**USER GUIDE FOR THE COMPUTATIONAL PROGRAM FOR MATERIAL ANALYSIS**

**USER GUIDE FOR CRYSTALPLOT: UNIT CELL STRUCTURE VISUALIZATION CODE**

1. **Introduction**  
   CrystalPlot is an extension application developed on the MATLAB platform for visualizing crystal structures from POSCAR-formatted data. This application enables users to create high-quality graphical representations of unit cell structures, making it particularly suitable for materials science research and academic publications.

Key features:

* Supports importing POSCAR files from the VASP software
* Visualizes crystal structures with customizable and aesthetically pleasing graphics
* Saves high-quality graphical images for reporting and publication purposes

1. **System Requirements**

* MATLAB (version 2019a or later)
* CrystalPlot.mlappinstall application package

1. **Installation**

* Open MATLAB
* Select "Apps" on the toolbar
* Click "Install App" and select the CrystalPlot.mlappinstall file from the source folder
* After installation, the application will appear in the Apps section with the CrystalPlot icon

1. **Usage Guide**

* Launch the application:
  + Access "Apps" in MATLAB and select CrystalPlot
* Import data:
  + Click "Import" in the CrystalPlot interface
  + Select the POSCAR structure file from your directory
* Customize the image:
  + Adjust display parameters such as color, atom size, and bonding styles (if supported)
* Export images:
  + Click "Export" to save images in popular formats like PNG, JPG, or SVG

1. **Usage Tips**

* Ensure that the POSCAR file is correctly formatted according to VASP standards for accurate structure recognition
* Use MATLAB's zoom and rotation tools to adjust the viewing angle before saving the image

1. **Troubleshooting Common Issues**

* Unable to launch the application:
  + Verify that MATLAB is installed with the correct version
  + Ensure that the correct CrystalPlot.mlappinstall file was selected during installation
* Error when importing POSCAR files:
  + Confirm that the POSCAR file is error-free and contains complete information on lattice vectors and atomic positions

1. **References**

* For further information or support, please contact the development team via email or refer to the documentation provided with the product.

**USER GUIDE FOR ML\_AHC\_CAL: AUTOMATED AHC CALCULATION CODE**

1. **Introduction**  
   ML\_AHC\_cal is a Python-based code that automates the calculation of Anomalous Hall Conductivity (AHC) for compounds using structural data and DFT parameters. The program integrates tools like VASP and Wannier90 to perform precise calculations and streamline the workflow.
2. **System Requirements**

* Operating system: Linux (recommended for better support of scientific computing tools)
* Required software:
  + Python 3.x (necessary libraries: os, shutil, numpy, re)
  + VASP (version 5.4.4 or later) installed at a path specified in the code
  + Wannier90 (version 3.1.0 or later) installed at a path specified in the code

1. **Input Information**

* Directory structure:
  + Project directory: Each compound to be calculated for AHC must have a separate folder containing:
    - step1: Files such as POSCAR, POTCAR, INCAR, and KPOINTS for the first DFT calculation step
* Data files:
  + structures.in: A list of folder names containing compound structures to be calculated. If this file is absent, the program will automatically list folders in the current path.
  + POSCAR file in each folder containing crystal structure information

1. **Workflow**  
   The automated process is divided into 5 steps:

* Step 2 - Prepare Wannier90 data:
  + Use the CHGCAR file from step1 to create the wannier90.win file with lattice vector, atomic position, and Fermi energy parameters
  + Perform SOC (Spin-Orbit Coupling) calculations using VASP
* Step 3 - Optimize Wannier90 structure:
  + Optimize Wannier90 parameters to improve accuracy
* Band\_w90 - Calculate energy bands using Wannier90:
  + Use optimized files from step 3 to compute and visualize energy bands with Wannier90
* Band\_vasp - Calculate energy bands using VASP:
  + Create a KPOINTS file to calculate energy bands through VASP
* Step 4 - Calculate AHC:
  + Use Wannier90 to compute Berry curvature and AHC on the k-mesh grid

1. **How to Use**

* Prepare the environment:
  + Ensure VASP and Wannier90 are installed
  + Declare the paths to VASP and Wannier90 executables in the source code:

python

vasp\_link = '/path/to/vasp/bin'

wannier90\_link = '/path/to/wannier90/bin'

* Run the program:
  + Save the source code as ML\_AHC\_cal.py
  + Run the program:

bash

python ML\_AHC\_cal.py

* Output:
  + AHC results: Saved in the wannier90.wout file in the step4 folder for each compound
  + Wannier90 energy bands: Output files from band\_w90 to plot the energy spectrum
  + VASP energy bands: Result files from band\_vasp

