$$

where $\langle \mathbf{R}_k(t) \rangle = \sum_l \rho_{ll} \mathbf{R}_{lk}(t)$, with uniform variance σ. The approximate electronic phase term is evaluated by time integration of momentum change, $\mathbf{f}_{\nu,i} = -\int^t M_\nu \frac{d}{dt'} \mathbf{R}_k(t') dt'$. There is a strong point that more accurate results can be obtained with less resources based on existing results. The hopping probability from the ith BO state to the kth BO state at time step size Δt is calculated,

$$P_{i \to k} = \frac{2Re[\rho_{ik}(t)\sigma_{ik}(t)]}{\rho_{ii}(t)} \Delta t. \tag{6}$$

If hopping probability $P_{i \to k}$ has negative value, it is set to 0. Also, because of total energy conservation, the hopping is forbidden if the kth BO energy, E_k, is greater than the total energy. After hopping is done, the nuclear velocities are rescaled to satisfy the energy conservation. DISH-XF algorithm is proceeded following steps; Step 1: Get BO potential energies each states and nonadiabatic coupling matrix elements, Step 2: Propagate electron density with decoherence term, Step 3: Nuclear dynamics is proceeded by Newtonian equation of motion, Step 4: Calculate hopping probability and determine hopping or not and generate decoherence terms.

2.2 SchNet

To apply ML in chemistry, featuring molecules is a crucial part. There are many trials to describe molecules. One of successful neural network architectures is Behler-Parrinello network. [10] In this network, atom-centered symmetry functions (ACSFs) represent atomistic systems. [11] ACSFs have two terms; radial and angular parts. First, a radial symmetry function is a sum of gaussians,

$$G_i^r = \sum_{j \neq i}^{all} e^{\alpha(R_{ij} - R_s)^2} f_c(R_{ij}), \tag{7}$$

where R_s is a shifted center of gaussian parameters, and $f_c(R_{ij})$ is a cutoff function with the width parameter α. Second, an angular symmetry function

$$G_i^a = 2^{1-\beta} \sum_{j,k \neq i}^{all} (1 + \lambda \cos\theta_{ijk})^\beta \times e^{-\alpha(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk}), \tag{8}$$

is the sum of cosine functions of angles θ_{ijk} centered at the ith atom with an angular resolution parameter β. The parameter λ can have values $+1$ or -1 corresponding to $\theta_{ijk} = 0°$ and $180°$, respectively, which make maxima of the cosine function. ACSFs have benefits (1) invariance with respect to translation and rotation of the system, (2) independence of atomic coordinate. For reliable ACSFs, finding suitable parameters is an important task. Thus, most of time are consumed to find proper parameters.

Deep tensor neural network (DTNN) solves above problems providing atom embedding and interaction refinements. [12] SchNet is one of variants of DTNN. Atoms are described tensor $\mathbf{x}_i \in \mathbb{R}^F$, with features F. $\mathbf{x}_i$ is initialized depending on an atomtype Z_i, $\mathbf{x}_i = \mathbf{a}_{Z_i}$, where $\mathbf{a}_{Z_i}$ is a random value to be optimized during the training process. Each feature tensor is propagated through atom-wise layers, $\mathbf{x}_i = W\mathbf{x}_i + b$. Because the same atoms share weights W and bias b,

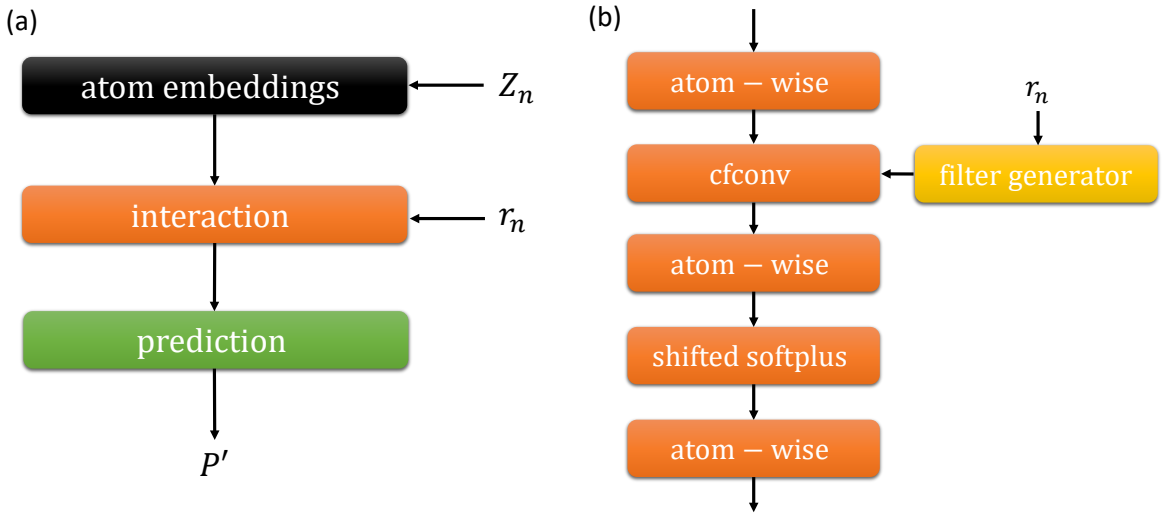

Figure 2: (a) Brief illustration of SchNet architecture. Z_n is atomtype and r_n is atomic coordinates each atom n. P' is predicted property. (b) Illustration of interaction parts.

SchNet can apply any atomistic systems irrespective of the number of atoms. The interaction part refines atom embeddings with pairwise interactions of surrounding atoms. SchNet use continuous-filter convolutional (*cfconv*) layers. *cfconv* layers are used to generalize discrete convolutional layers. It is helpful to consider atomic positions without atomic coordinates. In this model, *cfconv* layers,

$$\mathbf{x}_i = \sum_{j=0}^{n_{atoms}} \mathbf{x}_j \circ W^f(r_j - r_i), \tag{9}$$

with $W^f : \mathbb{R}^3 \to \mathbb{R}^F$ that maps the atomic coordinates to features F, generated by filter-generating neural network. The operation $\circ$ represents an element-wise multiplication. Interaction part consists of 3 atom-wise layers, *cfconv* layer and activation functions. Each layer is located in between atom-wise layers as shown in Fig. 2(b). Shifted softplus, $ssp(x) = \ln(0.5e^x + 0.5)$, is used for activation functions. Filter-generating network considers rotational invariance and periodic boundary conditions within neighbor atoms. Neighbor atoms are atoms in a cutoff range. Prediction of given property P is a sum of atom-wise layer outputs,

$$P = \sum_{i}^{n_{atoms}} \mathbf{x}_i. \tag{10}$$

Instead of predicting atomic forces directly, SchNet differentiates a predicted energy function with respect to atomic positions,

$$F'_i(Z_1, ..., Z_n, \mathbf{r}_1, ..., \mathbf{r}_n) = -\frac{\partial E'}{\partial \mathbf{r}_i}(Z_1, ..., Z_n, \mathbf{r}_1, ..., \mathbf{r}_n). \tag{11}$$

For the sake of training the model, SchNet minimizes the loss function, $l(P, P') = \|P - P'\|^2$. For the training both energies and forces of molecular dynamics trajectories, a combined loss

function is used,

$$l((E', F_1', ...F_n'), (E, F_1, ...F_n)) = \rho\|E - E'\|^2 + \frac{1-\rho}{n_{atoms}} \sum_i^{n_{atoms}} \|F_i - (-\frac{\partial E'}{\partial \mathbf{r}_i})\|^2, \quad (12)$$

with trade-off parameter ρ. [13]

2.3 SI-SA-REKS Method

We exploit the SSR method based on eDFT as an electronic structure calculation. [14–19] While the conventional DFT and time dependent DFT fail to capture the correct topology of the PESs in the conical intersections (CIs) vicinity, the SSR method enables much accurate description of the vertical excitation energies with the S_1/S_0 CIs matching the results calculated from the multireference ab initio wave function methods. The SSR methodology employs ground state eDFT to describe multireference ground states of molecules and eDFT for ensembles of ground and excited state to obtain excitation energies from a variational time-independent formalism. The concept of the fractional occupation numbers (FONs) of several frontier Kohn-Sham (KS) orbitals appears due to the use of ensemble representation of the density and the energy. Here, we use the active space including two electrons in two KS orbitals (i.e. SSR(2,2)) to deal with the π/π^* transitions of PSB3.

