

UNIVERSITY OF  
**Waterloo**



**LabMind: An R&D workflow and data management  
orchestration system demonstrated on Path Integral Molecular  
Dynamics Simulations**

**Course: NE459 Nanotechnology Engineering Research Project**

**Author: Yunheng Zou**

**Student Number: 20833174**

**Supervisor: Professor Pierre-Nicholas Roy**

**Second Reader: Professor Nasser Mohieddin Abukhdeir**

**Submitted: 2024/04/08**

## Acknowledgement

I would like to thank Professor Pierre-Nicholas for supervising this work. My gratitude also extends to Professor Nasser Mohieddin Abukhdeir for being my second reader. I would also like to acknowledge Prof. Youngki Yoon for serving as the course coordinator. Additionally, I am grateful to the people who allowed me to interview them during the development of the software automation system. They provided valuable insights and mentoring, including:

### **Within University of Waterloo:**

Pierre-Nicholas Roy (Professor, Canada Research Chair in Quantum Molecular Dynamics, Chemistry)

Conrard Giresse Tetsassi Feugmo (Assistant Professor, Chemistry)

Michael Pope (Associate Professor, Chemical Engineering)

Nicholas Wilson (PhD Candidate, Chemical Engineering)

Agosh Saini (Master Student, Mechanical and Mechantronics Engineering)

Tamer Shahin (Senior Manager, Velocity Digital)

Pascal Poupart (Professor & CIFAR AI Chair at the Vector Institute, Computer Science)

Xander Gouws (Master Student, Chemistry)

### **Externally:**

Daniel Li (PhD, Chief Scientific Officer, Hansen Advanced Materials)

Willi Gottstein (PhD, Senior Scientist, DSM-Firmenich)

Yixuan Li (PhD, Senior Cell Engineer, Tesla Inc.)

Jakob Zeitler (PhD, Research Scientist, Matterhorn Studio)

Gabe Graves (Master Student, Georgia Institute of Technology)

Mark Wilson (Associate Professor, University of Toronto)

Alana Ogata (Assistant Professor, University of Toronto)

The concept quickly gained traction and successfully advanced to the semifinals of the Velocity Pitch Competition, ranking among the top 24 out of over 100 participating teams. I was also invited to Waterloo Institute for Nanotechnology

Pitches and Demos Day to share the idea to Waterloo researchers.

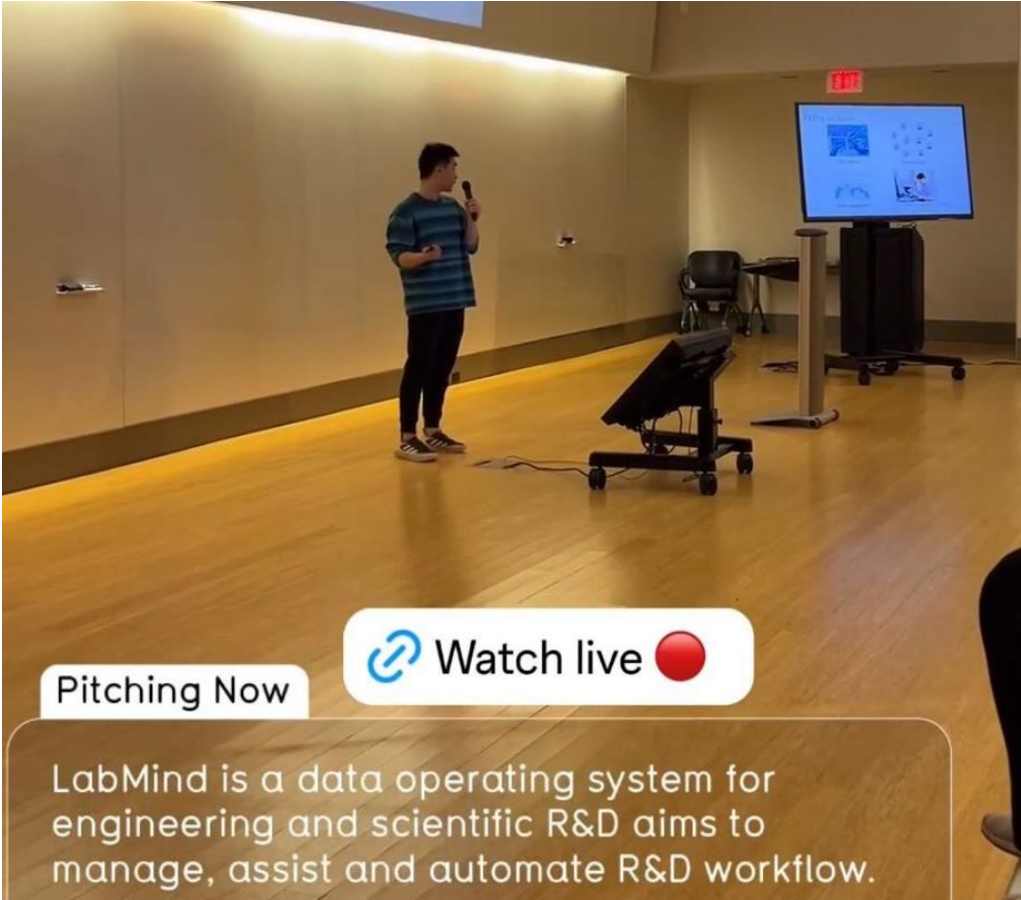


Figure 1. Velocity Pitch Competition Semifinal

# Table of Contents

## Contents

Acknowledgement .....	2
Table of Contents .....	3
Abstract .....	5
Introduction .....	5
Knowledge Background .....	6
Path Integral Molecular Dynamics .....	6
Data Lakehouse .....	8
Workflow Manager .....	9
LLM Agent .....	9

Simulation Design and Methodology.....	10
Software Architecture.....	10
Class simulation_run_time .....	11
Class simulation_states.....	13
Optimization .....	14
Force Field Selection .....	15
Parameter Selection and Optimization.....	15
Steps.....	15
Equilibration Steps .....	16
Friction Coefficient.....	16
Time Per step (dt) .....	17
Temperature .....	17
Beads.....	17
Error Bar minimization.....	18
LabMind Orchestrator Design.....	19
Data entity .....	19
FileObject .....	19
KnowledgeObject.....	20
Automation .....	20
Data Science.....	21
Construct LLM Agent and copilots.....	22
Results.....	25
Conclusions .....	26
Recommendations .....	26
References .....	28

# Abstract

This report presents a simulation of low-temperature water using the ring polymer path integral molecular dynamic (RPPIMD) simulation. We successfully utilized the path integral technique to capture the quantum effects associated with the ground state energy of water monomers, simulated at 20K, 30K, 40K, and 50K. We systematically optimized the simulation algorithms and parameters with the q-TIP4P/F force field and demonstrated the accuracy of our model by comparing it with literature values. Additionally, we introduce a novel central orchestrator, named the LabMind system. We demonstrated that LabMind can efficiently orchestrate different simulation procedures and perform automatic FAIR data management for simulation raw data, leading to a productivity boost during the research process.

## Introduction

This report introduces a comprehensive study divided into two interconnected parts, each of paramount importance for advancing the field of computational simulations. The first part unveils LabMind, an innovative orchestrator system designed to automate and streamline simulation pipelines. LabMind simplifies computational science, offering capabilities for constructing automated pipelines, managing simulation tasks, facilitating seamless transfer of data to cloud storage and connection to AI agents. Its development heralds a new era of efficiency and integration in computational research workflows.

