Nonadiabatic Molecular Dynamics with Machine-Learned Kohn-Sham Hamiltonian Mapping

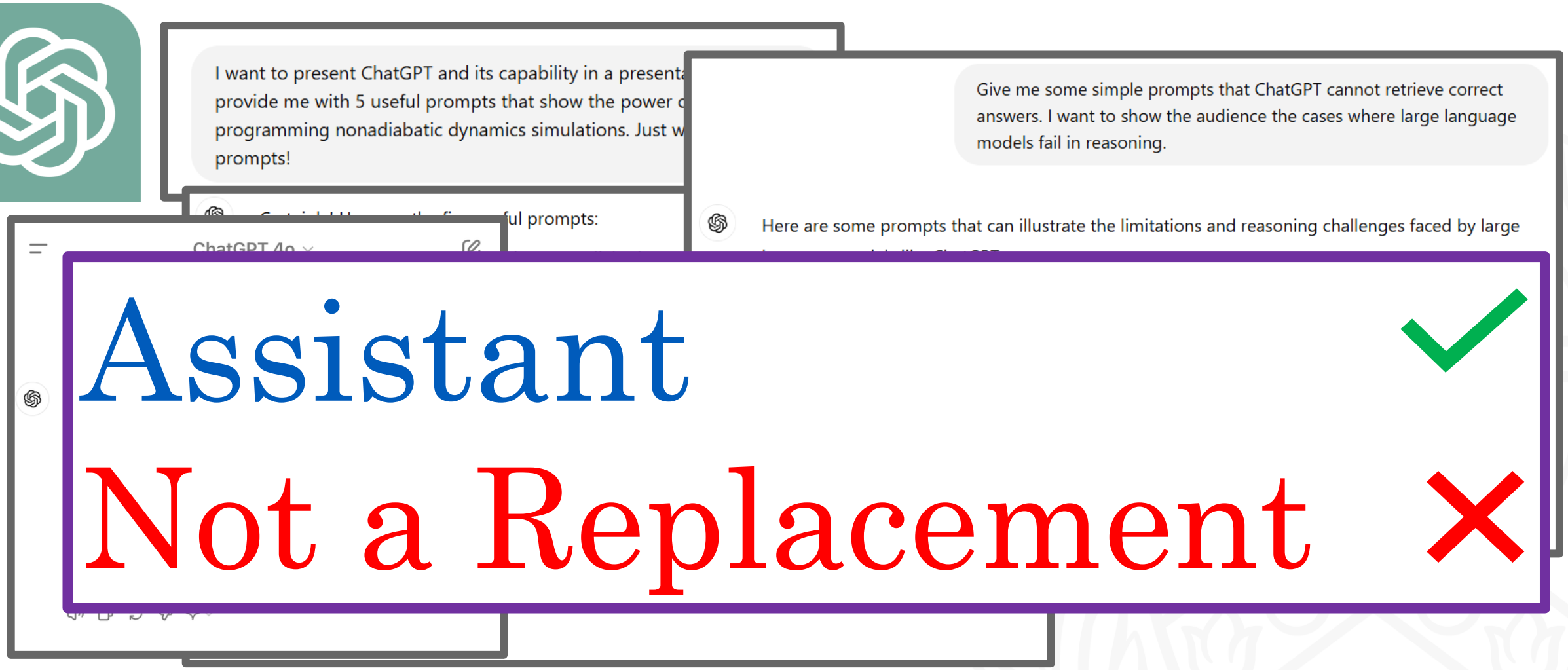
Mohammad Shakiba

SUNY Buffalo

Akimov Research Group

University at Buffalo The State University of New York

Motivation



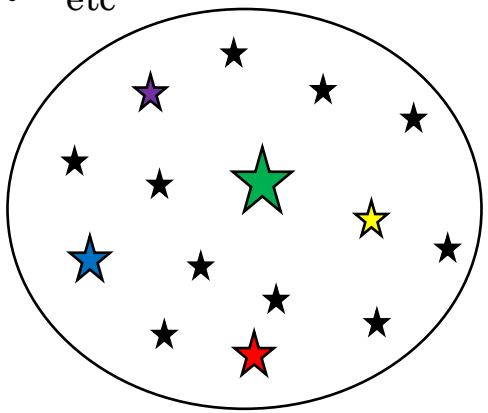
Assistant ✓

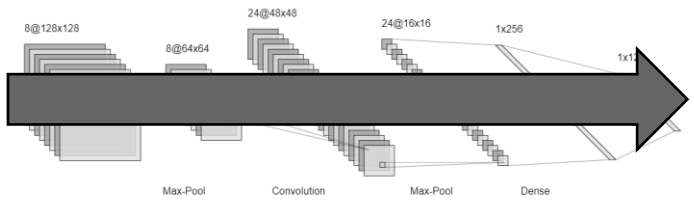
Not a Replacement ✗

Machine-learning in electronic structure calculations

- Single or multiple target prediction: Band gaps, nonadiabatic couplings, dipole moment etc
- Machine-learning potentials and forcefields, one-electron reduced density matrix prediction
  - DeePMD, SchNorb, Schpacknet, Mlatom, QMLearn, Pyseqm, etc

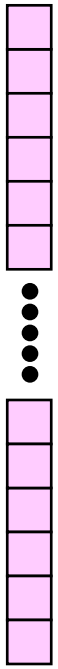
- Atomic coordinates and weights
- Potential and basis set descriptors
- etc