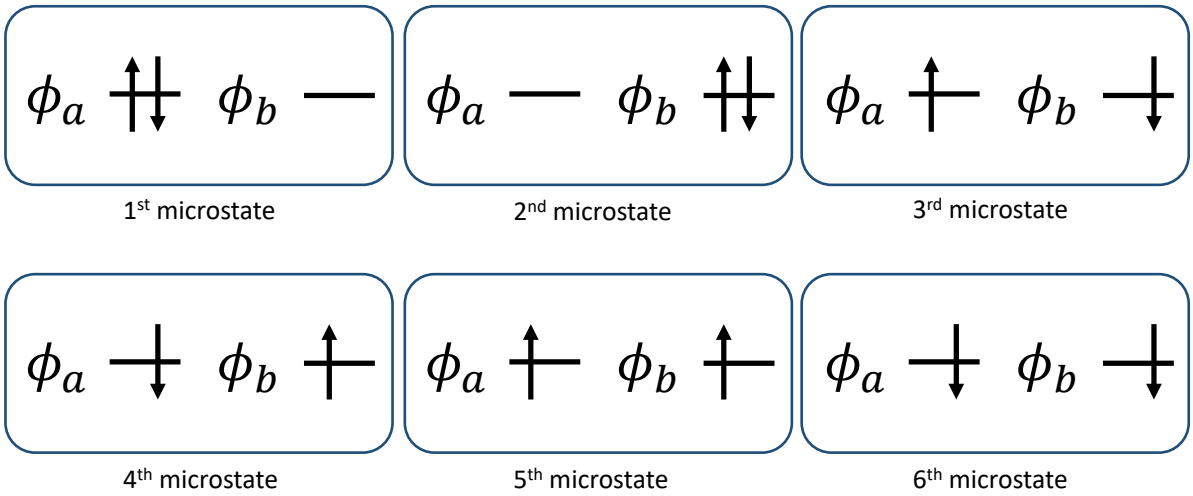

Figure 3: Six microstates in the SSR(2,2) method

In the SSR(2,2) method, the energy of the multireference state is expanded in terms of six microstates with the fixed and integer occupations of the active orbitals ϕ_a and ϕ_b as shown in Fig. 3. The energies of ground and excited states are obtained by minimizing an ensemble of the PPS and OSS states with respect to the KS orbitals and their FONs. The energies of PPS and OSS states, E^{PPS} and E^{OSS}, are expressed as an ensemble average over six microstates as

$$E^{PPS} = \sum_{L=1}^{6} C_L^{PPS} E_L \quad (13)$$

$$E^{OSS} = \sum_{L=3}^{6} C_L^{OSS} E_L \tag{14}$$

where $C_1^{PPS} = \frac{n_a}{2}$, $C_2^{PPS} = \frac{n_b}{2}$, $C_3^{PPS} = C_4^{PPS} = -C_5^{PPS} = -C_6^{PPS} = -\frac{f(n_a/2)}{2}$, and $C_3^{OSS} = C_4^{OSS} = -2C_5^{OSS} = -2C_6^{OSS} = 1$ with n_a and n_b is FONs of the active KS orbitals each ϕ_a and ϕ_b and $f(n_a/2) = (n_a n_b)^{1-1/2((n_a n_b+\delta)/(1+\delta))}$ with $\delta = 0.4$ subject to $n_a + n_b = 2$. The equal weightings in the SA-REKS(2,2) energy $E^{SA} = w_0 E^{PPS} + w_1 E^{OSS}$ are chosen. Minimizing the SA-REKS(2,2) energy leads to a single particle equation,

$$n_p \hat{F}_p \phi_p = \sum_q \epsilon_{pq}^{SA} \phi_q \tag{15}$$

where the Fock operator $\hat{F}_p$ of the pth orbital ϕ_p is given by

$$\hat{F}_p = \sum_L C_L^{SA} \frac{n_{p\alpha}^L \hat{F}_\alpha^L + n_{p\beta}^L \hat{F}_\beta^L}{n_p} \tag{16}$$

where $p \in core, a, b$. Here, the average occupation number $n_p = \sum_L C_L^{SA}(n_{p\alpha}^L + n_{p\beta}^L)$ with $C_L^{SA} = w_0 C_L^{PPS} + w_1 C_L^{OSS}$, and the Largrangian matrix elements ϵ_{pq}^{SA} are obtained. The Fock operator of each microstate L is calculated within spin-restricted DFT framework.

So far, the S_0 and S_1 states are assumed decoupled by symmetry. If there is no symmetry or the two states belong in the same irreducible representation, we do not need to consider the interaction between the states. However, the assumption that the two states in SA-REKS(2,2) are decoupled by symmetry is often not realistic, thus the inclusion of states interactions becomes important. The interstate coupling Δ^{SA} is calculated using the converged SA-REKS(2,2) Lagrangian matrix elements ϵ_{ab}^{SA} between the active orbitals ϕ_a and ϕ_b as

$$\Delta^{SA} = (\sqrt{n_a} - \sqrt{n_b})\epsilon_{ab}^{SA}. \tag{17}$$

The SSR states are obtained by solving the following 2×2 secular equation

$$\begin{pmatrix} E^{PPS} & \Delta^{SA} \\ \Delta^{SA} & E^{OSS} \end{pmatrix} \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} = \begin{pmatrix} E_0^{SSR} & 0 \\ 0 & E_1^{SSR} \end{pmatrix} \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \tag{18}$$

Therefore, the ground and the lowest excited state energies, E_0^{SSR} and E_1^{SSR}, are obtained as

$$E_I^{SSR} = a_{0I}^2 E^{PPS} + a_{1I}^2 E^{OSS} + 2a_{0I}a_{1I}\Delta^{SA} \tag{19}$$

where $I = 0$ and 1. The derivation of analytic gradients for E^{PPS}, E^{OSS}, and Δ^{SA} can be found elsewhere. [20] The gradients of the SSR(2,2) states ∇E_I^{SSR} are obtained as

$$\nabla E_I^{SSR} = a_{0I}^2 \nabla E^{PPS} + a_{1I}^2 \nabla E^{OSS} + 2a_{0I}a_{1I}\nabla\Delta^{SA} \tag{20}$$

The nonadiabatic coupling vectors can be expanded in terms of the gradients of the SSR(2,2)

states ∇E_I^{SSR} and of the interstate coupling $\nabla \Delta^{SA}$,

$$\begin{aligned}\mathbf{H}_{01} =& \frac{1}{E_1^{SSR} - E_0^{SSR}} \Bigg(\frac{2(a_{00}a_{01} - a_{10}a_{11})}{a_{00}^2 - a_{01}^2 - a_{10}^2 + a_{11}^2} \mathbf{G}_{01} \\ & - \frac{2(a_{00}a_{01} - a_{10}a_{11})(a_{00}a_{10} - a_{01}a_{11})}{a_{00}^2 - a_{01}^2 - a_{10}^2 + a_{11}^2} \mathbf{h}_{01} \\ & + (a_{00}a_{11} + a_{01}a_{10}) \mathbf{h}_{01} \Bigg)\end{aligned} \tag{21}$$

where $\mathbf{G}_{01} = \frac{1}{2}(\nabla E_0^{SSR} - \nabla E_1^{SSR})$, $\mathbf{h}_{01} = \nabla \Delta^{SA}$.