The second part of our study delves into quantum mechanical simulations of water at low temperatures, employing the ring polymer path integral molecular dynamics (RPPIMD) technique. This investigation focuses on elucidating the quantum effects on the ground state energy of water monomers at temperatures ranging from 20K to 50K. By leveraging the q-TIP4P/F force field and optimizing simulation parameters, this research not only sheds light on the intrinsic properties of water at quantum levels but also serves as a critical demonstration of LabMind's capabilities.

The integration of the LabMind system with the path integral water simulation exemplifies the synergy between advanced computational tools and scientific inquiry. LabMind's role transcends mere data management; it embodies foundational support for conducting sophisticated simulations, ensuring the fidelity, reproducibility, and accessibility of research data. Thus, this report presents findings

on the quantum behaviors of water and showcases LabMind as a pivotal innovation in the automation and management of scientific simulations.

By harmonizing these two significant endeavors, our research contributes to the broader scientific community's understanding of both quantum simulations and the essential tools needed to facilitate such complex analyses.

## Knowledge Background

### Path Integral Molecular Dynamics

Quantum Particles, different than classical ones, can be seen as probabilities waves. Thus, traditional Molecular Dynamics, cannot efficiently approximate those quantum effects.

Partition function  $Z$ , the most important quantity in statistical mechanics, can be used to compute various system properties.

In a canonical system, the expression of the partition function is shown as

$$Z = \sum_n e^{-\beta E_n} = \underbrace{\text{Tr} e^{-\beta \hat{H}}}_{\text{most general}} \quad (1)$$

Further expression for quantum system can be derived as

$$Z = \text{Tr} e^{-\beta \hat{H}} = \sum_n e^{-\beta E_n} = \int d^3 r_1 \int d^3 r_2 \cdots \int d^3 r_N \langle r_1, r_2, \dots, r_N | e^{-\beta \hat{H}} | r_1, r_2, \dots, r_N \rangle \quad (2)$$

where integral expression is used when energy states  $E_n$  are unknown.

$$Z_{\text{Classical}} = \frac{1}{h^{3N}} \int d^3 p_1 \cdots \int d^3 p_N \int d^3 r_1 \cdots \int d^3 r_N e^{-\beta p^2/2m} e^{-\beta V(r_1, \dots, r_N)} \quad (3)$$

Integral expression also holds for classical partition function.

To compute quantum partition function, analysis in 1D is conducted.

$$\hat{H} = \frac{\hat{p}^2}{2m} + \hat{V}(x) = \hat{K} + \hat{V}$$

$$Z = \int dx \langle x | e^{-\beta \hat{H}} | x \rangle = \int dx \langle x | e^{-\beta(\hat{K} + \hat{V})} | x \rangle \quad (4)$$

Due to uncommuted nature of quantum operators, errors are introduced if following expression must be used.

This operation is wrong

$$e^{-\beta(\hat{K}+\hat{V})} \rightarrow e^{-\beta\hat{K}}e^{-\beta\hat{V}}$$

as

$$[\hat{K}, \hat{V}] = \hat{K}\hat{V} - \hat{V}\hat{K} \neq 0 \quad (5)$$

To minimize the effect of the error, trotter factorization is introduced. By factorize  $\beta$  with P. As P approximates infinite, Trotter Error infinitely close to 0.

$$e^{-\beta(\hat{K}+\hat{V})} = (e^{-\frac{\beta}{P}(\hat{K}+\hat{V})})^P$$

$$\tau = \beta/P. \quad (6)$$

$$Z = \int dx \langle x | (e^{-\tau\hat{V}/2} e^{-\tau\hat{K}} e^{-\tau\hat{V}/2})^P | x \rangle \quad (7)$$

By removing operators, we acquire computable partition functions as shown below.

$$\langle x_i | e^{-\tau\hat{V}/2} e^{-\tau\hat{K}} e^{-\tau\hat{V}/2} | x_{i+1} \rangle = \left( \frac{m}{2\pi\hbar^2\tau} \right)^{1/2} \exp \left( -\frac{m}{2\hbar^2\tau} (x_i - x_{i+1})^2 - \frac{\tau}{2} [V(x_i) + V(x_{i+1})] \right) \quad (8)$$

$$Z = \int dx_1 \cdots \int dx_P \prod_{i=1}^P \left( \frac{m}{2\pi\hbar^2\tau} \right)^{1/2} \exp \left( -\frac{m}{2\hbar^2\tau} (x_i - x_{i+1})^2 - \frac{\tau}{2} [V(x_i) + V(x_{i+1})] \right) \quad (9)$$

Visually, a classical particle, in PIMD, is represented as P beads connected by harmonic spring, to better describe the quantum effects.

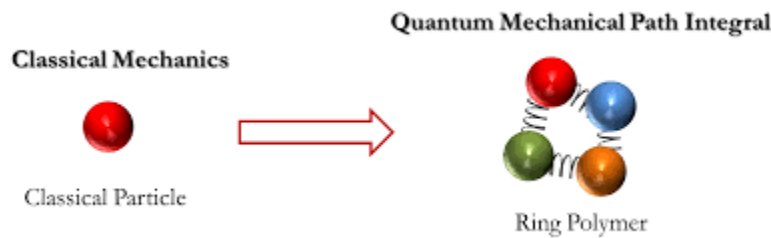


Figure 2. Quantum Mechanical Path Integral Visualization (Lu, 2018)

## Data Lakehouse

A data Lakehouse is a modern data management architecture that combines the key features and benefits of a data lake and a data warehouse into a single platform. It allows organizations to store all types of data, which are structured, semi-structured, and unstructured data, while being able to query different data types and perform data science operations on top. (Google Cloud, 2024) (Databricks, 2024) Data Lakehouse's rely heavily on file metadata to provide data warehouse-like functionality on top of data lake storage. Compare to traditional OS file finding strategy, in a data Lakehouse environment, users can use query such as SQL and NoSQL syntax to directly find data and files. (Snowflake, 2024) Data Lakehouse' architecture determined that every data stored inside, with proper metadata labels, are naturally following the FAIR data principle (Findable, Accessible, Interoperable, Reusable), a critical strategy to manage scientific data. (Wilkinson et al., 2016) Combining data Lakehouse with FAIR principle, researchers will have the foundation to apply data driven research strategies such as active learning, data science, and AI/ML techniques to their workflow.

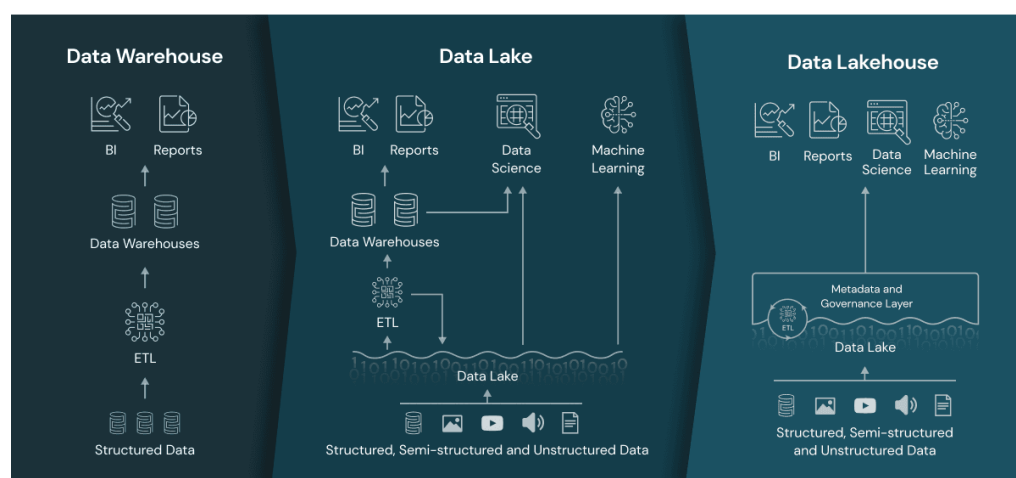


Figure 3. Data lake, Data Warehouses, and Data Lakehouse (Databricks, 2024)



## Workflow Manager

A workflow manager is a system that helps users design, create, execute, and monitor workflows. Its primary role is to streamline and monitor processes by managing the sequence of tasks, resources, and data involved in those processes.