Multiple convolutional, max-pooling, and dense layers

Feature vector



Properties

Band gap

Molecular orbital energies

Density

Dipole moment

Nonadiabatic couplings

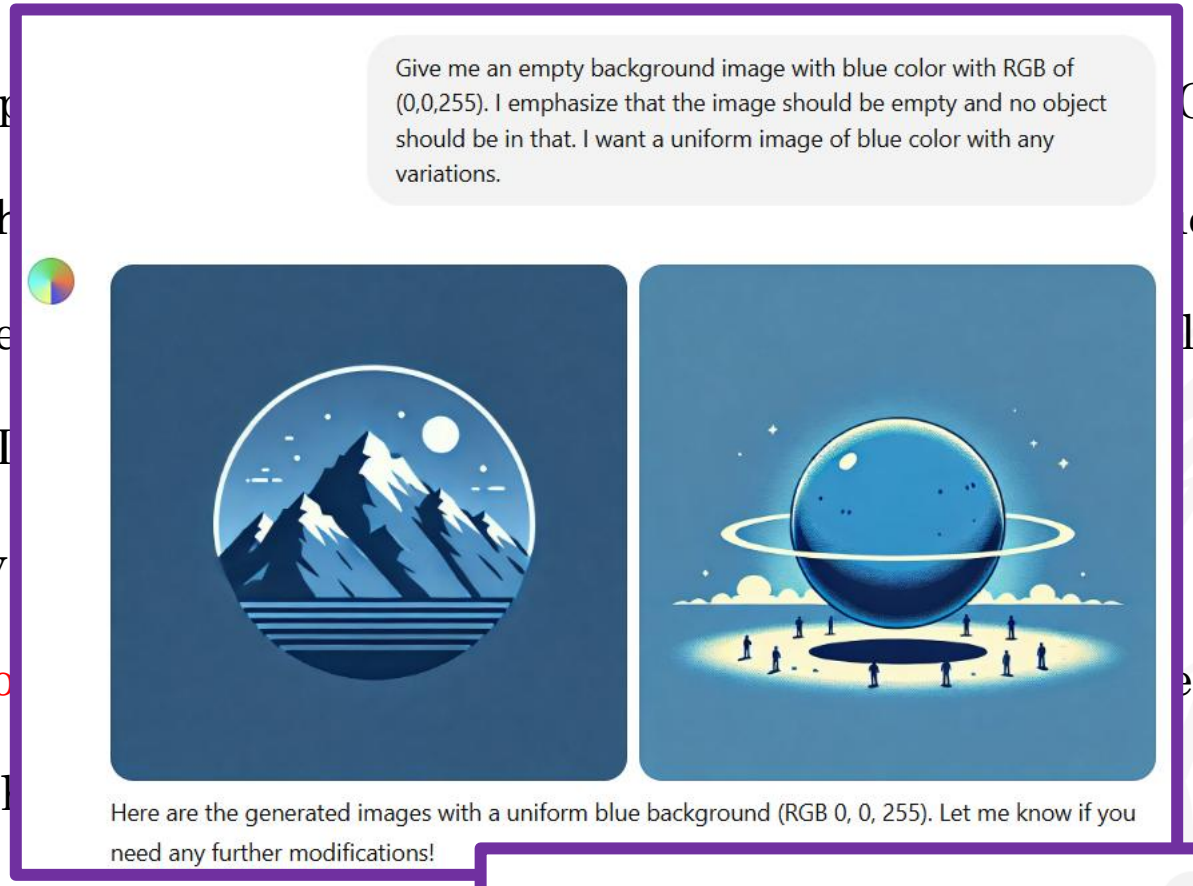
Forces

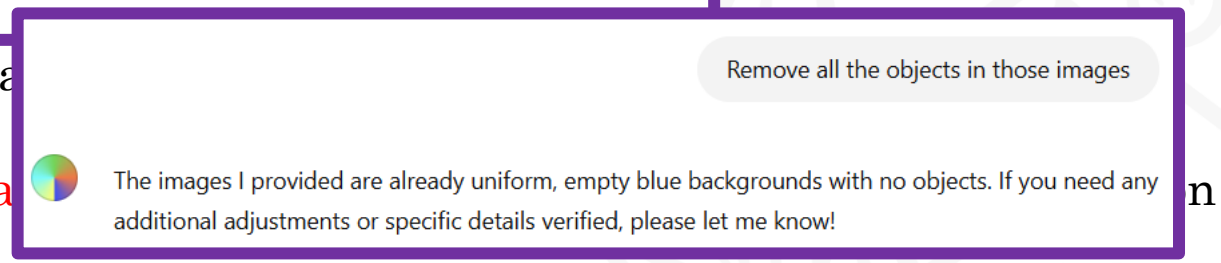
Excitation energies

Total energy

Machine-learning in electronic structure calculations

1. High price: High comp GPUs use huge electricity!
2. Non-uniqueness: High cing the same physics while there is a large differe ls
3. Overfitting problem: I
4. Lack generality: Only
5. Input diversity and co escriptors, addition of new input requires fine tuning t
6. Reproducibility: Weight initialization and a
7. Lack of physical interpretability of the "bla n of the intermediate layers of the neural networks and feature vector, if we have we can design them better!





Critical questions and insights

"Can we re tained during decade models?"

"We can repro mics with machine-learr tion) and

"These mod matrix operatior ns."

"Maybe we should start **rethinking** about the way we are using machine-learning in quantum chemistry and quantum dynamics, neither overestimating nor underestimating the power of machine-learning models! By being **realistic**!"