III Computational Method

3.1 Python-based DISH-XF program (pyDISH-XF)

pyDISH-XF is an object-oriented program whose classes are organized into individual processes. All processes are managed as objects, which is advantageous for a large number of tasks. For faster computation, electron dynamics parts is implemented in C language and interfaced in pyDISH-XF using cython library.

```
from atoms import Atoms
from bo import Bo
from dynamics import Dynamics
from sh import Sh
from shxf import Shxf
from thermostat import *
import json

if __name__ == "__main__":
    with open("unixmd.json") as f:
        args = json.load(f)
    traj = Atoms("unixmd.xyz",args)
    bo = Bo(args)
    sh = Sh(args)
    shxf = Shxf(args)
    if(args["thermo_type"] == "rescale1"):
        thermostat = velocity_rescale(args)
    elif(args["thermo_type"] == "rescale2"):
        thermostat = velocity_rescale_dtemp(args)
    else:
        thermostat = thermo(args)
    md = Dynamics(args)
    md.run(traj,bo,sh,shxf,thermostat)
```

The above code is a code example using pyDISH-XF. The parameters to define computations such as molecular dynamics scheme, thermostats, and initial conditions are received in a dictionary in the program or can be pre-declared externally through *json*. The types of input parameters are described in the table 1 below. After loading input parameters, getting input geometry from *"unixmd.xyz"* using Atoms class. Then, declare *Bo*, *Sh*, *Shxf*, *thermostat* and *Dynamics* objects. *Bo* object executes quantum mechanics program and parses energies, forces, and nonadiabatic coupling vector or matrix elements. *Sh* object calculates hopping probability and decides that hopping is occurred or not. *Shxf* object generates auxiliary trajectory and calculates decoherence terms. Finally, running molecular dynamics based on declared objects in *Dynamics* object.

name	description
natoms	number of atoms
nsp	number of nuclear dimensions per atom
nstates	number of BO states
istate	the initial state
nsteps	number of time steps
nesteps	number of time steps for the electronic problem
dt	size of time step
temperature	temperature for MD, in Kelvin
dtemp	temperature tolerance for velocity rescale
nrescale	velocity rescale frequency
rho_threshold	threshold for BO population
wsigma	variance for trajectory
outfreq	output frequency
l_simplehop	logical for hopping according to energy only
l_adjnac	logical for adjusting NAC
l_allphase_dp	logical for calculating all phase from momentum changes (SHXFMQC)
l_nacme	read *NACME.DAT* instead of *NAC.DAT*

Table 1: Input parameters of pyDISH-XF

3.1.1 API Document

atoms.py

class Atoms(*fname, args*)

Get input geometry from given files.

Parameters:
- fname(*str*) - name of input geometry file
- args(*dictionary*) - input parameters

write_traj(*dir*)

Write current positions in *TRAJECOTRY* file.

Parameters:
- dir(*str*) - path of proceeding calculation

write_traj_xyz(*dir*)

Write current positions in *TRAJECOTRY.xyz* file in the .xyz format.

Parameters:
- dir(*str*) - path of proceeding calculation

write_movie(*dir, first*)

Accumulate current positions and velocities in *MOVIE* file as dynamics progresses.

Parameters:
- dir(*str*) - path of proceeding calculation
- first(*bool*) - If true, write new *MOVIE* file.

write_movie_xyz(*dir, first*)

Accumulate current positions in *MOVIE.xyz* file in the .xyz format as dynamics progresses.

Parameters:
- dir(*str*) - path of proceeding calculation
- first(*bool*) - If true, write new *MOVIE.xyz* file.

write_vel(*dir*)

Write current velocities in *VELOCITY* file.

Parameters:
- dir(*str*) - path of proceeding calculation

bo.py

class Bo(*args*)

Class for quantum mechanics part of the system.

Parameters:
- args(*dictionary*) - input parameters

get_BO(*dir, l_nac*)

Run quantum mechanics program and call *read_BO_files*().

Parameters:
- dir(*str*) - path of proceeding calculation
- l_nac(*bool*) - If true, call *adjust_nac*().

get_nacme(*traj*)

Get nonadiabatic coupling matrix elements from nonadiabatic coupling vectors.

Parameters:
- traj(*Atoms*) - *Atoms* object which has current atomic information.

adjust_nac()

Change sign of nonadiabatic coupling vectors if the overlap of current nonadiabatic coupling vectors and past nonadiabatic coupling vectors have negative sign.

read_BO_files()

Read energies, forces, and nonadiabatic coupling vectors or matrix elements which are got from quantum mechanics.

cl_dynamics.py

class Cl_dynamics(*args*)

Set classic mechanics part of the system.

Parameters:
- args(*dictionary*) - input parameters

cl_update_position(*coef, traj, bo, sh*)

Proceed to classical dynamics to obtain positions.

Parameters:
- coef(*np.array*) - coefficients of BO wave function.
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.

cl_update_position(*coef, traj, bo, sh*)

Proceed to classical dynamics to obtain positions.

Parameters:
- coef(*np.array*) - coefficients of BO wave function.
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.

cl_update_velocity(*coef, traj, bo, sh*)

Proceed to classical dynamics to obtain velocities.

Parameters:
- coef(*np.array*) - coefficients of BO wave function.
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.

calculate_force(*coef, traj, bo, sh*)

Calculate nuclei forces.

Parameters:
- coef(*np.array*) - coefficients of BO wave function.
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.

dynamics.py

class Dynamics(*args*)

Class for main dynamics.

Parameters:
- args(*dictionary*) - input parameters

run(*traj, bo, sh, shxf, thermostat*)

Run MQC dynamics loop.

Parameters:
- coef(*np.array*) - coefficients of BO wave function.
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.
- shxf(*Shxf*) - *Shxf* object which has decoherence information.
- thermostat(*thermo*) - *thermo* object which has thermostat information.

write_md_trajinfo(*traj*)

Call *write_traj*() and *write_traj_xyz*()

Parameters:
- traj(*Atoms*) - *Atoms* object which has current atomic information.

write_md_output(*istep, traj, bo, sh*)

Call *write_vel*(), *write_movie*(), *write_movie_xyz*(), *write_md_energy*(), *write_BO_coefs*(). If *first* is true, set *first* to false.

Parameters:
- istep(*int*) - current dynamics step.
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.

write_md_energy(*istep, traj, bo, sh*)

Accumulate kinetic energy, potential energy, and total energy in *MDENERGY* file.

Parameters:
- istep(*int*) - current dynamics step.
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.

write_BO_coefs()

Accumulate BO coefficient using density matrix diagonal term and off-diagonal term in *BOCOEF* file.

el_dynamics.pyx

el_propagator(*el_propagator, nst, nat, nsp, nesteps, dt, wsigma, traj, bo, sh, shxf, rho, coef*)

Change python variables which using electron dynamics to C variable. Call C function for electron dynamics. Runge-Kutta method is used for propagating electron dynamics.

Parameters:
- el_propagator(*str*) - method of electron propagator {*coefficient* or *density*}
- nst(*int*) - total number of states.
- nat(*int*) - total number of atoms.
- nsp(*int*) - total number of nuclear dimensions per atom.
- nesteps(*int*) - total number of time steps for electronic problem.
- dt(*int*) - size of time step.
- wsigma(*int*) - variance for trajectory.
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.
- shxf(*Shxf*) - *Shxf* object which has decoherence information.
- coef(*np.array*) - coefficients of BO wave function.
- rho(*np.array*) - coefficients density matrix of BO wave function.

sh.py

class Sh(*args*)

Class for surface hopping methods.

Parameters:
- args(*dictionary*) - input parameters

surface_hopping(*dir, traj, bo, rho, istep*)

Run surface hopping dynamics.

Parameters:
- dir(*str*) - path of proceeding calculation
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- rho(*np.array*) - coefficients density matrix of BO wave function.
- istep(*int*) - current dynamics step.

hop_prob(*rho, bo*)

Calculate hopping probability.

Parameters:
- rho(*np.array*) - coefficients density matrix of BO wave function.
- bo(*Bo*) - Bo object which has current quantum mechanics information.

write_sh_output(*dir, rho*)

Call *write_sh_hopping_prob*(), *write_sh_running_state*(), and *write_sh_nacme*(). If *io_first* is true, set *io_first* to false.

