《人工智能背景下的计算化学实验改革与实践》

补充信息

周 佳** 潘永龙 何思斯

（哈尔滨工业大学（深圳）城市水资源与水环境国家重点实验室 理学院，广东深圳 518055）

本补充信息作为论文《人工智能背景下的计算化学实验改革与实践》的必要附件，完整呈现了所有支撑文中所述实验与分析的 Python 源代码。旨在确保研究成果的高可复现性与透明度，并为读者提供深入理解本研究方法学、算法实现及数据处理流程的具体细节。特别是，本代码库详细展示了人工智能技术如何与计算化学实验的各个环节（从数据生成到模型构建与应用）无缝集成，体现了论文所探讨的实验改革与实践的精髓。代码结构上，它被设计为一系列逻辑清晰、功能独立的模块，涵盖了环境配置、量子化学计算接口、AI 模型训练与预测、数据处理与可视化等关键组成部分。每部分代码均附有必要的注释和文档说明，便于读者理解、修改和进一步开发利用。

Table S1 分子结构优化

from pyscf import gto, scf	# 导入 PySCF 库中的 gto（分子结构）和 scf（自洽场计算）模块
mol = gto.M(atom='O 0 0 0; H 0 1 0; H 0 0 1', basis='ccpvdz')	# 创建一个分子对象，定义分子的几何构型和基组 # 这里定义了一个水分子（O-H-H），氧原子在原点，两个氢原子分别位于 y 轴和 z 轴上 # 使用 cc-pVDZ 基组
mf = scf.RHF(mol)	# 创建 RHF（限制性 Hartree-Fock）计算对象，用于进行自洽场计算
from pyscf.geomopt.geometric_solver import optimize	# 导入几何优化模块中的几何优化求解器
mol_eq = optimize(mf, maxsteps=100)	# 使用几何优化求解器对分子进行几何优化 # maxsteps 参数指定最大优化步数为 100
print(mol_eq.tostring())生成优化能量和步骤的图	# 打印优化后的分子几何构型
mol_eq.tofile('optimized_structure.xyz')	# 将优化后的分子结构保存为 XYZ 格式文件

Table S2 分子轨道

from pyscf import gto, scf	# 导入 PySCF 库中的 gto（分子结构）和 scf（自

*哈尔滨工业大学深圳校区质量工程项目(高等教育教学改革项目)（项目编号：HITSZERP22009）

*哈尔滨工业大学深圳校区“AI+专业”课程建设

**周佳，Email: jiazhou@hit.edu.cn

	洽场计算）模块
from pyscf.tools import cubegen	# 从 pyscf.tools 中导入 cubegen 模块，用于生成立方体文件（用于可视化分子轨道等信息）
mol=gto.M(atom='optimized_structure.xyz', basis='ccpvdz')	# 定义分子，从优化后的 XYZ 文件中读取分子结构
mf = scf.RHF(mol).run()	# 运行 RHF（限制性 Hartree-Fock）计算
homo_index = sum(mf.mo_occ > 0) - 1 lumo_index = homo_index + 1	# HOMO 是最后一个被占据的轨道 # LUMO 是第一个未被占据的轨道
cubegen.orbital(mol, 'homo.cube', mf.mo_coeff[:, homo_index])	# 生成 HOMO 的立方格点文件
cubegen.orbital(mol, 'lumo.cube', mf.mo_coeff[:, lumo_index])	# 生成 LUMO 的立方格点文件
print(f"HOMO index: {homo_index}") print(f"LUMO index: {lumo_index}")	# 打印最高占据分子轨道（HOMO）的索引 # 打印最低未占据分子轨道（LUMO）的索引