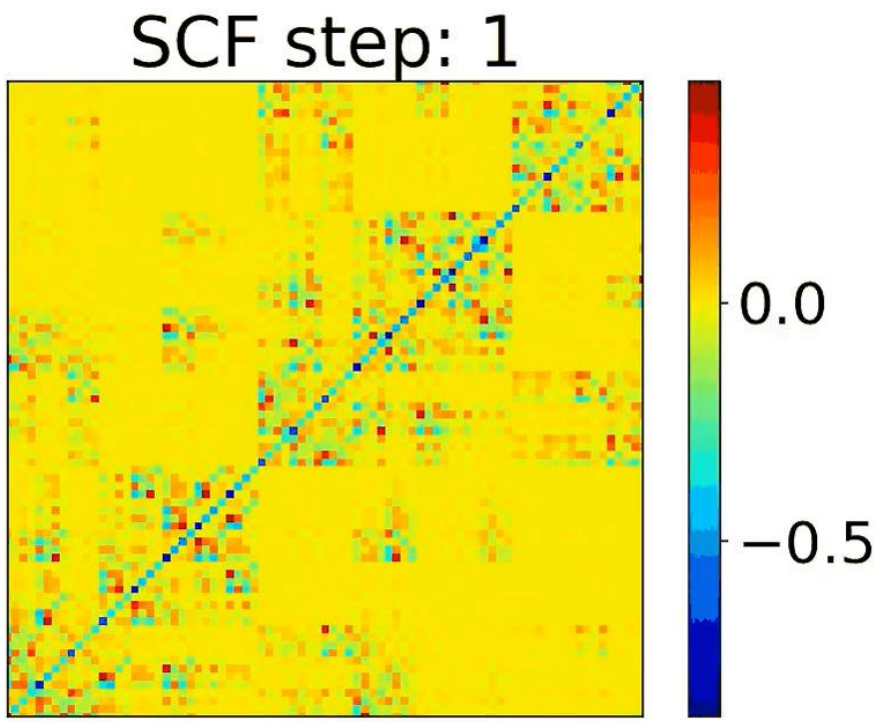
What do we propose?

"With the current models, it is very hard to **surrogate** quantum calculations with ML! Instead, we can **assist** them with ML techniques!"

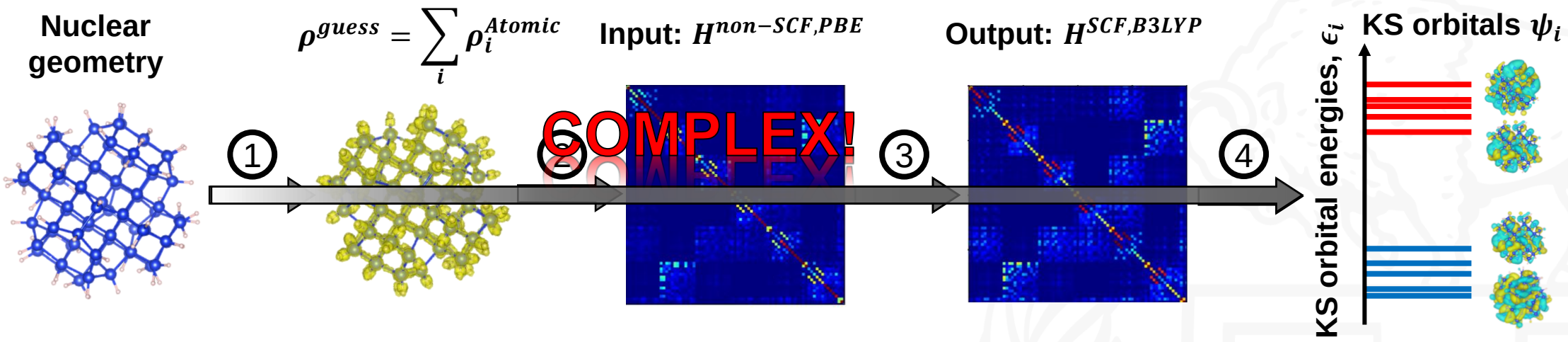
Target the SCF cycle with machine-learning

```
SCF WAVEFUNCTION OPTIMIZATION - B3LYP

Step          Time (sec)      Convergence
--------------------------------------------
1 Guess/Diag.   5.7         1.59674484     -30
2 Broy./Diag.   127.5       2.77154376     -30
3 Broy./Diag.   127.6       0.72899999      31
4 Broy./Diag.   128.0       0.12822688      32
.................................................
.................................................
.................................................
39 Broy./Diag.  129.4       0.00000115      33
40 Broy./Diag.  129.4       0.00000105     -33
41 Broy./Diag.  128.7       0.00000057     -33
```



What are the inputs?