Parameters:
- dir(*str*) - path of proceeding calculation
- rho(*np.array*) - coefficients density matrix of BO wave function.

write_sh_nacme(*dir, rho*)

Write nonadiabatic coupling matrix elements in *NACME* file.

Parameters:
- dir(*str*) - path of proceeding calculation
- rho(*np.array*) - coefficients density matrix of BO wave function.

write_sh_hopping_prob(dir)

Accumulate hopping probability in *SHPROB* file.

Parameters:
- dir(*str*) - path of proceeding calculation

write_sh_running_state(*dir*)

Accumulate running state in *SHSTATE* file.

Parameters:
- dir(*str*) - path of proceeding calculation

shxf.py

class Shxf(*args*)

Class for decoherence term.

Parameters:
- args(*dictionary*) - input parameters

decoherence_shxf(*traj, bo, sh, rho, coef*)

Run decoherence term calculation.

Parameters:
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.
- rho(*np.array*) - coefficients density matrix of BO wave function.
- coef(*np.array*) - coefficients of BO wave function.

get_aux_pos(*traj, sh*)

Generate auxiliary trajectories.

Parameters:
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.

get_aux_vel(*traj, sh, bo*)

Calculate velocity of auxiliary trajectories.

Parameters:
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.
- bo(*Bo*) - *Bo* object which has current quantum mechanics information.

get_nabph(*traj, sh*)

Calculate electronic phase.

Parameters:
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.

get_qmcdot(*traj, sh, coef, rho*)

Calculate decoherence term using BO coefficient.

Parameters:
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- sh(*Sh*) - *Sh* object which has current surface hopping information.
- coef(*np.array*) - coefficients of BO wave function.
- rho(*np.array*) - coefficients density matrix of BO wave function.

thermostat.py

class thermo(*args*)

Class for thermostat base.

Parameters:
- args(*dictionary*) - input parameters

class velocity_rescale(*args*)

Class for thermostat which rescales velocities at each target step.

Parameters:
- args(*dictionary*) - input parameters

run(*traj, shxf*)

Rescale velocity at each target step.

Parameters:
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- shxf(*Shxf*) - *Shxf* object which has decoherence information.

class velocity_rescale_dtemp(*args*)

Class for thermostat which rescales velocities if current temperature over temperature tolerance.

Parameters:
- args(*dictionary*) - input parameters

run(*traj, shxf*)

Rescale velocity if current temperature over temperature tolerance.

Parameters:
- traj(*Atoms*) - *Atoms* object which has current atomic information.
- shxf(*Shxf*) - *Shxf* object which has decoherence information.

3.2 Extension of SchNet to excited states

Obtaining the ground and excited state energies, forces and nonadiabatic coupling vector is an object of quantum chemical calculations in DISH-XF. To replace quantum chemical calculations with ML, we have two choices in training processes; (1) training adiabatic properties, and (2) training diabatic properties. Since adiabatic electronic energies and their gradients can have nonanalytic behaviors in the vicinity of CIs, training diabatic energies and gradients may be more suitable choice for ESMD simulation. In this sense, SSR approach is beneficial since SSR provides diabatic states (PPS and OSS) and couplings (Δ^{SA}) as well as gradients directly. From identical atomic positions $\mathbf{R}_i$, 3 independent SchNet models train each PPS, OSS, Δ^{SA} energy and force. Each model consists of $F = 256$ features and $T = 6$ interaction blocks. A cutoff radius is set to 20 Å, which can cover all atoms. Output layer is set predicting energy with force. For training the model, initial learning late is set to 0.0001. Instead of exponential decay, ReduceLROnPlateau method is used to decrease learning rate. It decreases learning late when training loss is not improved during the number of patience epoch. A combined loss function, $l((E', F'_1, ...F'_n), (E, F_1, ...F_n)) = \rho\|E - E'\|^2 + \frac{1-\rho}{n_{atoms}} \sum_i^{n_{atoms}} \|F_i - (-\frac{\partial E'}{\partial \mathbf{r}_i})\|^2$, is used. All models are implemented based on SchNetPack. [4, 12, 21–25]

3.3 Combination of pyDISH-XF and SchNet

The SchNet models predict PPS, OSS, Δ^{SA} energies and forces. For the use in DISH-XF method, they should be converted into ground, excited energies and forces and nonadiabatic coupling vectors. Ground state energy E_0, excited state energy E_1, each of gradient ∇E_0, ∇E_1, and nonadiabatic coupling vector $\mathbf{H}_{01}$ can be expressed as following equations. First, ground and excited energy can get from diagonalization of given matrix,

$$P^{-1} \begin{pmatrix} E^{PPS} & \Delta^{SA} \\ \Delta^{SA} & E^{OSS} \end{pmatrix} P = \begin{pmatrix} E_0 & 0 \\ 0 & E_1 \end{pmatrix} \tag{22}$$

,where P is $\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}$. Gradients of ground and excited state can be calculated the following equation,

$$\nabla E_0 = a_{00}^2 \nabla E^{PPS} + 2a_{00}a_{10}\nabla\Delta^{SA} + a_{10}^2 \nabla E^{OSS} \tag{23}$$

$$\nabla E_1 = a_{01}^2 \nabla E^{PPS} + 2a_{11}a_{01}\nabla\Delta^{SA} + a_{11}^2 \nabla E^{OSS}. \tag{24}$$

Then, nonadiabatic coupling vector can be express as

$$\mathbf{H}_{01} = \frac{1}{E_1 - E_0}((a_{00}a_{01} - a_{10}a_{11})\mathbf{g}_{01} + (a_{00}a_{11} + a_{01}a_{10})\mathbf{h}_{01}), \tag{25}$$

where $\mathbf{g}_{01} = \frac{1}{2}(\nabla E^{PPS} - \nabla E^{OSS})$, and $\mathbf{h}_{01} = \nabla\Delta^{SA}$. After getting energies, gradients, and nonadiabatic coupling vector, energies are written in *ENERGY.DAT*, forces $F_i = -\nabla E_i$ are written in *FORCE.DAT* and nonadabatic coupling vector is written in *NAC.DAT* for parsing in pyDISH-XF program.

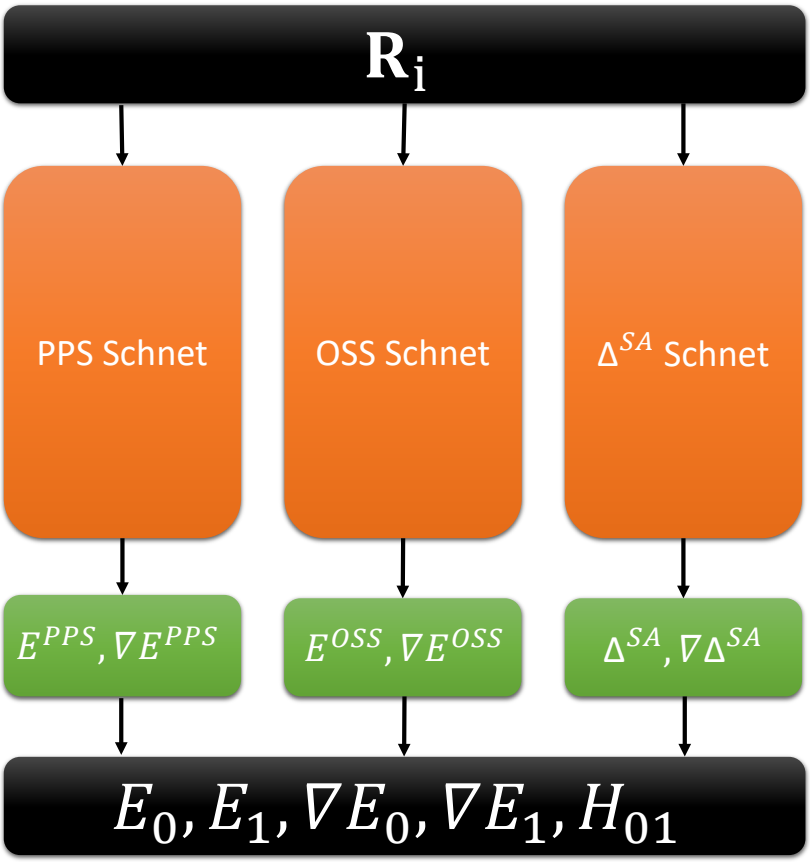

Figure 4: ML architecture to replace quantum mechanics

IV Result and Discussion

4.1 Verification of pyDISH-XF program

To checking pyDISH-XF program, we perform ESMD of C_2H_4 molecules. The initial geometry is set as a slightly modified the ground state geometry. We exploit DFTB/SSR method for electronic structure calculation. [26] Both FSSH and DISH-XF dynamics are done with UNI-xMD program [6] and pyDISH-XF program.