1. **Notes**

* Ensure the structure and input files in the compound folders are complete (POSCAR, POTCAR, KPOINTS, INCAR)
* Check the output.log file to monitor the calculation process and identify errors
* Checkpoint files allow resuming calculations from the previous step if interrupted

1. **Troubleshooting Common Issues**

* Checkpoint not found:
  + Ensure the compound folder contains results from the previous step
* Invalid path error:
  + Verify and correctly specify the paths to VASP and Wannier90

1. **Results**  
   The program automates the entire workflow from DFT calculations to AHC output, providing Berry curvature data and energy spectra. This supports research into anomalous Hall properties in materials.

**USER GUIDE: ML\_Combine\_OQMD.py**

**Purpose**  
ML\_Combine\_OQMD.py is a Python script designed to create a database in the OQMD format from input datasets. The output is a .csv file containing material properties, supporting material research and machine learning applications.

**Requirements**

1. Environment:
   * Python (version >= 3.8)
   * Python libraries: pandas, numpy
2. Necessary folders and files:
   * **databases** folder: Contains .csv files with material information from various data sources
   * **Elements.csv** file: Contains chemical element data (including properties like atomic number, atomic radius, electronegativity, etc.)

**Usage**

1. **Prepare the data**
   * Ensure the databases folder contains the required input .csv files to be combined
   * Verify that Elements.csv includes complete information for all related elements
2. **Run the script**
   * Open a terminal or a Python-supported IDE
   * Navigate to the folder containing the ML\_Combine\_OQMD.py file
   * Execute the command:

bash

python ML\_Combine\_OQMD.py

1. **Processing workflow**
   * **Combine data (combineData):**
     + Merges all .csv files in the databases folder into a single file named oqmd\_data.csv
   * **Process data (processing\_data):**
     + Removes the band\_gap column (if present)
     + Converts volume data into lattice constants
     + Replaces NaN values in the moment column with 0
     + Saves the result into processed\_oqmd\_data.csv
   * **Generate input data (make\_input):**
     + Reads data from processed\_oqmd\_data.csv and combines it with information from Elements.csv
     + Computes properties related to elements X, Y, and Z (e.g., atomic number, atomic radius, electronegativity)
     + Calculates atomic radius ratios between X, Y, and Z (ratio\_XY, ratio\_XZ, ratio\_YZ)
     + Saves the complete dataset into oqmd\_data\_plus.csv

**Output**

* **oqmd\_data.csv:** Combined .csv file containing data from the databases folder
* **processed\_oqmd\_data.csv:** Processed .csv file after data cleaning and transformation
* **oqmd\_data\_plus.csv:** Final .csv file with complete material information in OQMD format

**Notes**

* If required files or folders are missing, the script will not function correctly. Ensure all input data is available before execution
* The output data can be used for further research, including machine learning applications or material analysis

**USER GUIDE: auto\_fleur (ML\_SHC+OHC\_calc)**

**Introduction**  
The auto\_fleur folder contains a Python script (auto\_shc\_wannier90\_fleur.py) designed to automate the calculation of Spin Hall Conductivity (SHC) and Orbital Hall Conductivity (OHC) using FLEUR and Wannier90. It is a crucial tool for material simulation based on DFT.

**System Requirements**

1. Required software:
   * FLEUR
   * Wannier90
   * Python (>= 3.8)
   * Libraries: numpy, shutil, xml.etree.ElementTree
2. Necessary folders and files:
   * elements.list: Contains a list of elements and their parameters (name, lattice type, lattice constant)
   * run.sh: A bash script to execute the Python program

**How to Use**

1. **Prepare the data**
   * Ensure the auto\_fleur folder contains the following files:
     + auto\_shc\_wannier90\_fleur.py
     + elements.list
     + run.sh
   * Format of elements.list:

php

<Element Name> <Lattice Type (fcc/bcc/hcp)> <Lattice Constant>

Example:

Fe bcc 2.87

Co hcp 2.51

Ni fcc 3.52

1. **Run the program**
   * Use the terminal to navigate to the auto\_fleur directory
   * Execute the following command:

bash

bash run.sh

1. **Calculation Workflow**
   * **Step 1: SCF structure**
     + Generate POSCAR, KPOINTS, and POTCAR files from element data
     + Run VASP SCF calculations to optimize the structure and compute electronic density
   * **Step 2: Input preparation for FLEUR**
     + Convert data from Step 1 into inp files for FLEUR
     + Perform SCF calculations using FLEUR
   * **Step 3: Generate Wannier Functions (WF)**
     + Create projgen\_inp to initialize Wannier functions from FLEUR
     + Run projgen, prepwan90, and matrixamn steps
   * **Step 4: Optimize WF**
     + Optimize Wannier functions using Wannier90
   * **Step 5: Compute SHC and OHC**
     + Use Wannier parameters to calculate spin and orbital conductivities with Wannier90
2. **Program Log**
   * The calculation process is logged in output.log
   * Upon completion, SHC/OHC results are saved in the step5 directory