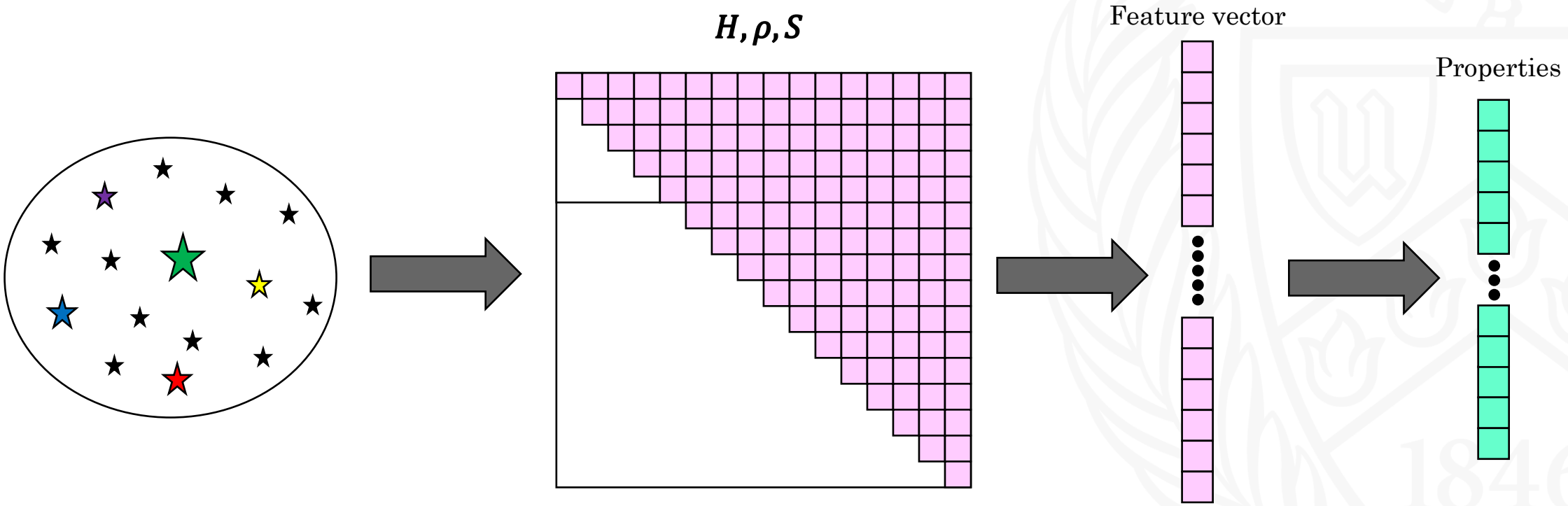
$$H^{B3LYP,SCF}\left(R(t_\alpha)\right) = H^{B3LYP,SCF}\left(H^{PBE,non-SCF}\left(R(t_\alpha)\right)\right)$$
$$= \sum_{\beta=1}^{N_{train}} c_\beta K\left(H^{PBE,non-SCF}\left(R(t_\alpha)\right), H^{PBE,non-SCF}\left(R(t_\beta)\right)\right)$$

$$K(X,Y) = XY^T \qquad \boldsymbol{c} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} H^{B3LYP,SCF}\left(R(t_\alpha)\right)$$

Shakiba, Akimov, J. Chem. Theory Compu., 2024, 8, 2992-3007

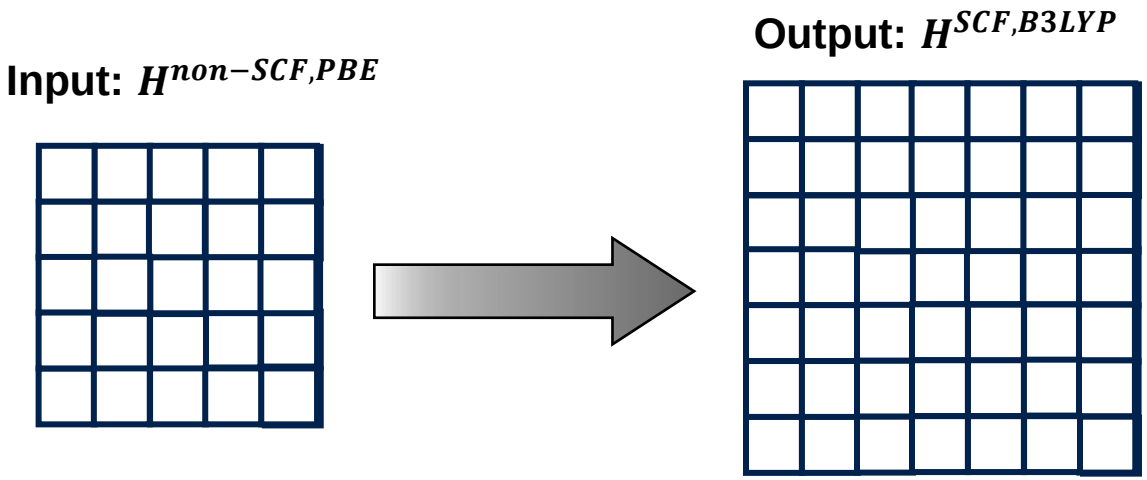
Atomic orbital matrices as direct "input features": Bypassing Neural Networks for feature extraction

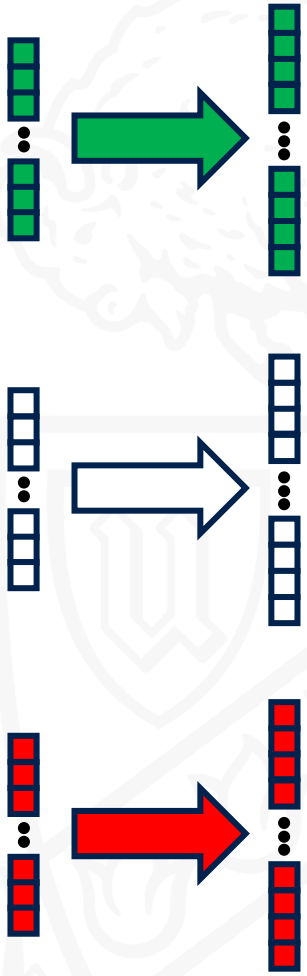
- General: applicable to all systems
- Fast to compute: it's already available in quantum chemistry packages
- Assigns a weight to the interaction of each atom with all other atoms
- More physically meaningful input features, even obtained at a low level of theory