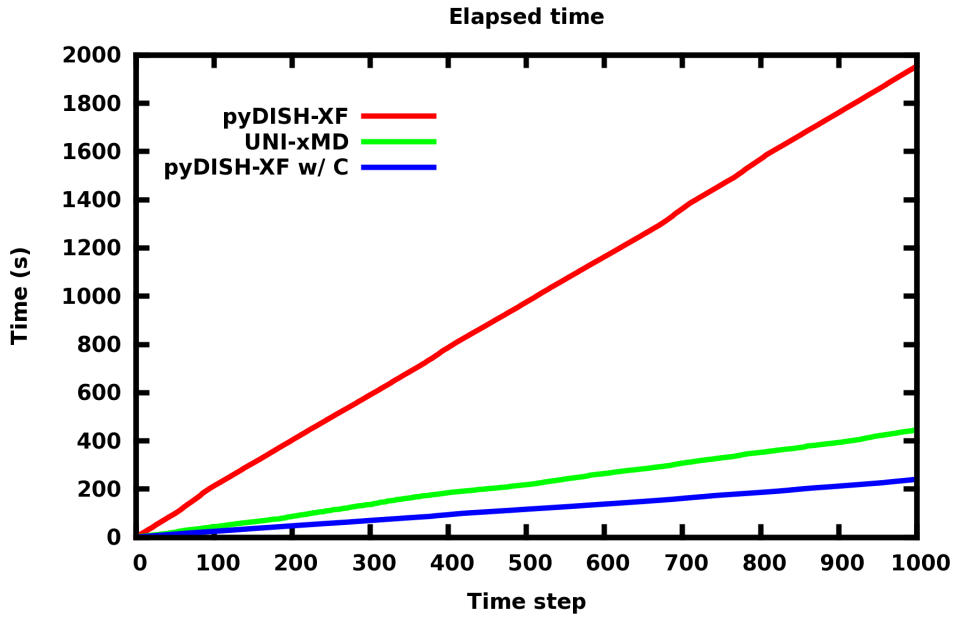

Figure 5: FSSH dynamics of C_2H_4.

FSSH dynamics is propagated up to 1000 steps with time step 0.12 fs shown in Fig. 5. Surface Hopping is done at 395th step forcibly for the same dynamics condition. UNI-xMD takes about 445 seconds. pyDISH-XF takes about 1952 seconds which is 4 times longer than UNI-xMD. For finding the reason why it takes much time, time analysis is performed. Python has defect that is slowdown in loop state. We find that elapsed time is proportional to the number of loop steps. In electronic propagator, 10000 electronic steps is proceeded. As a result, the bottleneck is electronic propagator part of program. Alternatively using python, electronic propagator is altered C code. After that, pyDISH-XF interfaced with C code takes about 240 seconds which is 1.9 times faster than UNI-xMD program.

DISH-XF dynamics is also propagated up to 1000 steps with time step 0.12 fs shown in Fig. 6. Surface Hopping is done at 362th step forcibly for the same dynamics condition. UNI-xMD takes about 312 seconds. pyDISH-XF takes about 2711 seconds which is 8.6 times longer than UNI-xMD. Additional calculation of decoherence terms also takes long times in python. It is not followed previous study that calculation of decoherence term is in the short time. [6] After converting python to C code, it takes about 227 seconds which is 1.4 times faster than

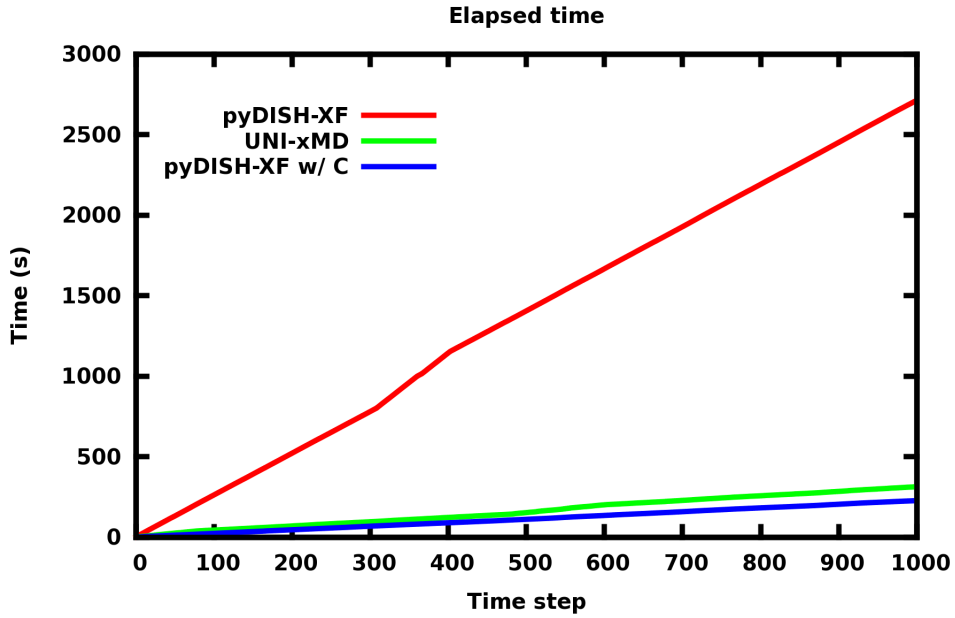

Figure 6: DISH-XF dynamics of C_2H_4.

UNI-xMD program. Also, pyDISH-XF program obtains total energy, ground state energy and excited state energy within 1e-5 errors compared to UNI-xMD program both surface hopping and DISH-XF dynamics described in Fig. 8 and almost the same BO population of state 1 and state 2 obtained pyDISH-XF and UNI-xMD at the surface hopping and DISH-XF dynamics described in Fig. 7.

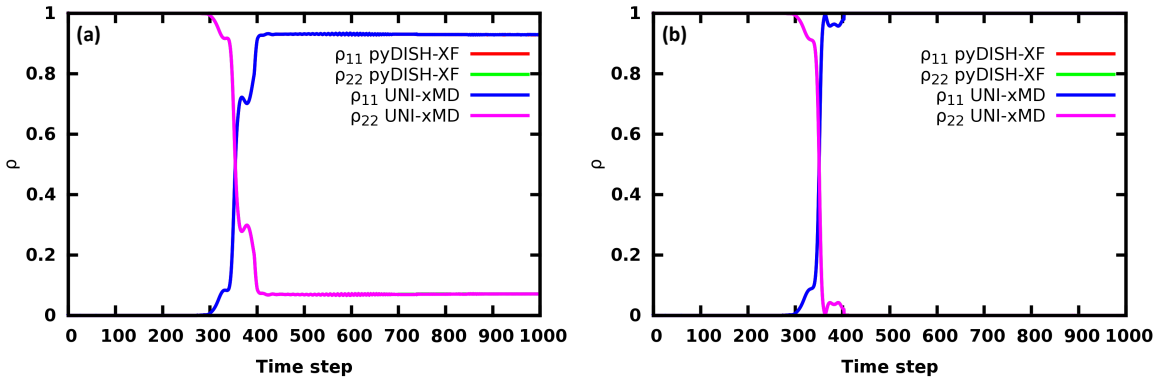

Figure 7: (a) BO population of state 1 and state 2 at Surface Hopping. (b) BO population of state 1 and state 2 at DISH-XF