Key features:

- Workflow construction: Users can chain different operations together to form 1-to-many, many-to-1, and 1-to-1 relationships.
- Workflow execution: Automated execution of workflow by mapping tasks to available computational resources, handling task dependencies, and managing data transfers. For example, for molecular dynamic simulations, workflow managers such as AIIDA (Huber et al., 2020) and Fireworks (Jain et al., 2015) can send defined molecular dynamic tasks to local or computational clusters and collect the result when the task is finished.

## LLM Agent

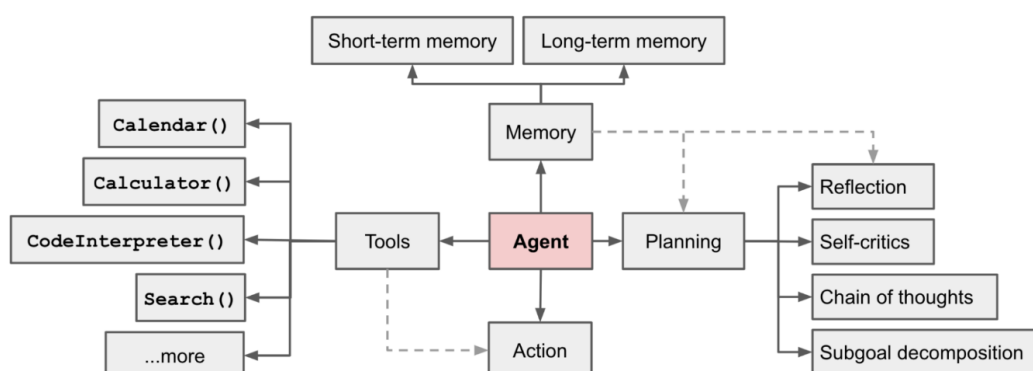


Fig 4. Overview of an LLM-powered autonomous agent system (Lilian, 2023)

An LLM (Large Language Model) agent is an AI system that utilizes a large language model as its core computational engine to exhibit autonomous and intelligent behavior in completing tasks and achieving goals. (Harrison, 2024) (Lilian, 2023) The key components, as shown in Fig 3., typically include:

- Agent Core (LLM): This acts as the brain of agents. It is responsible for understanding the task, making reasonings, and generating responses or even taking action. Usually, models such as OpenAI GPT3.5(ChatGPT), GPT4, and Anthropic Claude are used.

- **Memory Modules:** External memory store that allows agents to keep track of past interactions, observations, and intermediate results. Usually stored as a list of conversational histories.
- **Tools:** LLM agents can interact with a set up of tools or APIs that extend their capabilities beyond just text generation. These tools could include web searches, calculators, and databases. In the LabMind system, relevant tools include the NoSQL MongoDB Database, download and upload information to LabMind Data Lakehouse, and PythonREPL, which allow agents to write and execute Python programs.
- **Planning Module:** This component assists the LLM in breaking down complex tasks into a series of steps or a plan of action.

Compared to traditional software, LLM agents possess advanced reasoning and creativity and are currently under active research and commercialization to assist organizations and individuals in working more efficiently and intelligently. For instance, AbacusAI, a generative AI startup, is developing an applied AI system capable of connecting end-to-end with users' data. This system aids users in training ML models and performing data science operations, similar to the objectives of LabMind. (Abacus AI, 2024)

The objective of the simulation is to acquire accurate ground state energy of water monomers. To run Water PIMD simulations, the OpenMM library is used. (Eastman et al., 2017)

## Simulation Software Architecture

RPPIMD simulation workflow, implemented in Python, is shown in Figure 5.

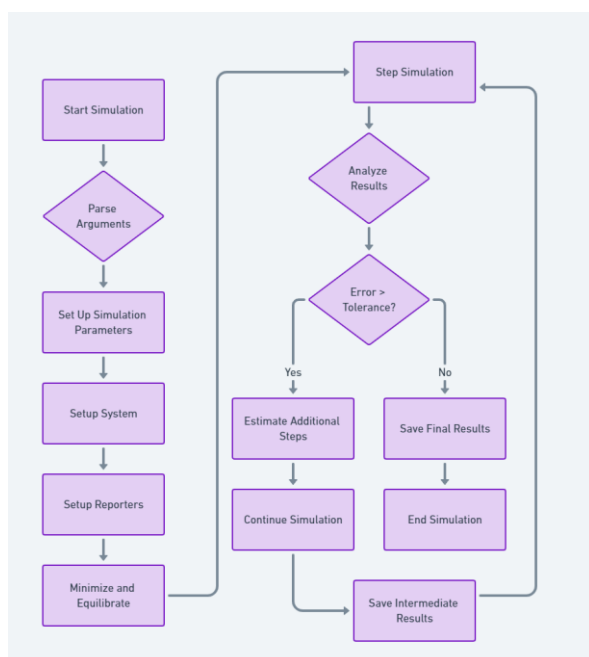


Fig 5. Flow chart of Water PIMD Simulation design

```
import uuid
import os
import argparse
from utils import params_to_json

start_time = time.time()
unique_id = uuid.uuid4()

#### Main code
#steps, equilibration_steps, skip_steps, gamma0, dt, temperature, P, pdb_file, forcefield_file, platform_name
parser = argparse.ArgumentParser(description='Water PIMD simulation')
parser.add_argument('--steps', type=int, default=1000, help='Total number of simulation steps.')
parser.add_argument('--equilibration_steps', type=int, default=100, help='Number of steps for equilibration.')
parser.add_argument('--skip_steps', type=int, default=1, help='Number of steps to skip for data collection.')
parser.add_argument('--gamma0', type=float, default=(1.0 / 0.17), help='Friction coefficient in 1/ps.')
parser.add_argument('--dt', type=float, default=0.12, help='Time step for the simulation in femtoseconds.')
parser.add_argument('--temperature', type=float, default=50.0, help='Simulation temperature in Kelvin.')
parser.add_argument('--P', type=int, default=100, help='Number of beads in the Path Integral formulation.')
args = parser.parse_args()

error_tolerance = 1
# run the simulation, save the results, and analyze the results,
# if the error is greater than 1% of the mean value, estimate the additional steps required to reduce the error to 1% of the mean value
# and run the simulation again
# repeat the process until the error is less than 1% of the mean value

simulation_run = simulation_run_time(uuid=unique_id, ...
simulation_run.run()
sim_analysis = simulation_state(params = simulation_run.metadata, ...
sim_analysis.analyze()
error_percentage = sim_analysis.error_percent

while error_percentage > error_tolerance:
    additional_steps = sim_analysis.estimate_additional_steps()
    print(f"Additional steps required: {additional_steps}")
    simulation_run.continue_simulations(additional_steps = additional_steps)
    simulation_run.save()
    sim_analysis = simulation_state(params = simulation_run.metadata, ...
    sim_analysis.analyze()
    error_percentage = sim_analysis.error_percent

print(f"Total steps taken: {simulation_run.steps}")
sim_analysis.save()

end_time = time.time()
print(f"Total time taken: {end_time - start_time} seconds")
```

Figure 6. Code snippet of main.py for RPPIMD simulation

To facilitate modular design and readability, class `simulation_run_time` and class `simulation_state` are created.