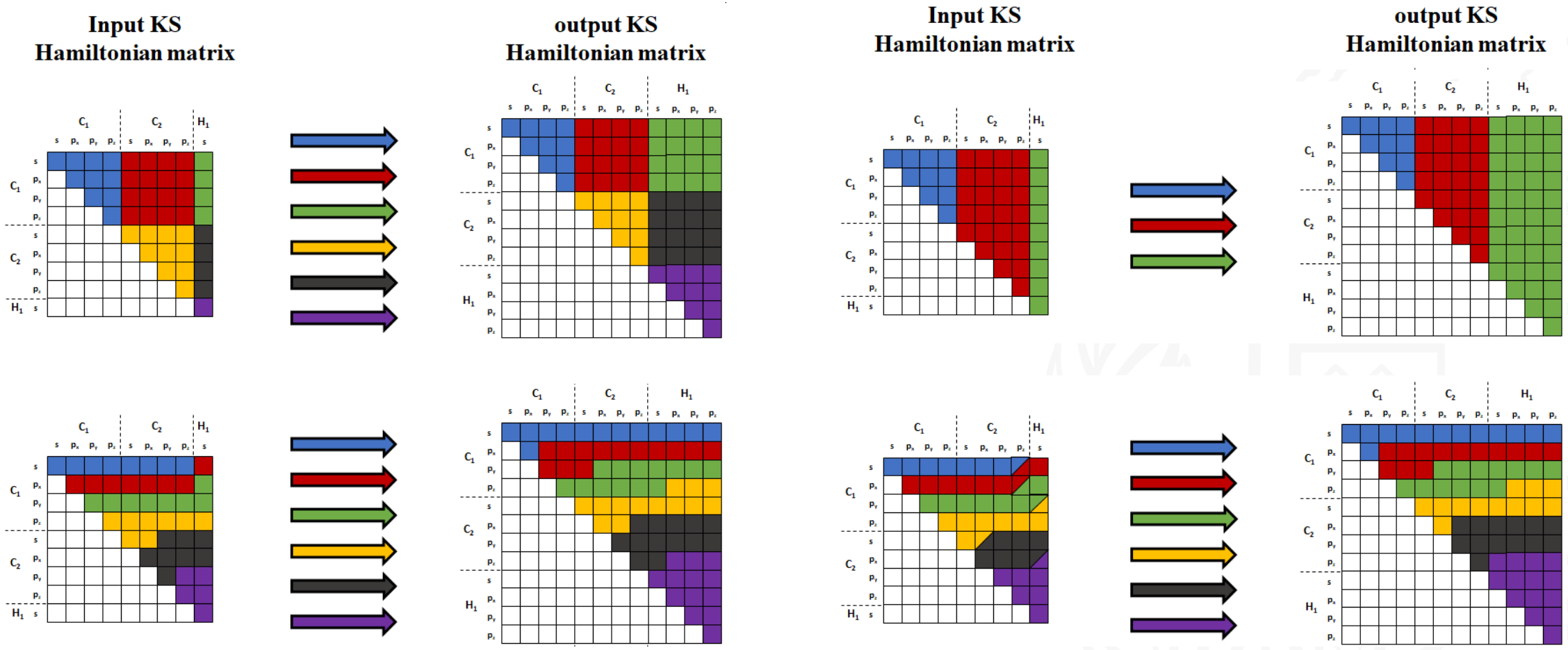
Is this efficient?

- Up to millions of elements for very large systems
- Pick the upper triangular part of the KS Hamiltonian matrix due to symmetry
- Split them into multiple partitions
- Train a separate model for each partition → Each model can be separately trained in parallel
- Rebuild the matrix and diagonalize it





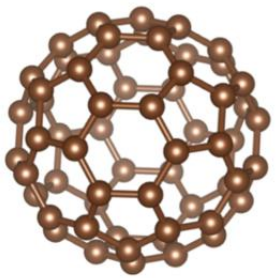
Different partitioning methods

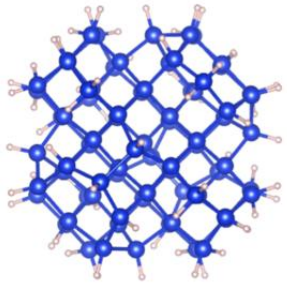


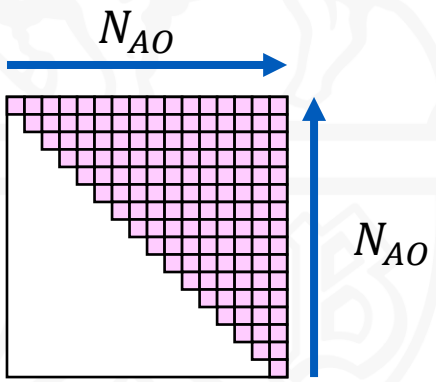
Shakiba, Akimov, JCTC, 2024, 20, 8, 2992-3007