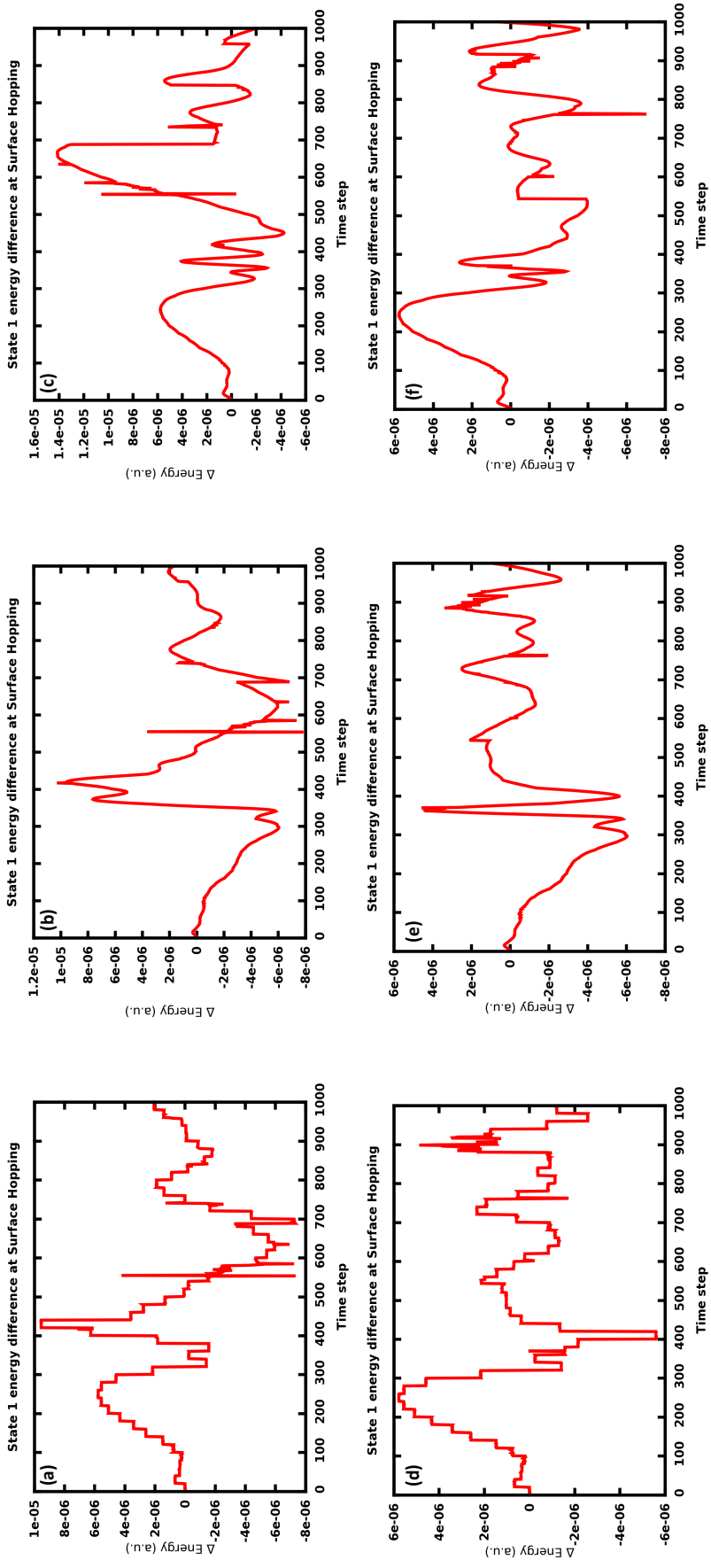

Figure 8: (a) Difference of total energy, (b) difference of state 1 energy, (c) difference of state 2 energy at Surface Hopping. Corresponding to (d), (e), (f) at DISH-XF

4.2 Training the model

Training dataset is calculated using the beta-testing version of the TeraChem program (v1.92P, release 7f19a3bb8334). [27–32] Total 62500 geometries are generated by ESMD of PSB3 that 50 trajectories are propagated up to 1250 steps with PPS and OSS of energies and gradients, and Δ^{SA} and its gradients. We train the model with two different conditions which are summarized in Table 2. First, 7500 geometries by obtaining 1250 steps each 6 of trajectories are used to train the model with trade-off between energy force loss, $\rho = 0.1$ and patience for ReduceLROnPlateau method is 100 at PPS, OSS and Δ^{SA} model. Initial learning rate is 1e-4. If the model is not improved during the number of patience epoch, learning rate is reduced to 0.5. Training is proceeded until learning rate is reduced to 1e-6. The model is trained with mini-batch stochastic gradient descent using ADAM optimizer. [33] Batch size is set to 100.

model	Learning rate	Decay Factor	Minimal Learning Rate	Patience	ρ	Batch Size
			$\boldsymbol{N = 7500}$			
PPS	0.0001	0.5	1e-6	100	0.1	100
OSS	0.0001	0.5	1e-6	100	0.1	100
Δ^{SA}	0.0001	0.5	1e-6	100	0.1	100
			$\boldsymbol{N = 50000}$			
PPS	0.0001	0.5	1e-6	50	0.1	100
OSS	0.0001	0.5	1e-6	50	0.1	100
Δ^{SA}	0.0001	0.5	1e-6	50	0.1	100

Table 2: Setup for training model

Fig. 9 demonstrates the variation of mean absolute error (MAE) energy and force in PPS, OSS and Δ^{SA} model during training. MAE energies are more fluctuating than MAE forces, because ρ is set to give weight to force loss. Because Δ^{SA} properties are 1000 times smaller than PPS and OSS, Δ^{SA} model is converged much faster than PPS and OSS model.

We evaluate the models using 55,000 geometries by obtaining 1250 steps each 44 trajectories. MAE energy and force of PPS are 0.32320 eV and 0.38730 eV/Å. MAE energy and force of OSS are 0.41073 eV and 0.44411 eV/Å. MAE energy and force of Δ^{SA} are 0.08501 eV and 0.20221 eV/Å. Compared to result of training set, all of MAEs are relatively high values. The result shows that this model predicts suitable properties only in training sets. It cannot be substituted SSR methods.

Then, we increase the training set from 7500 to 50000. It is randomly chosen from 62500 geometries by obtaining 1250 steps each 50 of trajectories. Patience for ReduceLROnPlateau method is 50 at PPS, OSS and Δ^{SA} model. MAE energy and force of PPS are reduced to 0.01001 eV and 0.01750 eV/Å. MAE energy and force of OSS are reduced to 0.00888 eV and 0.01785 eV/Å. MAE energy and force of Δ^{SA} are reduced to 0.00948 eV and 0.03720 eV/Å. The