## Class `simulation_run_time`

This class is designed as the workhorse of the simulation process. It sets up the molecular system using input parameters (e.g., temperature, pressure, number of beads in PIMD), selects and applies the appropriate force field, initializes the integrator, and conducts the simulation over the specified number of steps. It also handles the intermediate storage of simulation results, such as potential energy, atomic positions, and forces, which are essential for subsequent analysis. In this way, everything related to the actual simulation is handled by this single class.

### Core Functionalities:

**`simulation_run_time.run`:** Performs Step 3-7 of workflow shown in the Fig 5.

```
def run(self):
    self._setup_system()
    self._setup_reporters()
    self._minimize_and_equilibrate()
    PE, POS, FORCES = self._step_simulation(self.steps)
    self.PE = PE
    self.POS = POS
    self.FORCES = FORCES
```

Figure 7. Code snippet of `simulation_run_time.run`

**`simulation_run_time.continue_simulations`:** This function allows the simulation to be continued from the intermediate state. Allowing us to dynamically check the convergence of the simulation based on error bar criteria.

```
def continue_simulations(self, additional_steps):
    self.steps = self.steps + additional_steps
    self._get_metadata()
    PE, POS, FORCES = self._step_simulation(steps = additional_steps)
    self.PE = np.concatenate((self.PE, PE), axis=0)
    self.POS = np.concatenate((self.POS, POS), axis=0)
    self.FORCES = np.concatenate((self.FORCES, FORCES), axis=0)
```

Fig 8: Code snippet of `simulation_run_time.continue_simulations`

**`simulation_run_time.save`:** This function saves the metadata related to simulations, as well as the simulation results.

```
def save(self):
    self._save_metadata()
    self._save_results()
```

Fig 9: Code snippet of `simulation_run_time.save`

Metadata saved:

- UUID: The unique ID of each simulation runs
- Steps: Total simulation steps
- Equilibration steps: Steps used to equilibrate the system before recording
- Skip steps: Steps skipped during data recording
- Gamma: Friction coefficient inside PIMD
- $\Delta t$ : Time per step in femto seconds
- Temperature: Temperature used in Kelvin

- P: Number of beads
- Input PDB file: Input file name
- Forcefield: Input force field
- Platform Name: Computation resources used
- $\beta$ :  $1/k_bT$
- $\tau$ :  $\beta / P$

Results saved:

- Potential Energy: Potential of each bead at each time step. Saved as NumPy array of shape Time Steps \* Number of Beads
- Positions: Position of each bead at each time step. Saved as NumPy array of shape Time Steps \* Number of Beads\*4\*3
- Force: Force acted on each bead at each time step. Saved as NumPy array of shape Time Steps \* Number of Beads\*4\*3

### Class `simulation_states`

`Simulation_states` focus on analyzing the data generated by PIMD simulation. It utilizes potential energy, positions, forces, and metadata output from the simulation to compute important characterizations such as virial energy, geometry, and error bars.

#### Core Functionalities:

**`simulation_states.analyze`:** Acquire all important properties from the simulation system, including potential energy, virial kinetic energy, virial potential energy, and error bar.

```
def analyze(self):
    self.compute_PE()
    self.compute_K_virial()
    self.compute_H2O_structure()
    self.compute_E_virial()
    self.compute_error_bar_jz()
    self.print_log()
```

Fig 10. Code snippet of `simulation_states.analyze`

For details of each computation, refer to the GitHub link in the conclusion section to access the source code.

## Optimization

Advanced Numpy vectorizations are performed on top of the original code reference which heavily relies on for loop, leading to significant efficiency improvements in results analysis.

K\_virial algorithm:

```
def compute_K_virial_slow(self):
    self.K_virial = np.zeros(self.simulation_steps)
    for i in range(self.simulation_steps):
        K_virial = 0
        for beadi in range(self.P):
            posi = self.POS[i, beadi]
            forces = self.FORCES[i, beadi]
            for j in range(4):
                K_virial -= np.dot(posi[j], forces[j])
            self.K_virial[i] = K_virial
```

Fig 11. The original K\_virial algorithm relies on for loop

```
def compute_K_virial(self):
    # Adjust the einsum path to account for the num_atoms dimension
    # Now, the einsum string indicates:
    # - 'ijkl,ijkl->i' performs dot product across the last dimension (3 for x, y, z components),
    #   sums over all num_atoms and all beads for each step, resulting in one value per step
    # This assumes self.POS.shape and self.FORCES.shape are (steps, beads, num_atoms, 3)
    K_virial_all_steps = -np.einsum('ijkl,ijkl->i', self.POS, self.FORCES)
```

Fig 12. The new K\_virial algorithm relies on Numpy vectorization

Compared code shown in Fig 11 and Fig 12, the original K\_virial algorithm applies a triple layer for loop on position and force data, both of which are in the size of 40000\*(100~2000)\*4\*3. This results in a slow analysis process. The new algorithm, efficiently uses Numpy vectorization, are highly efficient tensor operations.

Tests using dummy samples with the size of 100\*1000\*4\*3 are performed, leading to 36X acceleration.

Old K_virial (s)	New K_virial (s)
9.11	0.25

Table 1. Comparison of K\_virial computation time between old and new algorithms

The same optimization strategy is applied to computing H2O structures to harvest the power of Numpy vectorizations.

```
def compute_H2O_structure_slow(self):
    for i in range(self.simulation_steps):
        for beadi in range(self.P):
            posi = self.POS[i, beadi]
            bead_rOH2=np.linalg.norm(posi[0]-posi[1])
            bead_rOH1=np.linalg.norm(posi[0]-posi[2])
            bead_rHH=np.linalg.norm(posi[2]-posi[1])
            bead_angle=np.arccos(np.dot(posi[0]-posi[2],posi[0]-posi[1])/bead_rOH1/bead_rOH2)*180./np.pi
            self.POS[i, beadi] = np.array([bead_rOH1*10., bead_rOH2*10., bead_rHH*10., bead_angle])
```

Figure 13. Original H2O structure algorithm

```
def compute_H2O_structure(self):
    POS = self.POS
    rOH2 = np.linalg.norm(POS[:, :, 0] - POS[:, :, 1], axis=-1)
    rOH1 = np.linalg.norm(POS[:, :, 0] - POS[:, :, 2], axis=-1)
    rHH = np.linalg.norm(POS[:, :, 2] - POS[:, :, 1], axis=-1)
    vec_OH1 = POS[:, :, 0] - POS[:, :, 2]
    vec_OH2 = POS[:, :, 0] - POS[:, :, 1]
    cos_angle = np.einsum('ijk,ijk->ij', vec_OH1, vec_OH2) / (rOH1 * rOH2)
    angle = np.arccos(np.clip(cos_angle, -1, 1)) * 180. / np.pi
    self.geometry = np.stack((rOH1 * 10., rOH2 * 10., rHH * 10., angle), axis=-1) #shape (simulation_steps, P, 4)
```

Figure 14. New H2O structure algorithm

These optimization strategies lead to over **4X** efficiency boost for the whole simulation pipeline compared to the original ones.

## Force Field Selection

The q-TIP4P/F force field is selected to incorporate accurate modeling of water molecules in low temperatures. (Habershon et al., 2009) It introduces a new simple point charge model for liquid water to better describe O-H stretches, leading to a better description of quantum effects, which path integral molecular dynamic simulation calculates.