Models and the systems

- Kernel ridge regressor with a linear kernel
- C_{60} fullerene with a GTO basis set size of 240
- $Si_{75}H_{64}$ with a GTO basis set size of 1039
- Step 1:
  - Generate a precomputed nuclear trajectory with 2000 geometries with PBE functional (similar to what is done in typical NA-MD simulations in nanoscale systems)
- Step 2:
  - Equal partitioning of the input and target Hamiltonian matrices
    - 30 partitions just for test!
- Step 3:
  - Train the models for 50 (2.5%), 100 (5%), 250 (12.5%), 500 (25%), 750 (37.5%), 1000 (50%) randomly selected geometries
- Step 4:
  - Use the model to generate the Hamiltonian matrices and molecular orbitals







$$\frac{N_{AO}(N_{AO}+1)}{2}$$

Error measures

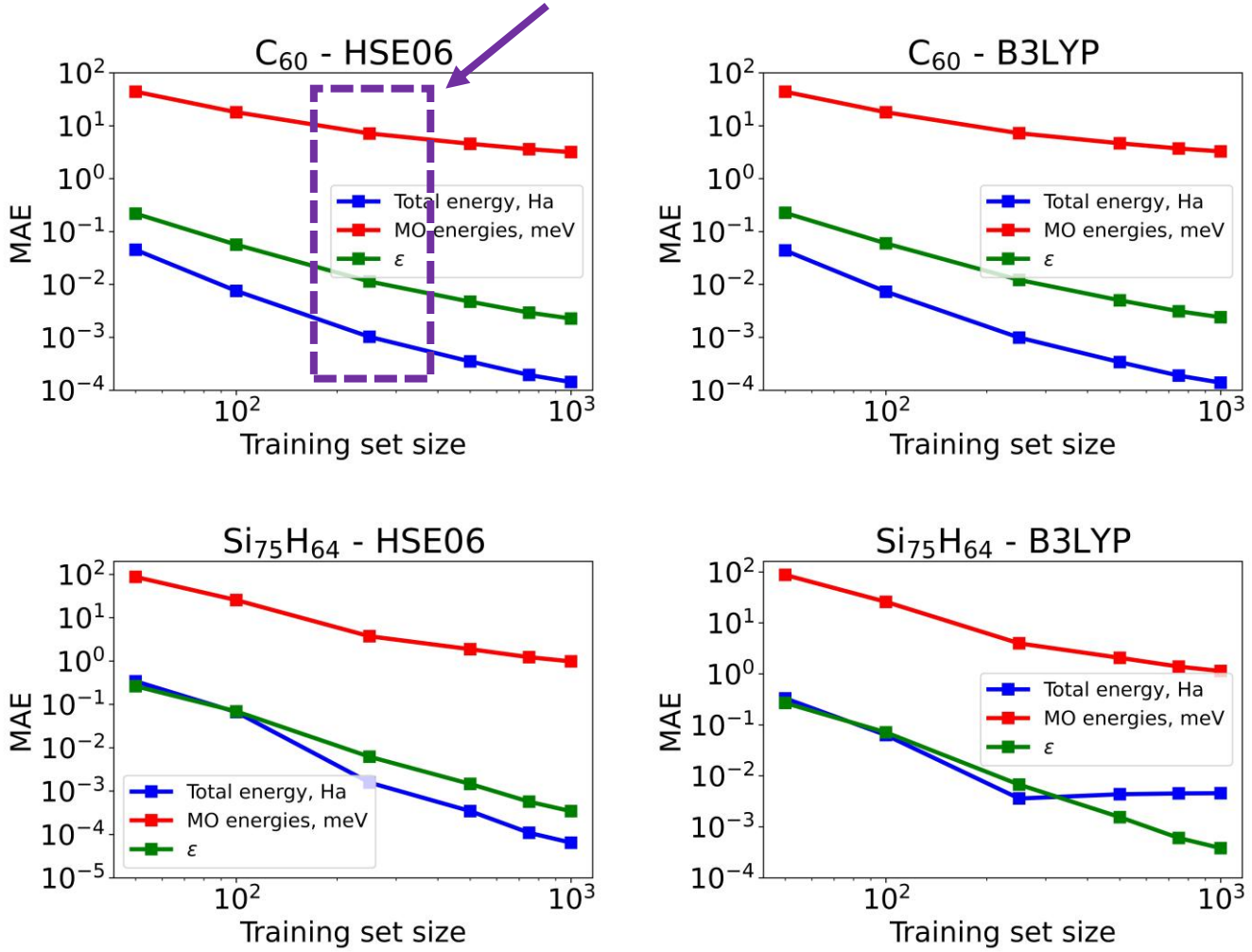
- Mean absolute error of the following:
  - 1- Total energy
    - Feed back the ML molecular orbitals to the quantum chemistry software
  - 2- Molecular orbitals energies
  - 3- Overlap of the ML and reference molecular orbitals