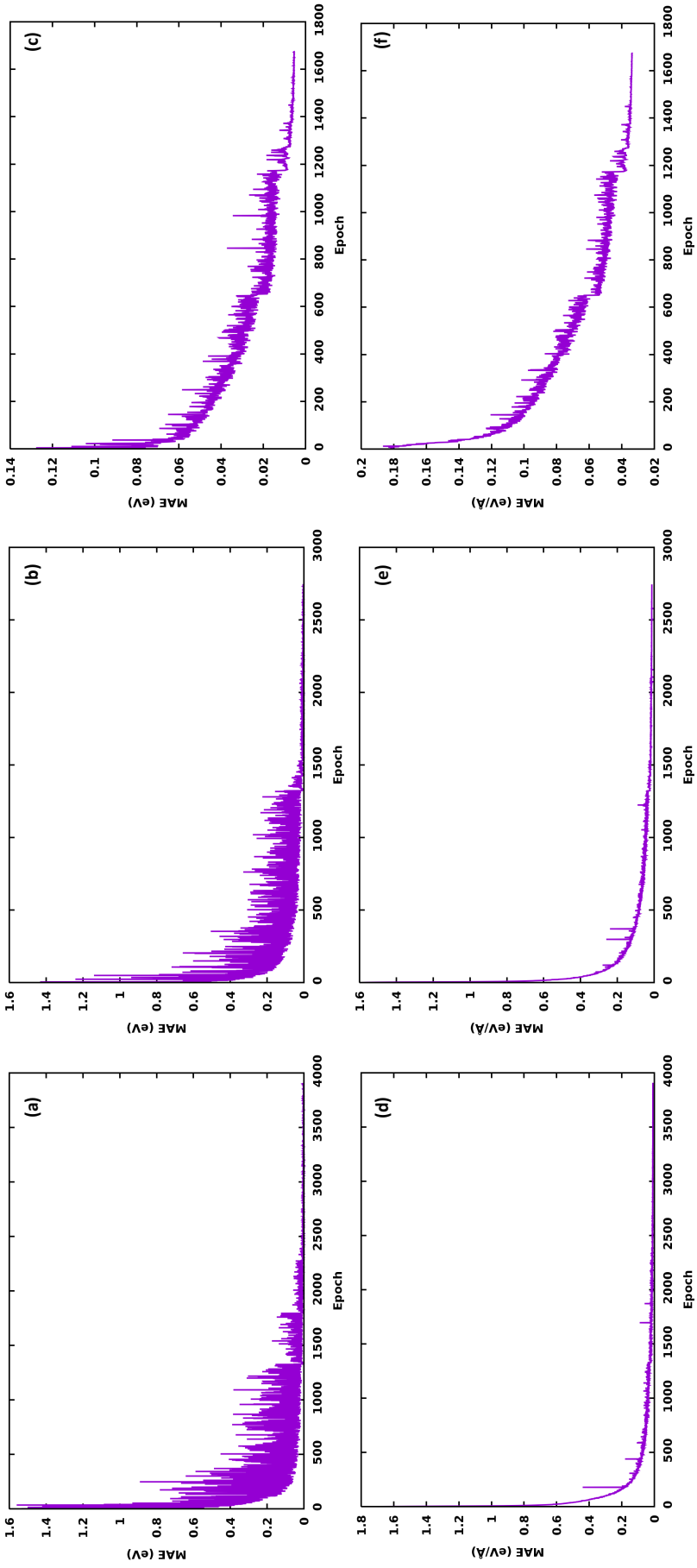

Figure 9: (a) MAE energy of PPS, (b) MAE energy of OSS, (c) MAE energy of Δ^{SA}. (d) MAE force of PPS, (e) MAE force of OSS, (f) MAE force of Δ^{SA}

	$N = 7500$		$N = 50000$	
model	MAE Energy (eV)	MAE Force (eV/Å)	MAE Energy (eV)	MAE Force (eV/Å)
PPS	0.32320	0.38730	0.01001	0.01750
OSS	0.41073	0.44411	0.00888	0.01785
Δ^{SA}	0.08501	0.20221	0.00948	0.03720

Table 3: Evaluation result of the model

results are summarized in Table 3. It is more suitable than the result getting from 7500 training model.

4.3 ESMD of PSB3 with Machine Learning

Total 100 trajectories are propagated up to 1250 steps with the time step 0.24 fs. Initial geometries are generated by the TeraChem software by Wigner sampling at $T = 300$ K in previous study. [3] We exploit pyDISH-XF with the SchNet model that is trained by 50000 training set. Dynamics is proceeded under microcanonical ensemble.

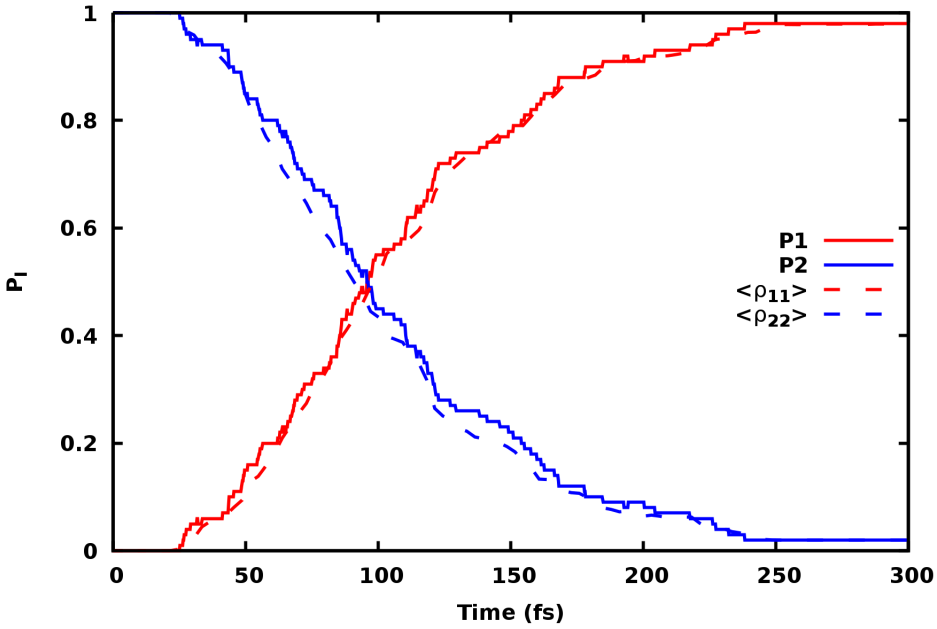

Figure 10: BO populations of PSB3 ESMD

We analyze BO populations of PSB3 ESMD. Two types of BO populations are used; P_I is obtained from the numbers of trajectories running on the Ith state, $\langle \rho_{II} \rangle$ is averaged BO population over all trajectories. Out of 2 trajectories are not hopped that is the reason why final BO populations are not 0 and 1. Fig. 10 shows two types of BO populations have the same tendency that decoherence is well applied in ESMD.

We compare our result with ESMD of PSB3 with SSR method. [3] Both P_I and $\langle \rho_{II} \rangle$ have

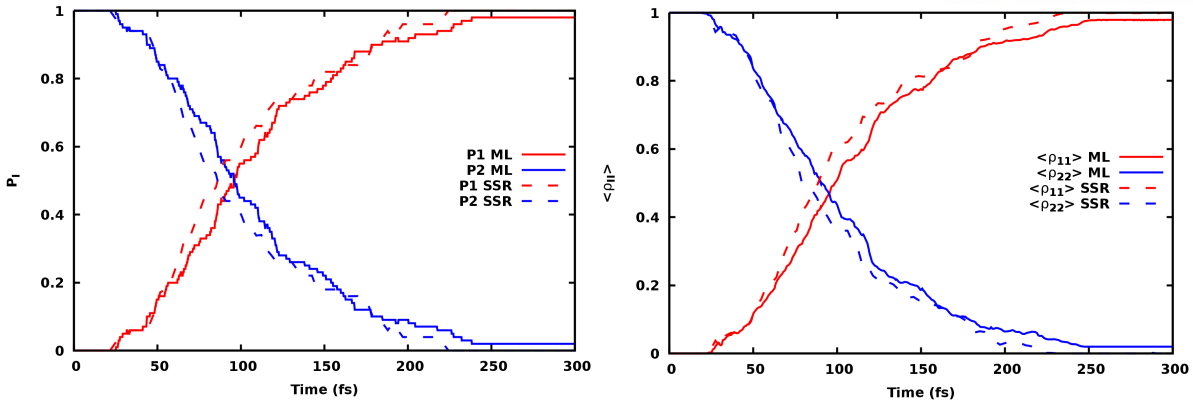

Figure 11: BO populations of PSB3 ESMD with ML and SSR method

comparable result. DISH-XF with SSR dynamics takes 1.7 days to finish one trajectory using 2 gpu. However, DISH-XF with ML dynamics takes 21 minutes to finish one trajectory using 1 cpu thread. It gets 117 times faster result despite low computational cost.

V Conclusion

In this study, machine learning has been attempted to replace SSR methodology in DISH-XF formalism. pyDISH-XF is implemented with python interfaced with the external C code that propagates electronic steps for combining DISH-XF method and machine learning method easily. ESMD of an ethylene molecule is proceeded with UNI-xMD program to benchmark pyDISH-XF program. pyDISH-XF program generates encouraging results that calculation is faster than UNI-xMD with reliable results.