## Parameter Selection and Optimization

Parameters selected in the correct range are critical to perform simulations that approximate ground truth. Input parameters used in simulations are

- Steps
- Equilibration steps
- Friction coefficient
- Time per step (dt)
- Temperature
- Beads

## Steps

Number of steps is selected roughly in the range of 10000-20000. According to the workflow in Fig 5, the simulation is run continuously until the error bar is minimized

within the 1% tolerance. Due to this continuous check, simulation done based on input steps only serves as a rough estimation of variance and provides data to predict the number of steps required to reach the tolerated error bar.

## Equilibration Steps

Equilibration steps are identified by plotting the potential v.s steps.

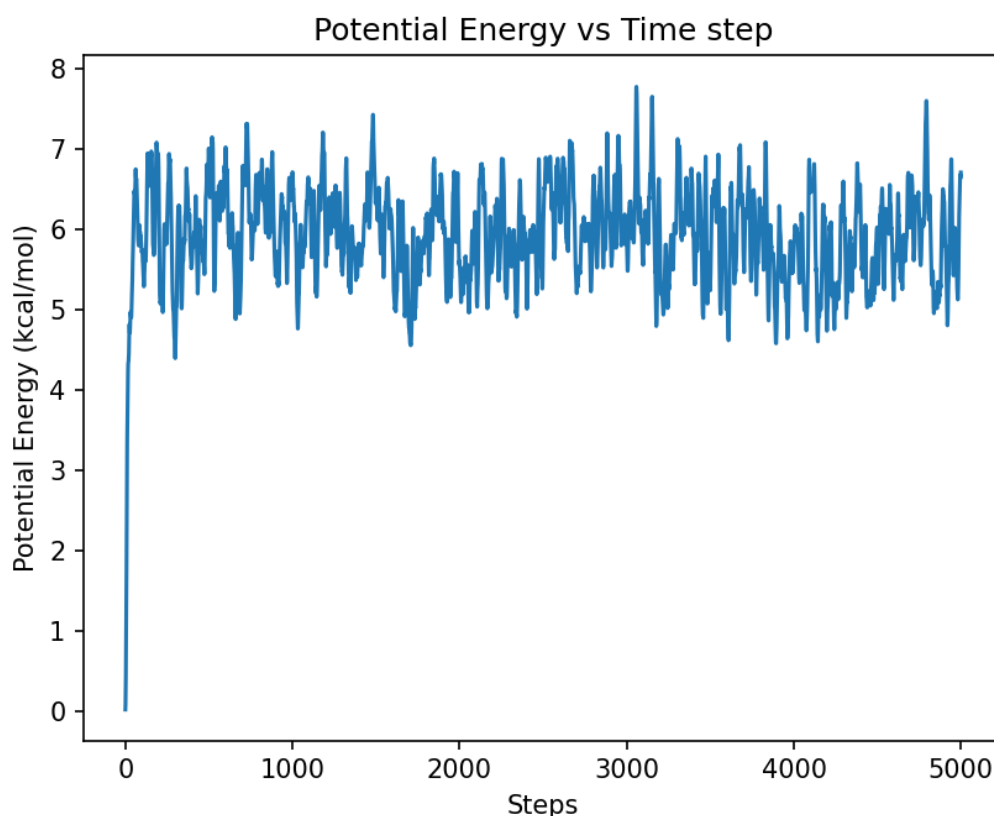


Figure 15. Potential energy vs steps used to determine equilibration steps required

The system equilibrates after ~200 steps. To accommodate different parameter settings, we equilibrate the system for 1000 steps, to further ensure the system equilibrates before data is recorded.

## Friction Coefficient

The friction coefficient is a parameter that couples the system to the heat bath. A high friction coefficient means strong coupling with the heat bath, leading to faster equilibrium but damping particles to explore the phase space. This leads to a larger variance but a smaller decorrelation time. A small friction coefficient means less coupling, which preserves the dynamics better but leads to longer decorrelation time. Decorrelation time and variance are competing parameters that affect error bar minimization. Thus, the friction coefficient is an efficiency parameter.



Tests using different friction coefficients with distinct values are performed on the same simulation settings with

- Beads: 100
- Dt: 0.12 fs
- Temperature: 50K
- Equilibration Steps: 1000

Friction Coefficient (ps <sup>-1</sup> )	Decorrelation Time based on potential (fs)	Total steps to reach 1% error bar	Finished Time (s)	E <sub>virial</sub> (kcal/mol)	Error Bar (kcal/mol)
1/1	1.56	42672	317	11.875	0.119
1/0.17	1.56	45660	326	11.948	0.118
1/0.0017	1.32	43968	309.5	11.813	0.114

Table 2. Study of Friction Coefficients' affect on simulation performance

Potentially due to the small size of the system, the friction coefficient doesn't seem to have huge effects on the speed of convergence. However, we do observe a small difference in E<sub>virial</sub> value.

### Time Per step (dt)

0.12 fs is used. This is determined through a convergence study, reported by Schmidt & Roy (Schmidt & Roy, 2018). As 0.12 fs per step yields good E<sub>virial</sub> estimation. The study in this report didn't optimize it further.

### Temperature

As the objective is investigating the ground state of water monomer, 20K, 30K, 40K, and 50K are selected to investigate convergence. As convergence was observed, lower temperatures were not selected.

### Beads

The Trotter error, a deterministic error term associated with path integral molecular dynamics (PIMD), as mentioned in the Knowledge Background section, is described as polynomial in the form shown in (10). To determine E<sub>0</sub>, the ground state energy, it is necessary to simulate multiple E(τ) values and perform polynomial fitting (polyfit). The smallest τ considered is 0.005. At 20K, a τ of 0.05 corresponds to P=1202. The largest τ is chosen with a fixed P = 100 across all temperatures.

$$E(\tau) = E_0 + b\tau^2 + c\tau^4 \quad (10)$$

## Error Bar minimization

$$Error = Var(Energy) / \sqrt{N_{\text{independent}}}$$

$$N_{\text{independent}} = steps / ceiling(T_{\text{decorrelation}} / \delta t) \quad (11)$$

Based on (11), to minimize the error bar within 1% of the final average, we prefer small decorrelation time, small dt, and small variance. Decorrelation time is auto-computed within the pipeline before it is used to determine the error bar.

Decorrelation time is computed with the function shown in Figure 16. It is quantified as 1/e of the autocorrelation series.

```
def full_autocorrelation(series, show: bool = False):
    """
    Compute the autocorrelation of the specified series for all possible lags.
    This should be the correct one

    :param series: The time series data.
    :return: A numpy array containing the autocorrelation values for all lags.
    """
    #series = np.mean(series, axis=1)
    N = len(series)
    # Subtract the mean from the series
    series_mean_subtracted = series - np.mean(series)

    # Calculate the autocorrelation using the numpy correlate function
    autocorrelations = np.correlate(series_mean_subtracted, series_mean_subtracted, mode='full')[N-1:] / N
    plt.plot(autocorrelations)
    plt.xlabel("Lag")
    plt.ylabel("Autocorrelation")
    plt.title("Autocorrelation Function of Potential Energy")
    if show:
        plt.show()
    return autocorrelations
```

```
def calculate_correlation_time(series):
    """
    Compute the correlation time for the given time series data.

    :param series: The time series data.
    :return: The correlation time, tau_c.
    """
    # Subtract the mean from the series to get δA(t)
    series = np.mean(series, axis=1)

    # Calculate the full autocorrelation function
    autocorrelations = full_autocorrelation(series, show = True)

    # Sum the autocorrelation function values to estimate the integral
    tau_c = find_nearest_half_life(autocorrelations)

    return tau_c
```

Fig 16. Autocorrelation function and decorrelation time calculation code snippet

# LabMind Orchestrator Design

LabMind orchestrator is designed to be generally purposed to orchestrate any workflow such as experiments, simulations, post-processing, and visualizations. Utilizing MongoDB, a NoSQL database, with cloud storage solutions, such as Google Drive, LabMind constructs a data Lakehouse architecture with the easiest accessible setup to allow any researchers to deploy it into their research. What's more, LabMind facilitates the use of AI agents. Users can easily connect AI agents to the central data Lakehouse to create a customizable copilot experience in workflow creation and data analysis.