Table S3 势能面

import numpy as np import matplotlib.pyplot as plt	# 导入 NumPy 库，并将其命名为 np。 # 导入 Matplotlib 库中的 pyplot 模块，并将其命名为 plt。
from pyscf import gto, scf	# 从 PySCF 库中导入 gto 模块(用于定义分子结构)和 scf 模块（用于进行自洽场计算）
def scan_oh_bond_lengths():	# 定义一个名为 scan_oh_bond_lengths 的函数。
angle = 104.5 angle_rad = np.deg2rad(angle)	# 固定 H-O-H 键角为 104.5 度（水的键角）并转换为弧度
bond_lengths_oh1 = np.linspace(0.8, 1.2, 10) bond_lengths_oh2 = np.linspace(0.8, 1.2, 10)	# 设置两个 O-H 键长的范围，单位为 Å
energies=np.zeros((len(bond_lengths_oh1), len(bond_lengths_oh2))) oh1_values = [ ] oh2_values = [ ]	# 存储能量和键长
for i, bl1 in enumerate(bond_lengths_oh1): for j, bl2 in enumerate(bond_lengths_oh2):	# 遍历每个 O-H 键长组合
h1_x = 0.0 h1_y = 0.0	# 根据键长和键角计算 H 原子的坐标

h1_z = bl1	# 氧原子固定在原点 (0, 0, 0) # 第一个 H 原子在 z 轴上
h2_x = bl2 * np.sin(angle_rad / 2) h2_y = 0.0 h2_z = -bl2 * np.cos(angle_rad / 2)	# 第二个 H 原子在 xz 平面内，与 z 轴的夹角为 104.5 度
mol = gto.M(atom=f'O 0 0 0; H {h1_x} {h1_y} {h1_z}; H {h2_x} {h2_y} {h2_z}', basis='ccpvdz')	# 构建分子模型
mf = scf.RHF(mol).run() energies[i, j] = mf.e_tot	# 使用自洽场方法 (RHF) 计算能量
oh1_values.append(bl1) oh2_values.append(bl2)	# 存储能量
With open("potential_energy_surface_2d.txt", "w") as f: f.write("O-H1(Å) O-H2(Å) Energy(Hartree)\n") for bl1, bl2, e in zip(oh1_values, oh2_values, energies.flatten()): f.write(f"{bl1:.3f} {bl2:.3f} {e:.8f}\n")	# 保存键长和对应能量到文本文件
X, Y = np.meshgrid(bond_lengths_oh1, bond_lengths_oh2) plt.contourf(X, Y, energies, levels=50, cmap='viridis') plt.colorbar(label='Energy (Hartree)') plt.xlabel("O-H1 bond length (Å)") plt.ylabel("O-H2 bond length (Å)") plt.title("2D Potential Energy Surface for O-H Bond Lengths") plt.show()	# 绘制二维能量面
scan_oh_bond_lengths()	# 执行扫描函数

Table S4 机器学习

import numpy as np import pandas as pd import matplotlib.pyplot as plt from sklearn.model_selection import train_test_split from sklearn.preprocessing import StandardScaler from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense from tensorflow.keras.optimizers import Adam	# 导入 NumPy 库，并将其命名为 np。 # 导入 Pandas 库，并将其命名为 pd。 # 导入 Matplotlib 库中的 pyplot 模块，并将其命名为 plt。 # 从 Scikit-learn 库中导入 train_test_split 函数。 # 从 TensorFlow 库中的 Keras 模块导入 Sequential 类。 # 从 Keras 库中导入 Dense 层。 # 从 Keras 库中导入 Adam 优化器。
Z_O = 8 Z_H = 1	# 氧原子序数 # 氢原子序数