**Output**

* **Folders for each calculation step:**
  + Subdirectories such as step1, step2, ..., step5\_tb store results from each step
* **SHC/OHC values:**
  + Stored in result files within the step5 or step5\_tb directories

**Notes**

1. **Software dependencies:**
   * Ensure that compatible versions of VASP, FLEUR, and Wannier90 are installed and correctly configured
2. **Lattice structure:**
   * Verify the elements.list file to ensure accurate lattice type and lattice constant values
3. **Parameter tuning:**
   * Modify parameters like encut, kmesh, and others in the source code to suit your specific research needs

**USER GUIDE: auto\_vasp (ML\_SHC+OHC\_calc)**

**Introduction**  
The auto\_vasp folder contains a Python script (auto\_shc\_wannier90\_vasp.py) designed for the automated calculation of Spin Hall Conductivity (SHC) and Orbital Hall Conductivity (OHC) using VASP and Wannier90. This powerful tool facilitates material simulations through DFT (Density Functional Theory) combined with Wannierized models.

**1. Components in auto\_vasp**

* **Main source file:**
  + auto\_shc\_wannier90\_vasp.py: The primary Python script for automating SHC/OHC calculations
* **Data file:**
  + elements.list: A list of elements and related parameters such as crystal lattice type and lattice constant
* **Execution script:**
  + run.sh: Shell script to run the Python program

**2. Structure of elements.list**

This file contains the list of elements used to automate the workflow. Format:

php

<Element Name> <Lattice Type (fcc/bcc/hcp)> <Lattice Constant> <Use Wannier (1/0)> <Calculate AHC (1/0)>

Example:

php

Fe bcc 2.87 1 1

Co hcp 2.51 0 0

Ni fcc 3.52 1 0

* **1/0 in the fourth column:** Specifies whether to optimize Wannier functions (1: Yes, 0: No)
* **1/0 in the fifth column:** Indicates whether to calculate Anomalous Hall Conductivity (AHC) (1: Yes, 0: No)

**3. Usage Guide**

**3.1. Prepare the environment**

1. **Install necessary software:**
   * **VASP:** Version supporting spin-orbit coupling (SOC) calculations
   * **Wannier90:** Version >= 3.1.0
   * **Python:** Requires libraries such as numpy and shutil
2. **Verify software paths in the script:**
   * The paths for VASP, Wannier90, and Wannier4Hall are defined in the script:

python

vasp\_link = '/home/user2/thiho/app/vasp.5.4.4\_wan3'

wannier90\_link = '/home/user2/thiho/app/wannier90\_OHC/wannier90-3.1.0'

wan4hall\_link = '/home/user2/thiho/app/wannier4hall'

**3.2. Run the script**

1. Navigate to the auto\_vasp folder in the terminal:

bash

cd /path/to/auto\_vasp

1. Execute the following command:

bash

bash run.sh

**4. Calculation Steps**

**4.1. Step 1: Data Preparation (SCF Calculation)**

* Automatically generates POSCAR, INCAR, KPOINTS, and POTCAR files
* Runs SCF calculations using VASP with srun

**4.2. Step 2: SOC Calculations and Wannier Optimization**

* Computes SOC and prepares input files for Wannier90
* Optimizes Wannier functions if specified in elements.list

**4.3. Step 3: Initialize Wannier Functions**

* Generates wannier90.win and runs Wannier function optimization via Wannier90

**4.4. Step 4: Compute SHC/OHC**

* Calculates Spin Hall Conductivity (SHC) or Anomalous Hall Conductivity (AHC) based on Wannier data

**4.5. Step 5: Additional Calculations with Wannier4Hall**

* Uses Wannier4Hall to compute OHC or simulate the band structure

**4.6. Band Structure (optional)**

* Generates band structure plots using VASP or Wannier90

**5. Output**

* **Log file:** output.log records the status of each calculation step
* **Results:**
  + SHC/OHC values are stored in the step4 or step4\_tb folder
  + Band structures (if generated) are saved in band\_vasp or band\_w90