## Data entity

Although R&D data is diverse, it can generally be categorized into two types: files and knowledge. 'Files' refers to data associated with a digital file, whereas 'knowledge' can be considered as data such as a number, string, or other items not directly linked to a digital file (akin to a row in an Excel spreadsheet). Therefore, the two fundamental data entities within LabMind are known as FileObject and KnowledgeObject.

## FileObject

A **FileObject** represents a digital file within the LabMind system, encapsulating all relevant details about the file, including its location, metadata, and associated cloud and NoSQL services. It serves multiple purposes:

- **Initialization:** Upon instantiation, a FileObject is assigned a local file path, metadata, cloud service, and NoSQL service identifiers. It optionally includes an embedding for vector search purposes.
- **Metadata Validation and Embedding:** The object ensures that all necessary metadata is present and correct. It also prepares metadata for embedding, facilitating efficient search and retrieval operations.
- **Cloud Interaction:** It includes methods for uploading the file to cloud storage, ensuring the existence of appropriate directories, and managing file metadata in cloud services.
- **Data Processing (ELT):** It supports Extract, Load, Transform (ELT) operations to prepare the data for analysis or storage, including executing specified ELT functions.
- **Metadata Documentation and Management:** The object can document metadata in a MongoDB collection and manage file metadata within specified collections based on category (Raw/Processed/Final).

## KnowledgeObject

A **KnowledgeObject**, while similar in purpose to a FileObject, focuses on encapsulating and managing pieces of knowledge within the system. It also includes a metadata component, a NoSQL service identifier, and an optional embedding. Key functionalities include:

- **Initialization:** It initializes with given metadata and NoSQL service details, preparing itself for further processing.
- **Metadata Validation and Processing:** Similar to FileObject, it validates required metadata fields and prepares metadata for embedding and efficient retrieval.
- **Knowledge and Metadata Management:** The object documents new knowledge entries in the MongoDB collection, saves knowledge metadata to its respective collection, and performs ELT if specified, enabling the processing of knowledge through predefined functions.

With those two data entities, LabMind can easily handle storage and perform operations on structured, semi-structured, and unstructured data.

## Automation

Actions can be chained together to create automation inside LabMind systems. Consider a simple workflow in Fig 17, starting with a number as a 'KnowledgeObject' in LabMind in Fig 18, with triggers connected to the 'add1' function in Fig 19, with termination of the workflow by removing the ELT field with condition, in this case,  $\text{new\_number} > 10$ . Leading to results stored in MongoDB shown in Fig 20.

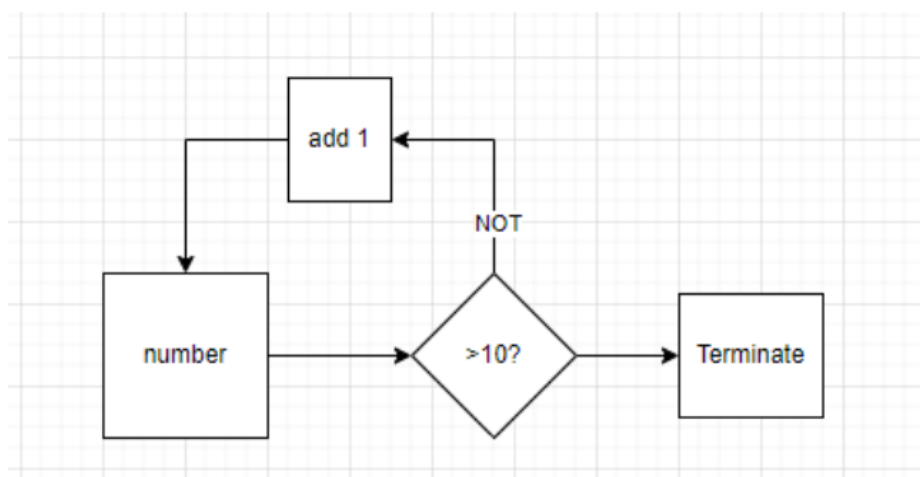


Figure 17. Workflow of number addition with condition

```
knowledge = {
    "db": "Tutorial",
    "collection": "knowledges",
    "number": 1,
    "unique_fields": ["number"],
    "ELT": ["add1"],
}
knowledge = KnowledgeObject(knowledge, nosql_service)
knowledge.parse()
```

Fig 18. Code Snippet of workflow defined in Fig 16

```
from LabMind.Infrastructure.data_objects.data_objects import KnowledgeObject
from LabMind.Infrastructure.client_config import nosql_service

def add1(previous_number: KnowledgeObject):
    number = previous_number.metadata["number"]
    new_number = number + 1
    metadata = previous_number.metadata
    metadata["number"] = new_number
    #pop incase Object_id is not tolerated with json serialization
    metadata.pop('_id')
    if new_number > 10:
        metadata.pop("ELT")
    result = KnowledgeObject(metadata, nosql_service)
    return [result]
```

Fig 19. Add1 function

```

_id: ObjectId('660ad59a5cbad50dde45e283')
db: "Tutorial"
collection: "knowledges"
number: 1
▶ unique_fields: Array (1)
▶ ELT: Array (1)
▶ embedding: Array (1536)

_id: ObjectId('660ad59c5cbad50dde45e284')
db: "Tutorial"
collection: "knowledges"
number: 2
▶ unique_fields: Array (1)
▶ ELT: Array (1)
▶ embedding: Array (1536)

_id: ObjectId('660ad59e5cbad50dde45e285')
db: "Tutorial"
collection: "knowledges"
number: 3
▶ unique_fields: Array (1)
```

Figure 20. Results stored in MongoDB

## Data Science

Data science can be done by converting NoSQL database collections into a data frame using PyMongo and Pandas libraries, as shown in Fig 21. Users can easily

perform data science operations including transformation and visualization with the dataframe.

```
collection = nosql_service["Tutorial"]["knowledges"]
#convert to df and print with only field of db, collection, number
import pandas as pd
df = pd.DataFrame(list(collection.find({},{"db":1,"collection":1,"number":1})))
print(df)
```

_id	db	collection	number
660ad4f45cbad50dde45e278	Tutorial	knowledges	NaN
660ad59a5cbad50dde45e283	Tutorial	knowledges	1.0
660ad59c5cbad50dde45e284	Tutorial	knowledges	2.0
660ad59e5cbad50dde45e285	Tutorial	knowledges	3.0
660ad59f5cbad50dde45e286	Tutorial	knowledges	4.0
660ad5a15cbad50dde45e287	Tutorial	knowledges	5.0
660ad5a25cbad50dde45e288	Tutorial	knowledges	6.0
660ad5a45cbad50dde45e289	Tutorial	knowledges	7.0
660ad5a65cbad50dde45e28a	Tutorial	knowledges	8.0
660ad5a75cbad50dde45e28b	Tutorial	knowledges	9.0
660ad5a95cbad50dde45e28c	Tutorial	knowledges	10.0

Figure 21. Conversion of NoSQL Collection to Dataframe

## Construct LLM Agent and copilots

Users can create LLM agents that can use specific tools and are well connected to the LabMind database to perform efficient data operations and manipulations using natural language.