def calculate_coulomb_matrix(bond_length_oh1, bond_length_oh2, angle_degrees):	# 计算库伦矩阵
angle_rad = np.deg2rad(angle_degrees)	# 将键角转换为弧度
h1_x = 0.0 h1_y = 0.0 h1_z = bond_length_oh1	# 计算原子坐标 # 氧原子固定在原点 (0, 0, 0) # 第一个氢原子在 z 轴上
h2_x = bond_length_oh2 * np.sin(angle_rad / 2) h2_y = 0.0 h2_z = -bond_length_oh2 * np.cos(angle_rad / 2)	# 第二个氢原子在 xz 平面内，与 z 轴的夹角为 angle_degrees
R_O = np.array([0.0, 0.0, 0.0]) R_H1 = np.array([h1_x, h1_y, h1_z]) R_H2 = np.array([h2_x, h2_y, h2_z])	# 原子坐标
dist_oh1 = np.linalg.norm(R_O - R_H1) dist_oh2 = np.linalg.norm(R_O - R_H2) dist_h1h2 = np.linalg.norm(R_H1 - R_H2)	# 计算原子间距离
M = np.zeros((3, 3)) M[0, 0] = 0.5 * Z_O ** 2.4 M[1, 1] = 0.5 * Z_H ** 2.4 M[2, 2] = 0.5 * Z_H ** 2.4	# 构建库伦矩阵 # 氧原子的对角元素 # 第一个氢原子的对角元素 # 第二个氢原子的对角元素
M[0, 1] = Z_O * Z_H / dist_oh1 M[1, 0] = M[0, 1]	# 氧和第一个氢的相互作用
M[0, 2] = Z_O * Z_H / dist_oh2 M[2, 0] = M[0, 2]	# 氧和第二个氢的相互作用
M[1, 2] = Z_H * Z_H / dist_h1h2 M[2, 1] = M[1, 2]	# 两个氢的相互作用
return M	# 从函数中返回一个变量 M 的值
try: data= pd.read_csv("potential_energy_surface_2d.txt", sep='\s+') print("文件读取成功！") print("列名:", data.columns) except Exception as e: print("文件读取失败，请检查文件路径或格式。错误信息：", e) exit()	# 加载数据
required_columns = ["O-H1(Å)", "O-H2(Å)", "Energy(Hartree)"] if not all(col in data.columns for col in required_columns): print("列名不匹配，请检查文件列名。") print("当前列名:", data.columns) print("需要的列名:", required_columns)	# 检查列名是否匹配

exit()	
X = [] for _, row in data.iterrows(): bond_length_oh1 = row["O-H1(Å)"] bond_length_oh2 = row["O-H2(Å)"] M = calculate_coulomb_matrix(bond_length_oh1, bond_length_oh2, angle_degrees=104.5) X.append(M.flatten()) X = np.array(X)	# 计算库伦矩阵并展平为特征向量
y = data["Energy(Hartree)"].values	# 提取能量
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)	# 将数据集分为训练集和测试集
scaler = StandardScaler() X_train = scaler.fit_transform(X_train) X_test = scaler.transform(X_test)	# 标准化输入特征
model = Sequential([Dense(64, activation='relu', input_shape=(9,)) Dense(64, activation='relu'), Dense(64, activation='relu'), Dense(1)])	# 构建神经网络模型 # 输入层，9 个特征（3x3 库伦矩阵展平） # 隐藏层 # 隐藏层 # 输出层，1 个输出（能量）
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')	# 编译模型
history = model.fit(X_train, y_train, epochs=200, batch_size=32, validation_split=0.2)	# 训练模型
test_loss = model.evaluate(X_test, y_test) print(f"Test Loss (MSE): {test_loss}")	# 评估模型

x1 = np.linspace(0.8, 1.2, 100) # O-H1 键长范围
x2 = np.linspace(0.8, 1.2, 100) # O-H2 键长范围
X1, X2 = np.meshgrid(x1, x2)
X_grid = []
for bl1, bl2 in zip(X1.ravel(), X2.ravel()):
M = calculate_coulomb_matrix(bl1, bl2, angle_degrees=104.5)
X_grid.append(M.flatten())
X_grid = np.array(X_grid)
X_grid_scaled = scaler.transform(X_grid)
y_pred=model.predict(X_grid_scaled).reshape(X1.shape)
	# 使用模型预测势能面

plt.figure(figsize=(10, 8)) plt.contourf(X1, X2, y_pred, levels=50, cmap='viridis') plt.colorbar(label='Predicted Energy (Hartree)')	# 绘制势能面

plt.xlabel("O-H1 bond length (Å)") plt.ylabel("O-H2 bond length (Å)") plt.title("Neural Network Fitted Potential Energy Surface (Coulomb Matrix Descriptor)") plt.show()	
plt.figure(figsize=(8, 6)) plt.plot(history.history['loss'], label='Training Loss') plt.plot(history.history['val_loss'], label='Validation Loss') plt.xlabel('Epochs') plt.ylabel('Loss (MSE)') plt.title('Training and Validation Loss') plt.legend() plt.show()	# 绘制训练损失曲线