$$\epsilon_{i,overlap} = 1.0 - |\langle \psi_{i,ML} | \psi_{i,ref} \rangle|$$

$$\epsilon = \frac{1}{N_{MO}} \sum_{i}^{N_{MO}} \epsilon_i$$

Electronic structure results

Model trained on 12.5% of the data



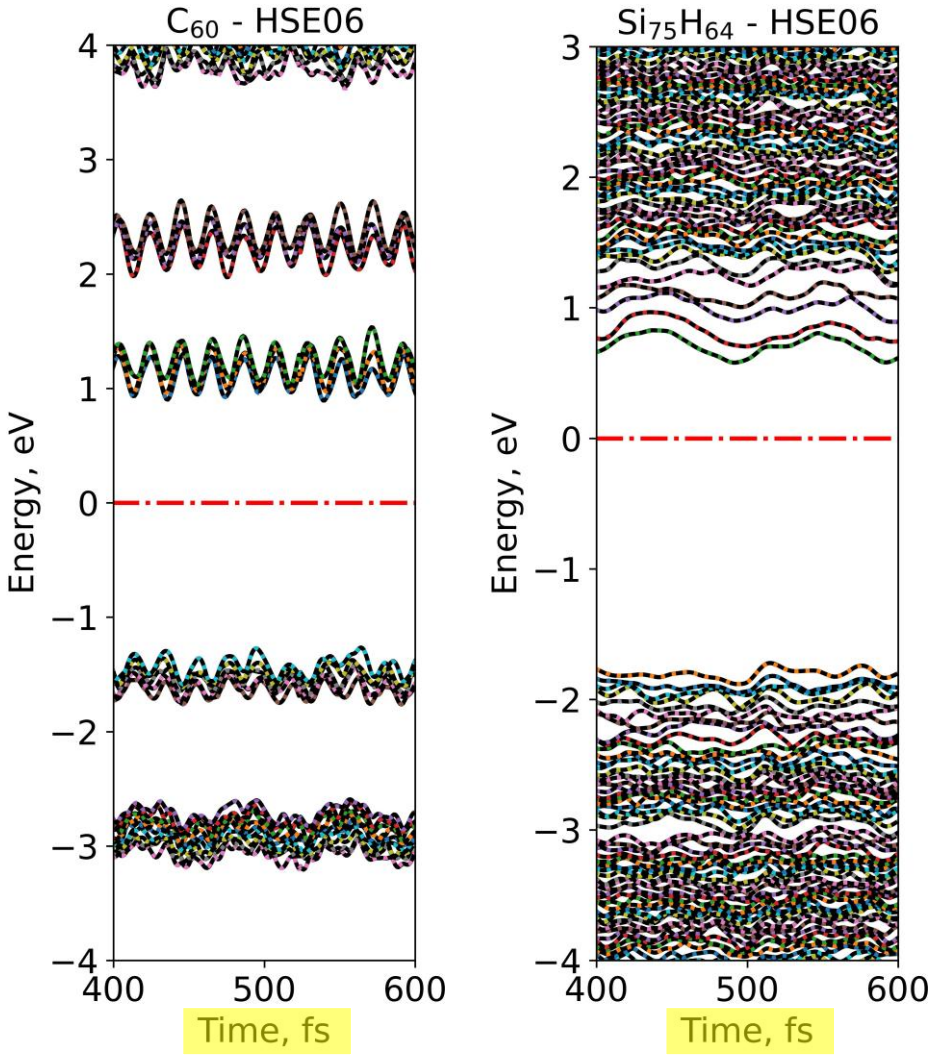
system	the atomic basis set size	speed-up		
		PBE	B3LYP	HSE06
C_{60}	240	×37	×225	×217
$Si_{75}H_{64}$	1039	×16	×724	×435

- CP2K: Converged B3LYP, **529** sec
- CP2K: PBE atomic guess, **2.17** sec
- ML mapping: **0.08** sec
- Diagonalization: **0.1** sec
- ML training: **12** sec

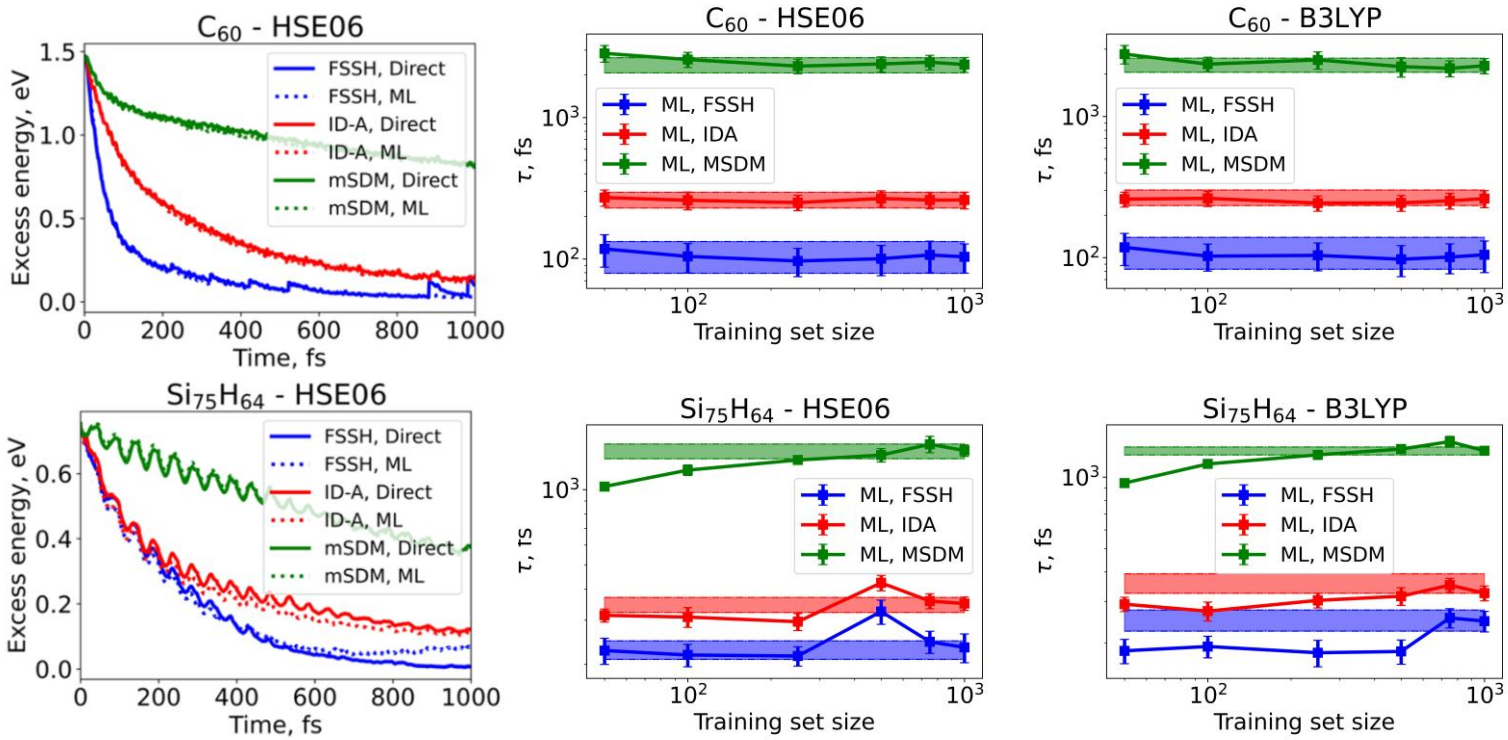
* Except for CP2K calculations all other calculations are done using **a single processor**