As an example shown in Figure 22, users can create a MongoDBAgent with one line of code. The agent can freely interact with the database by writing PyMongo code, a Python API to MongoDB, and the code is executed automatically through a modified PythonREPL tool, which the result is fed back to the agent as a response to help the agent answer the users prompt as shown in Figure 23. Other operations such as acquiring information about documents within collections, and converting collections to CSV are shown in Figure 24 and Figure 25 respectively.

```
from LabMind.Agent.agent import LangChainMongodbOperator
agent = LangChainMongodbOperator(model = "gpt-3.5-turbo-0125")
```

Figure 22. Creating an Mongodb Agent with one line of code

```
agent.invoke("can you list all the dbs and all the collections associated with the dbs inside this client")
```

```

> Entering new AgentExecutor chain...

Invoking: `PyMongo_REPL` with `print(client.list_database_names())`

['NE452_A3', 'NE452_Simulation', 'NE459_Final_Fixed', 'Tutorial', 'chat_history', 'admin', 'Local']

Invoking: `PyMongo_REPL` with `print(client['NE452_A3'].list_collection_names())`

['processed', 'Vanalysis', 'MetaData', 'raw']

Invoking: `PyMongo_REPL` with `print(client['NE452_Simulation'].list_collection_names())`

['demo', 'Assignment2', 'raw', 'MetaData', 'processed']

Invoking: `PyMongo_REPL` with `print(client['NE459_Final_Fixed'].list_collection_names())`

```

```

['message_store']
Here are the databases and collections associated with each database:

1. Database: NE452_A3
   - Collections: processed, Vanalysis, MetaData, raw

2. Database: NE452_Simulation
   - Collections: demo, Assignment2, raw, MetaData, processed

3. Database: NE459_Final_Fixed
   - Collections: E_virial, raw, MetaData

4. Database: Tutorial
   - Collections: MetaData, knowledges, raw

5. Database: chat_history
   - Collections: message_store

```

Figure 23. Images of agent operation and results. The image on the top: User's prompt for the agent. Image in the middle: The agent took a series of actions, including printing all dbs inside the client, and then print collections of each dbs one by one to acquire all the collections. Image at the bottom: Agent's response to the question.

```
agent.invoke("what are the fields inside NE459_Final_Fixed E_virial collection,")
```

```

Invoking: `PyMongo_REPL` with `for doc in client['NE459_Final_Fixed']['E_virial'].find().Limit(1): print(doc.keys())`

dict_keys(['_id', 'E_virial', 'Error_bar', 'db', 'collection', 'creator', 'description', 'date', 'uuid', 'params', 'unique_fields', 'default', 'ELT', 'embedding'])
The fields inside the `E_virial` collection of the `NE459_Final_Fixed` database are as follows:

- `_id`
- `E_virial`
- `Error_bar`
- `db`
- `collection`
- `creator`
- `description`
- `date`
- `uuid`
- `params`
- `unique_fields`
- `default`
- `ELT`
- `embedding`

> Finished chain.

```

Figure 24. Images of agent operation and results. The image on the top: User's prompt to the agent. Image at the bottom: The agent prints one document inside a collection, acquires the keys (fields) of this document, and uses it to answer the user's question.

```

agent.invoke("save NE459_Final_Fixed E_virial collection as E_virial.csv to current dir,make sure all nested fields are flattened")

> Entering new AgentExecutor chain...

Invoking: `PyMongo_REPL` with `import pandas as pd
from pandas.io.json import json_normalize

db = client['NE459_Final_Fixed']
collection = db['E_virial']

df = pd.DataFrame(List(collection.find()))
df_flattened = json_normalize(df.to_dict('records'))
df_flattened.to_csv('E_virial.csv', index=False)`

CODE EXECUTION FAILED. Your query must include a print() statement inside code to provide useful feedback. print either results or feed
Invoking: `PyMongo_REPL` with `import pandas as pd
from pandas.io.json import json_normalize

db = client['NE459_Final_Fixed']
collection = db['E_virial']

df = pd.DataFrame(List(collection.find()))

df = pd.DataFrame(List(collection.find()))
df_flattened = json_normalize(df.to_dict('records'))
df_flattened.to_csv('E_virial.csv', index=False)
print('E_virial collection has been saved as E_virial.csv with all nested fields flattened.')`

ImportError("cannot import name 'json_normalize' from 'pandas.io.json' (c:\\Users\\Lenovo\\anaconda3\\envs\\LMTest\\Lib\\site-packages
Invoking: `PyMongo_REPL` with `import pandas as pd

db = client['NE459_Final_Fixed']
collection = db['E_virial']

df = pd.DataFrame(List(collection.find()))
flattened_data = []
for index, row in df.iterrows():
    flattened_row = pd.json_normalize(row.to_dict(), max_level=1)
    flattened_data.append(flattened_row)
flattened_df = pd.concat(flattened_data, ignore_index=True)
flattened_df.to_csv('E_virial.csv', index=False)
print('E_virial collection has been saved as E_virial.csv with all nested fields flattened.')`
responded: It seems that there was an issue with importing the `json_normalize` function from `pandas.io.json`. Let me try a different
E_virial collection has been saved as E_virial.csv with all nested fields flattened.
The E_virial collection from the NE459_Final_Fixed database has been saved as E_virial.csv in the current directory. ALL nested fields

> Finished chain.

```

Figure 25. Images of agent operation and results. The image on the top: User's prompt to the agent. Image at the bottom: The agent prints one document inside a collection, acquire the keys (fields) of this document, and uses it to answer the user's question.



## Results

Our method utilizes the ring polymer path integral molecular dynamics (RPPIMD) technique to simulate a water monomer at temperatures of 20K, 30K, 40K, and 50K. For each temperature, 10 simulations at different imaginary time steps  $\tau$  were conducted to accurately fit Trotter error parameters. This process ensures that the ground state energy,  $E_0$ , is determined with an error tolerance within 1%. Simulation parameters were chosen based on the 'Parameter Selection and Optimization' section of this report. All 40 simulations were managed by the LabMind system, with results including virial energy, error bars, and geometry data being automatically uploaded and processed. Data is directly extracted from LabMind for polynomial fitting, and the result, generated by a coding agent with direct database access, is illustrated in Figure 26. The  $E_0$  values obtained are presented in Table 3. PIMD results, with  $E_0 = 13.2$  kcal/mol, are consistent with previously developed methods such as LePIGS, which also reported 13.2 kcal/mol (Schmidt & Roy, 2018), and the DMC method at 13.18 kcal/mol (Mallory et al., 2015). These comparisons demonstrate that RPPIMD can calculate energy both accurately and reliably.