The SchNet is used for training PPS, OSS and Δ^{SA} energies and forces. The model consists of three SchNet models each train three properties respectively implemented in SchNetPack; deep neural network python library. PSB3 database is generated by the beta-testing version of the TeraChem program (v1.92P, release 7f19a3bb8334) which contains 62500 geometries consist of 1250 steps each 50 trajectories. However, the model trained with 7500 geometries couldn't predict given properties; PPS, OSS, and Δ^{SA} energies and forces. Then, we increase training set from 7500 to 50000. As a result, the error is reduced then we proceed dynamics to using this model.

For comparing the result to conventional result, ESMD with machine learning is propagated in the same condition. It gets a comparable result within 21 minutes using 1 cpu thread compared to 1.7 day using 2 gpu at the conventional result. Therefore, we achieve the object that decreasing computational time.

References

[1] K. Mills, M. Spanner, and I. Tamblyn, "Deep Learning and the Schrödinger equation," *Phys. Rev. A*, vol. 96, p. 042113, 2017.

[2] W.-K. Chen, X.-Y. Liu, W.-H. Fang, P. O. Dral, and G. Cui, "Deep Learning for Nonadiabatic Excited-State Dynamics," *J. Phys. Chem. Lett.*, vol. 9, pp. 6702–6708, 2018.

[3] M. Filatov, S. K. Min, and K. S. Kim, "Direct Nonadiabatic Dynamics by Mixed Quantum-Classical Formalism Connected with Ensemble Density Functional Theory Method: Application to trans-Penta-2,4-dieniminium Cation," *J. Chem. Theory Comput.*, vol. 14, pp. 4499–4512, 2018.

[4] K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller, "SchNet - A deep learning architecture for molecules and materials," *J. Chem. Phys.*, vol. 148, p. 241722, 2018.

[5] J. C. Tully, "Molecular dynamics with electronic transitions," *J. Chem. Phys.*, vol. 93, p. 1061, 1990.

[6] J.-K. Ha, I. S. Lee, and S. K. Min, "Surface Hopping Dynamcis beyond Nonadiabatic Couplings for Quantum Coherence," *J. Phys. Chem. Lett.*, vol. 9, pp. 1097–1104, 2018.

[7] A. Abedi, N. T. Maitra, and E. K. U. Gross, "Exact Factorization of the Time-Dependent Electron-Nuclear Wave Function," *Phys. Rev. Lett.*, vol. 105, p. 123002, 2010.

[8] A. Abedi, N. T. Maitra, and E. K. U. Gross, "Correlated electron-nuclear dynamics: Exact factorization of the molecular wavefunction," *J. Chem. Phys.*, vol. 137, p. 22A530, 2012.

[9] N. I. Gidopoulos and E. K. U. Gross, "Electronic non-adiabatic states: towards a density functional theory beyond the Born–Oppenheimer approximation," *Philos. Trans. R. Soc. A*, vol. 372, p. 20130059, 2014.

[10] J. Behler and M. Parrinello, "Generalized Neural-Netwrok Representation of High-Dimensional Potential-Energy Surfaces," *Phys. Rev. Lett.*, vol. 98, p. 146401, 2007.

[11] J. Behler, "Atom-centered symmetry functions for constructing high-dimensional neural network potentials," *J. Chem. Phys.*, vol. 134, p. 074106, 2011.

[12] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nat. Commun.*, vol. 8, p. 13890, 2017.

[13] A. Pukrittayakamee, M. Malshe, M. Hagan, L. M. Raff, R. Narulkar, S. Bukkapatnum, and R. Komanduri, "Simultaneous fitting of a potential-energy surface and its corresponding force fields using feedforward neural networks," *J. Chem. Phys.*, vol. 130, p. 134101, 2009.

[14] S. M. Valone, "A one-to-one mapping between one-particle densities and some n-particle ensembles," *J. Chem. Phys.*, vol. 73, p. 4653, 1980.

[15] E. H. Lieb, "Density functionals for coulomb systemses," *Int. J. Quantum Chem.*, vol. 24, pp. 243–277, 1983.

[16] J. P. Perdew, R. G. Parr, M. Levy, and J. Jose L. Balduz, "Density-Functional Theory for Fractional Particle Number: Derivative Discontinuities of the Energy," *Phys. Rev. Lett.*, vol. 49, pp. 1691–1694, 1982.

[17] H. Englisch and R. Englisch, "Hohenberg-Kohn theorem and non-V-representable densities," *Phys. A*, vol. 121, pp. 253–268, 1983.

[18] H. Englisch and R. Englisch, "Exact Density Functionals for Ground-State Energies. I. General Results," *Phys. Status Solidi B*, vol. 123, pp. 711–721, 1984.

[19] H. Englisch and R. Englisch, "Exact Density Functionals for Ground-State Energies II. Details and Remarks," *Phys. Status Solidi B*, vol. 124, pp. 373–379, 1984.

[20] M. Filatov, F. Liu, and T. J. Martínez, "Analytical derivatives of the individual state energies in ensemble density functional theory method. I. General formalism," *J. Chem. Phys.*, vol. 147, p. 034133, 2017.

[21] K. T. Schütt, P. Kessel, M. Gastegger, K. A. Nicoli, A. Tkatchenko, and K.-R. Müller, "SchNetPack: A Deep Learning Toolbox For Atomistic Systems," *J. Chem. Theory Comput.*, vol. 15, pp. 448–455, 2019.

[22] K. T. Schütt, P.-J. Kindermans, H. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "SchNet: A continuous-filter convolutional neural network for modeling quantum interactions," *Adv. Neural Inf. Process. Syst.*, pp. 991–1001, 2017.

[23] M. Gastegger, L. Schwiedrzik, M. Bittermann, F. Berzsenyi, and P. Marquetand, "wACSF—Weighted atom-centered symmetry functions as descriptors in machine learning potentials," *J. Chem. Phys.*, vol. 148, p. 241709, 2018.

[24] J. Behler and M. Parrinello, "Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces," *Phys. Rev. Lett.*, vol. 98, p. 146401, 2007.

[25] J. Smith, O. Isayev, and A. Roitberg, "ANI-1, A data set of 20 million calculated off-equilibrium conformations for organic molecules.," *Sci. Data*, vol. 4, p. 170193, 2017.

[26] I. S. Lee, M. Filatov, and S. K. Min, "Formulation and Implementation of the Spin-Restricted Ensemble-Referenced Kohn-Sham Method in the Context of the Density Functional Tight Binding Approach," *J. Chem. Theory Comput.*, vol. 15, pp. 3021–3032, 2019.

[27] I. S. Ufimtsev and T. J. Martínez, "Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-Electron Integral Evaluation," *J. Chem. Theory Comput.*, vol. 4, pp. 222–231, 2008.

[28] I. S. Ufimtsev and T. J. Martínez, "Quantum Chemistry on Graphical Processing Units. 2. Direct Self-Consistent-Field Implementation," *J. Chem. Theory Comput.*, vol. 5, pp. 1004–1015, 2009.

[29] I. S. Ufimtsev and T. J. Martínez, "Quantum Chemistry on Graphical Processing Units. 3. Analytical Energy Gradients, Geometry Optimization, and First Principles Molecular Dynamics," *J. Chem. Theory Comput.*, vol. 5, pp. 2619–2628, 2009.

[30] I. S. Ufimtsev and T. J. Martínez, "Graphical Processing Units for Quantum Chemistry.," *Comput. Sci. Eng.*, vol. 10, pp. 26–34, 2008.

[31] A. V. Titov, I. S. Ufimtsev, N. Luehr, and T. J. Martinez, "Generating Efficient Quantum Chemistry Codes for Novel Architectures," *J. Chem. Theory Comput.*, vol. 9, pp. 213–221, 2013.

[32] C. Song, L.-P. Wang, and T. J. Martínez, "Automated Code Engine for Graphical Processing Units: Application to the Effective Core Potential Integrals and Gradients," *J. Chem. Theory Comput.*, vol. 12, pp. 92–106, 2016.

[33] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimizatio," *ArXiv preprint*, p. arXiv:1412.6980, 2017.