# Kohn-Sham Hamiltonian Mapping Approach with Machine-Learning

CompChemCyberTraining Workshop, July 2024

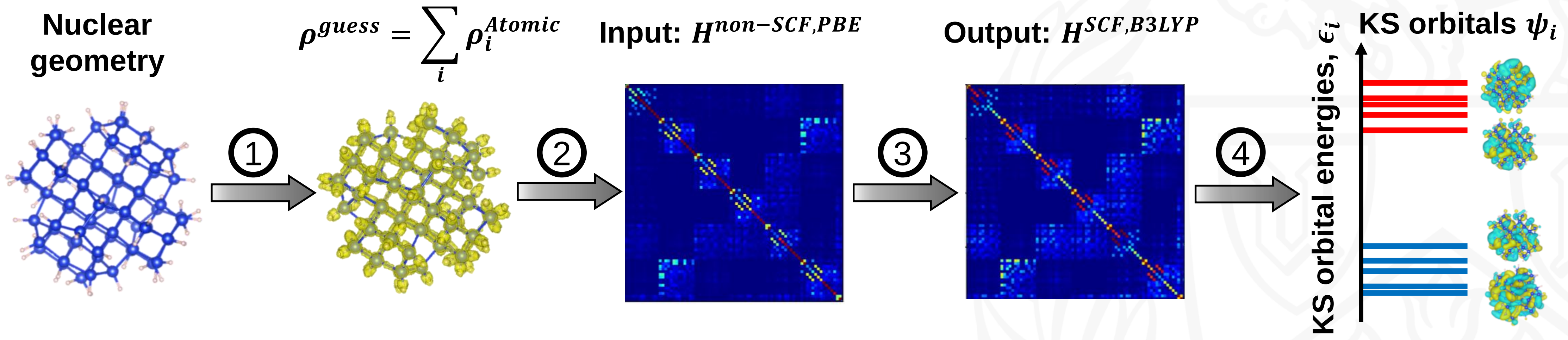
Mohammad Shakiba

SUNY Buffalo

Akimov Research Group

University at Buffalo The State University of New York

## Kohn-Sham Hamiltonian mapping workflow



## Kohn-Sham Hamiltonian mapping workflow

- Step 1 (**data generation**):
  - Generate a precomputed nuclear trajectory with either PBE or xTB (similar to what is done in typical NA-MD simulations in nanoscale systems)
  - Generate guess (xTB or PBE atomic guess) and target Hamiltonian (hybrid functional) for a set of random geometries of the precomputed trajectory (**high-throughput**)
- Step 2 (**training the models**):
  - Select the partitioning of the input and target Hamiltonian matrices
    - Equal partitioning
    - Atomic partitioning
  - Train multiple models for each partition with the generated data
- Step 3 (**use the model**):
  - Generate guess Hamiltonians for all geometries and use the trained models to generate the target Hamiltonian matrices and molecular orbitals (**high-throughput**)

## Kohn-Sham Hamiltonian mapping workflow

- Step 1 (**data generation**):
  - Generate a precomputed nuclear trajectory with either PBE or xTB (similar to what is done in typical NA-MD simulations in nanoscale systems)
  - Generate guess (xTB or PBE atomic guess) and target Hamiltonian (hybrid functional) for a set of random geometries of the precomputed trajectory (**high-throughput**)
- Step 2 (**training the models**):
  - Select the partitioning of the input and target Hamiltonian matrices
    - Equal partitioning
    - Atomic partitioning
  - Train multiple models for each partition with the generated data
- Step 3 (**use the model**):
  - Generate guess Hamiltonians for all geometries and use the trained models to generate the target Hamiltonian matrices and molecular orbitals (**high-throughput**)

University at Buffalo The State University of New York

## 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)
```

University at Buffalo The State University of New York

## 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)
```

# 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)
```

## Let’s do the calculations!