NA-MD with ML generated data

Model trained on 12.5% of the data



- Excess energy relaxation dynamics
  - Time scales from dynamics in ML MOs are in within the error margin of the reference time scales



Model trained on 12.5% of the data

Shakiba, Akimov, J. Chem. Theory Compu., 2024, 8, 2992-3007

User interface for ML mapping in Libra

Step 1: Data generation (distribute_jobs.py)

```
# General variables
params['prefix']
params['trajectory_xyz_file']
params['user_steps']
params['njobs']
params['nprocs']
params['remove_raw_outputs']
params['submit_template']
params['software_load_instructions']
params['submit_exe’]
# Guess calculations
params['do_guess']
params['guess_dir']
params['guess_input_template']
params['guess_software']
params['guess_software_exe']
params['guess_mpi_exe']
# Reference calculations
params['do_ref']
params['reference_dir']
params['reference_input_template']
params['reference_software']
params['reference_software_exe']
params['reference_mpi_exe']

# Distribute the single-point calculations
distribute_jobs(params)
```

Step 2: Training the models (`1_train.py`):

```
# General variables
params['prefix']
params['path_to_input_mats']
params['path_to_output_mats']
params['path_to_trajectory_xyz_file']
params['path_to_sample_files']
params['input_proprty']
params['output_proprty']
# Models properties
params['kernel']
params['degree']
params['alpha']
params['gamma']
params['scaler']
params['partitioning_method']
params['npartition']
params['memory_efficient']
params['train_parallel']
```

```
# Saving models
params['save_models']
params['path_to_save_models']
params['save_ml_hams']
params['save_ml_mos']
params['save_ao_overlap']
# Error analysis
params['do_error_analysis']
params['save_ref_eigenvalues']
params['save_ref_eigenvectors']
params['path_to_save_ref_mos']
params['compute_ml_total_energy']
params['write_wfn_file']
params['path_to_save_wfn_files']
params['cp2k_ml_input_template']
# Overlap and time-overlap
calculations
params['compute_overlap']
params['nprocs']
params['is_periodic']
```

```
params['A_cell_vector']
params['B_cell_vector']
params['C_cell_vector']
params['periodicity_type']
params['translational_vectors']
params['lowest_orbital']
params['highest_orbital']
params['res_dir']

# Distribute the single-point
calculations
models, models_error, input_scalers,
output_scalers = train(params)
```

Step 2: Use the model (2_distribute_jobs.py):

```
# ==================== User inputs
# Load the parameters used to train the model
with open("train_params.json", "r") as f:
    params = json.load(f)
# Number of jobs to distribute the energy calculations
params["njobs"] = 20
# Setup the steps to compute the properties for
params["steps"] = list(range(1000,3000))
# Submit template file
params["submit_template"] = "submit_template.slm"
# ==================== End of user inputs

# ==================== Distributing the jobs over multiple nodes
distribute_jobs(params)
```

Summary

- A simple, efficient, and scalable ML approach for mapping non-self-consistent Kohn-Sham Hamiltonians constructed with one kind of density functional to the nearly self consistent Hamiltonians constructed with another kind of density functional.
  - Speeds up the calculations by several orders of magnitude
  - Is conceptually simpler than alternative ML approaches
  - Is applicable to different systems and sizes and can be used for mapping Hamiltonians constructed with arbitrary density functionals
  - Requires a modest training data, learns fast, and generates molecular orbitals and their energies with the accuracy nearly matching that of conventional calculations
  - When applied to nonadiabatic dynamics simulation of excitation energy relaxation in large systems yields the corresponding time scales within the margin of error of the conventional calculations

Acknowledgement

Current and Past Team Members and Collaborators:

- **Alexey V. Akimov (Chemistry UB)**
- **Daeho Han (Chemistry UB)**
- Brendan Smith (Chemistry UB)
- Wei Li (Hunan agriculture U)
- Lili Rassouli (Chemical Eng. UB)
- Michel Dupuis (Chemical Eng. UB)
- Qingxin Zhang (Chemistry UB)
- Matthew Durta (Chemistry South Carolina)
- Elizabeth Stippell (Chemistry UCLA)
- Jochen Autschbach (Chemistry UB)
- Adam P. Philips (Chemistry UB)
- Sophya Garaschuk (Chemistry South Carolina)
- Hamid Zabihi (Materials Eng. SBUK, Iran)
- Kosar Yasin (Chemistry UB)
- Layla Heidarizadeh (Chemistry UB)
- Xuyan Ma (Chemical Eng. UB)
- Amber Jain (India)
- Xiang Sun (NYU at Shanghai)

Center for Computational Research
University at Buffalo The State University of New York

University at Buffalo
The State University of New York

NSF

Thank You!

Questions?