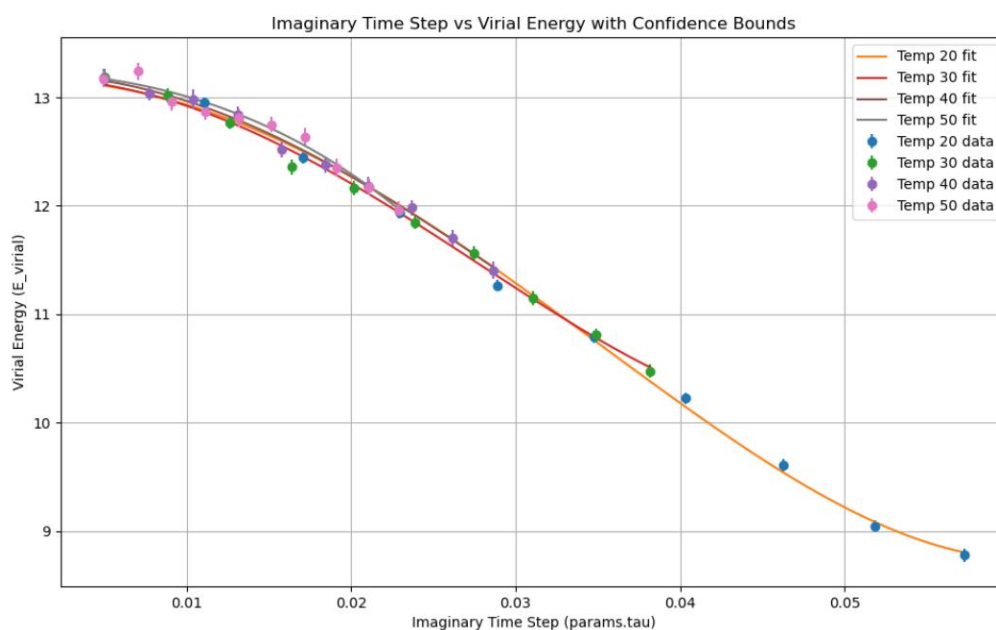


Figure 26. Imaginary Time Step vs Virial Energy with confidence bound

Temperature (K)	$E_0$ (Kcal/mol)	b	c
20	13.1619	-2368.15	316270.71
30	13.1868	-2690.17	586708.20
40	13.2237	-2574.87	477283.23

50	13.2254	-2116.11	-532638.21
----	---------	----------	------------

Table 3. Trotter parameters and ground state energy at different temperatures

## Conclusions

In conclusion, this report has demonstrated that the Ring Polymer Path Integral Molecular Dynamics (RPPIMD) method is a reliable technique for simulating quantum significant systems under cryogenic conditions, such as the water monomer. The ground state energy  $E_0$  was accurately determined and aligns with established benchmarks, corroborating previous findings. The RPPIMD pipeline has been effectively implemented and optimized for algorithm performance and parameter selection. This encompasses the OpenMM RPPIMD simulation, energy calculations, geometry calculations, and decorrelation time calculations. The code is publicly available on GitHub at: <https://github.com/yunhzou/WaterPIMD>. Additionally, the report highlights the capabilities of LabMind, a sophisticated orchestrator that manages simulation processes and handles data input and output automatically via a cloud platform. LabMind also introduces an agentic workflow, enabling researchers to perform data transformation and post-analysis using intuitive natural language commands.

## Recommendations

Further investigations could extend the application of RPPIMD to more complex molecular systems, with promising candidates such as water dimers and Deuterium oxide. A diverse array of forcefields, including those based on ab initio data like MB-pol (Schmidt & Roy, 2018), could be utilized to further validate the system.


LabMind holds a promising future in aiding researchers to simplify their research workflows through enhanced data management, automation, and copiloting capabilities. Although currently verified with OpenMM simulations, the LabMind workflow could be expanded to integrate other simulation toolkits such as LAMMPS, Gaussian, and AIIDA. Additionally, High Performance Computing (HPC) could be seamlessly integrated as a user-friendly feature, facilitating access to local computing clusters and HPC cloud providers such as Compute Canada, Azure, and AWS.

Moreover, LabMind could evolve to orchestrate bench experiments as more robotic and automated equipment enters academia and industry R&D facilities. The

integration of semi-automatic and automatic experimental workflows could fit seamlessly into current infrastructures, enhancing control and efficiency.

Lastly, LabMind's rich metadata and multimodal data capabilities could enable faster connections to agent and SaaS solutions than traditional data management systems. For instance, agents equipped with Bayesian Optimization toolkits could sift through a reaction dataset stored in LabMind, automatically applying BO algorithms to provide recommendations for subsequent experiments. This integration of advanced tools could significantly streamline researchers' workflows and accelerate their decision-making processes.

## References

- Abacus AI. (2024). *The world's first AI assisted end-to-end data science and MLOps platform*. Abacus.AI. <https://abacus.ai/>
- Databricks. (2024). *What is a Data Lakehouse?* <https://www.databricks.com/glossary/data-lakehouse>
- Eastman, P., Swails, J., Chodera, J. D., McGibbon, R. T., Zhao, Y., Beauchamp, K. A., Wang, L.-P., Simmonett, A. C., Harrigan, M. P., Stern, C. D., Wiewiora, R. P., Brooks, B. R., & Pande, V. S. (2017). OpenMM 7: Rapid development of high performance algorithms for Molecular Dynamics. *PLOS Computational Biology*, 13(7). <https://doi.org/10.1371/journal.pcbi.1005659>
- Google Cloud. (2024). *What is a Data Lakehouse, and how does it work?* | *google cloud*. Google. <https://cloud.google.com/discover/what-is-a-data-lakehouse>
- Habershon, S., Markland, T. E., & Manolopoulos, D. E. (2009). Competing quantum effects in the dynamics of a flexible water model. *The Journal of Chemical Physics*, 131(2). <https://doi.org/10.1063/1.3167790>
- Harrison, C. (2024). *Agents*.  LangChain. <https://python.langchain.com/docs/modules/agents/>
- Huber, S. P., Zoupanos, S., Uhrin, M., Talirz, L., Kahle, L., Häuselmann, R., Gresch, D., Müller, T., Yakutovich, A. V., Andersen, C. W., Ramirez, F. F., Adorf, C. S., Gargiulo, F., Kumbhar, S., Passaro, E., Johnston, C., Merkys, A., Cepellotti, A., Mounet, N., ... Pizzi, G. (2020). AIIDA 1.0, a scalable computational infrastructure for automated reproducible workflows and Data Provenance. *Scientific Data*, 7(1). <https://doi.org/10.1038/s41597-020-00638-4>
- Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G., Hautier, G., Gunter, D., & Persson, K. A. (2015). Fireworks: A dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17), 5037–5059. <https://doi.org/10.1002/cpe.3505>
- Lu, J. (2018). Video: Jianfeng Lu, “path integral molecular dynamics.” path integral molecular dynamics. <http://www.birs.ca/events/2018/5-day-workshops/18w5023/videos/watch/201811140903-Lu.html>

- Lilian, W. (2023a). *Overview of a LLM-powered autonomous agent system*. LLM Powered Autonomous Agents. Github. Retrieved April 6, 2024, from <https://lilianweng.github.io/posts/2023-06-23-agent/>.
- Lilian, W. (2023b). *Overview of a LLM-powered autonomous agent system*. LLM Powered Autonomous Agents. Github. Retrieved April 6, 2024, from <https://lilianweng.github.io/posts/2023-06-23-agent/>.
- Mallory, J. D., Brown, S. E., & Mandelshtam, V. A. (2015). Assessing the performance of the diffusion monte carlo method as applied to the water monomer, dimer, and hexamer. *The Journal of Physical Chemistry A*, 119(24), 6504–6515. <https://doi.org/10.1021/acs.jpca.5b02511>
- Ormeño, F., & General, I. J. (2024). Convergence and equilibrium in molecular dynamics simulations. *Communications Chemistry*, 7(1). <https://doi.org/10.1038/s42004-024-01114-5>
- Pandas Developer. (2024). *Pandas.dataframe#*. pandas.DataFrame - pandas 2.2.1 documentation. <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>
- Schmidt, M., & Roy, P.-N. (2018). Path integral molecular dynamic simulation of flexible molecular systems in their ground state: Application to the water dimer. *The Journal of Chemical Physics*, 148(12). <https://doi.org/10.1063/1.5017532>
- Snowflake. (2024). *What is a Data Lakehouse?* <https://www.snowflake.com/guides/what-data-lakehouse/>
- Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., ... Mons, B. (2016). The Fair Guiding Principles for Scientific Data Management and Stewardship. *Scientific Data*, 3(1). <https://doi.org/10.1038/sdata.2016.18>