Table S5 势能面键长和能量对应的数据

O-H1(Å)	O-H2(Å)	Energy(Hartree)
0.8	0.8	-75.95386365
0.8	0.844	-75.97178784
0.8	0.889	-75.98098876
0.8	0.933	-75.98369829
0.8	0.978	-75.98160356
0.8	1.022	-75.97598569
0.8	1.067	-75.96781973
0.8	1.111	-75.95784831
0.8	1.156	-75.94663713
0.8	1.2	-75.93461708
0.844	0.8	-75.97178784
0.844	0.844	-75.9898456
0.844	0.889	-75.99918025
0.844	0.933	-76.00202401
0.844	0.978	-76.00006379
0.844	1.022	-75.99457992
0.844	1.067	-75.98654638
0.844	1.111	-75.97670461
0.844	1.156	-75.96561912
0.844	1.2	-75.95371973
0.889	0.8	-75.98098876
0.889	0.844	-75.99918025
0.889	0.889	-76.00864905
0.889	0.933	-76.01162737

0.889	0.978	-76.00980152
0.889	1.022	-76.00445092
0.889	1.067	-75.99654843
0.889	1.111	-75.98683432
0.889	1.156	-75.97587205
0.889	1.2	-75.96409052
0.933	0.8	-75.98369829
0.933	0.844	-76.00202401
0.933	0.889	-76.01162737
0.933	0.933	-76.01474019
0.933	0.978	-76.01304803
0.933	1.022	-76.00782932
0.933	1.067	-76.0000558
0.933	1.111	-75.99046673
0.933	1.156	-75.97962465
0.933	1.2	-75.96795769
0.978	0.8	-75.98160356
0.978	0.844	-76.00006379
0.978	0.889	-76.00980152
0.978	0.933	-76.01304803
0.978	0.978	-76.01148803
0.978	1.022	-76.00639894
0.978	1.067	-75.99875153
0.978	1.111	-75.98928417
0.978	1.156	-75.97855862
0.978	1.2	-75.96700243
1.022	0.8	-75.97598569
1.022	0.844	-75.99457992
1.022	0.889	-76.00445092
1.022	0.933	-76.00782932
1.022	0.978	-76.00639894
1.022	1.022	-76.00143631
1.022	1.067	-75.99391135
1.022	1.111	-75.98456168
1.022	1.156	-75.97394846
1.022	1.2	-75.96249879
1.067	0.8	-75.96781973
1.067	0.844	-75.98654638
1.067	0.889	-75.99654843
1.067	0.933	-76.0000558

1.067	0.978	-75.99875153
1.067	1.022	-75.99391135
1.067	1.067	-75.98650446
1.067	1.111	-75.97726787
1.067	1.156	-75.96676232
1.067	1.2	-75.95541455
1.111	0.8	-75.95784831
1.111	0.844	-75.97670461
1.111	0.889	-75.98683432
1.111	0.933	-75.99046673
1.111	0.978	-75.98928417
1.111	1.022	-75.98456168
1.111	1.067	-75.97726787
1.111	1.111	-75.96813933
1.111	1.156	-75.95773643
1.111	1.2	-75.94648566
1.156	0.8	-75.94663713
1.156	0.844	-75.96561912
1.156	0.889	-75.97587205
1.156	0.933	-75.97962465
1.156	0.978	-75.97855862
1.156	1.022	-75.97394846
1.156	1.067	-75.96676232
1.156	1.111	-75.95773643
1.156	1.156	-75.9474309
1.156	1.2	-75.93627205
1.2	0.8	-75.93461708
1.2	0.844	-75.95371973
1.2	0.889	-75.96409052
1.2	0.933	-75.96795769
1.2	0.978	-75.96700243
1.2	1.022	-75.96249879
1.2	1.067	-75.95541455
1.2	1.111	-75.94648566
1.2	1.156	-75.93627205
1.2	1.2	-75.92519987