**6. Notes**

1. **Software availability:**
   * Ensure all required software (VASP, Wannier90, etc.) is correctly installed and the paths are accurate
2. **Element structure:**
   * Verify the accuracy of elements.list for lattice type and lattice constant inputs
3. **Calculation parameters:**
   * Adjust parameters such as kmesh, encut, or num\_iter in the script to fit your research objectives

**USER GUIDE: plot\_figure (ML\_SHC+OHC\_calc)**

**Introduction**  
The plot\_figure folder contains MATLAB scripts to visualize results from SHC/OHC calculations performed in the auto\_fleur and auto\_vasp directories. These MATLAB scripts process output data from VASP, Fleur, or Wannier90 to generate plots, including:

* Brillouin Zone (BZ) structure
* Band structure
* Spin Hall Conductivity (SHC)
* Orbital Hall Conductivity (OHC)

**1. Components in plot\_figure**

**1.1. Main MATLAB scripts:**

* get\_inside\_BZ.m: Identifies k-points inside the Brillouin Zone (BZ).
* plot\_bands\_berry\_fleur.m: Plots band structure from Fleur results.
* plot\_bands\_combine\_vasp.m: Combines VASP band structure data with SHC/OHC values.
* plot\_data.m: Displays output data from VASP or Wannier90.
* plot\_scan.m: Scans Fermi energy and plots SHC/OHC results.

**1.2. Supporting data files:**

* colormap\_ud.mat: Stores colormap definitions for use in plots.

**1.3. Auxiliary functions:**

* import\_procar\_ohc.m: Imports data from PROCAR files or similar formats for analysis.
* plot\_3dBZ\_fleur.m: Generates 3D Brillouin Zone plots from Fleur data.

**2. MATLAB Usage Guide**

**2.1. Prepare MATLAB Environment**

1. Open MATLAB.
2. Change the working directory to the plot\_figure folder:

matlab

cd /path/to/plot\_figure

1. Load the required data files (e.g., outputs from VASP or Fleur).

**2.2. Example Usage**

**a. Identify k-points in the Brillouin Zone (BZ)**

* **Script:** get\_inside\_BZ.m
* **Usage:**

matlab

[id, knew] = get\_inside\_BZ(kpoints, hkl, G, inv\_lat, true);

* **Inputs:**
  + kpoints: List of initial k-points (4 columns: kx, ky, kz, validity index).
  + hkl: Set of reciprocal lattice vectors.
  + G: Matrix of reciprocal lattice vectors.
  + inv\_lat: Inverse of the lattice matrix.
  + conv: Boolean to indicate if k-coordinates need conversion to reciprocal space (true/false).
* **Outputs:**
  + id: Indices of k-points inside the BZ.
  + knew: Selected k-points in reciprocal space.

**b. Plot band structure from Fleur**

* **Script:** plot\_bands\_berry\_fleur.m
* **Usage:**

matlab

plot\_bands\_berry\_fleur('path\_to\_data');

* **Input:**
  + path\_to\_data: Path to Fleur output data.
* **Result:**
  + Band structure plot with SHC/OHC values highlighted (if available).

**c. Plot band structure from VASP**

* **Script:** plot\_bands\_combine\_vasp.m
* **Usage:**

matlab

plot\_bands\_combine\_vasp('path\_to\_band\_data', 'path\_to\_SHC\_data');

* **Inputs:**
  + path\_to\_band\_data: Path to VASP band structure data.
  + path\_to\_SHC\_data: SHC data exported from Wannier90 or VASP.
* **Result:**
  + Combined band structure plot with SHC results.

**d. Display OHC data**

* **Script:** plot\_data.m
* **Usage:**

matlab

plot\_data('path\_to\_OHC\_data');

* **Input:**
  + path\_to\_OHC\_data: Path to OHC data file.
* **Result:**
  + Visualization of Orbital Hall Conductivity values in reciprocal space or along the energy axis.

**e. Scan Fermi energy for SHC analysis**

* **Script:** plot\_scan.m
* **Usage:**

matlab

plot\_scan('path\_to\_SHC\_scan\_data');

* **Input:**
  + path\_to\_SHC\_scan\_data: File containing SHC data during Fermi energy scans.
* **Result:**
  + Plot of SHC versus Fermi energy.

**3. Notes**

1. **Directory structure:**
   * Ensure output data from VASP/Fleur is stored in accessible locations for MATLAB scripts.
2. **MATLAB environment:**
   * Install required MATLAB toolboxes to ensure plotting functions work properly.
3. **Data validation:**
   * Verify that input files are in the correct format (e.g., PROCAR, wannier90.win).

**USER GUIDE: ML\_Tc\_calc – Curie Temperature (Tc) Calculation**

**Introduction**  
The ML\_Tc\_calc directory contains Python scripts for calculating the Curie temperature (Tc) of magnetic materials based on results from OpenMX and Vampire. The primary script, ML\_Tc\_cal.py, automates the process of extracting magnetic system information and running simulations through OpenMX and Vampire.

**1. Directory Structure**

* **ML\_Tc\_cal.py:** Main Python script to execute the entire calculation workflow
* **run.sh:** Bash script for running ML\_Tc\_cal.py on a cluster computing system
* **examples/:** Directory with sample structures and calculation results
* **Result files (.png, .fig, .eps):** Includes graphs or illustrative data of Curie temperatures for sample materials

**2. Calculation Workflow**

**Step 1: Data Preparation and Reading**

* **Input structure files:**
  + Structure folder names are read from structures.in
  + If the file is absent, the script generates the list from the current directory and saves it
* **Data within structure folders:**
  + **POSCAR:** Contains lattice and atomic coordinate information
  + **OUTCAR:** Contains magnetic calculation results

**Step 2: Magnetic System Check**

* **Function:** check\_mag(path)
  + Analyzes atomic magnetic moments from OUTCAR
  + If the system is non-magnetic (magnetic moments below mag\_thres), the calculation stops

**Step 3: OpenMX Calculations**

* **Function:** run\_step2(path)
  + Generates input.dat for OpenMX using structural information (POSCAR) and magnetic moments (OUTCAR)
  + Creates magnetic interaction matrix (Jij) based on lattice parameters and magnetic interactions

**Step 4: Vampire Simulations**

* **Function:** run\_step4(path)
  + Generates input files for Vampire:
    - vampire.UCF: Describes lattice structure
    - vampire.mat: Contains magnetic material information
    - input: Configures simulation parameters (e.g., system size, temperature range)
  + Vampire simulates magnetic properties, scans temperature, and determines the Curie temperature

**3. Key Parameters in the Script**

**a. OpenMX Parameters**

* **Wavefunctions (PAO):**
  + Standard PBE19 wavefunctions (vps\_pao, std\_pao)
* **Energy cutoff and k-point grid:**
  + cutoff = 400.0 Ry
  + kpoints\_openmx = [31, 31, 31]
* **Magnetism threshold:**
  + mag\_thres = 0.1

**b. Vampire Parameters**

* **System size:**
  + system\_size\_x = 3.0 nm, system\_size\_y = 3.0 nm, system\_size\_z = 3.0 nm
* **Temperature range:**
  + max\_temp = 1500 K, min\_temp = 0 K
* **Temperature step:**
  + For non-field-cooled simulations, the step size is 25 K

**4. Example Usage**

**a. Run the Python script**

* On a cluster system:

bash

bash run.sh

* Or directly with Python:

bash

python3 ML\_Tc\_cal.py

**b. Output**

* **Simulation results:** Saved in the step4/ directory of each structure:
  + **stdout:** Vampire simulation log
  + **output:** Curie temperature and magnetization data
* **Graphs:** .png or fig files illustrating magnetization vs. temperature curves

**5. Notes**

1. **System compatibility:**
   * Ensure OpenMX and Vampire are installed and their paths are correctly specified in the script (openmx\_link, vampire\_link)
2. **Input data:**
   * Verify that structure folders contain complete POSCAR and OUTCAR files
3. **Customization:**
   * Parameters like cutoff, kpoints\_openmx, or system\_size\_x/y/z can be adjusted in the script to suit specific systems

**USER GUIDE: ML\_get\_data\_Restful – Data Collection from Materials Project API**

**Introduction**  
The ML\_get\_data\_Restful directory contains a Python script (ML\_get\_data\_Restful.py) to collect data from the Materials Project website through its RESTful API. The program fetches information about Heusler compounds (L21 or Xa structures) and exports the data as a .csv file.

**1. Directory Structure**

* **ML\_get\_data\_Restful.py:** Main Python script for data collection
* **workdir/:** Working directory for storing auxiliary or output files
* **ML\_get\_data\_Restful.sln and ML\_get\_data\_Restful.pyproj:** Solution and project files, typically for use in integrated development environments (e.g., Visual Studio)

**2. Data Collection Workflow**

**a. Element Lists**

The script defines three preconfigured element lists:

* **X\_list:** Transition metals such as Mn, Fe, Co, Ni, etc.
* **Y\_list:** Main group and transition metals such as Mg, Ni, Cu, Ag, etc.
* **Z\_list:** Non-metals or metalloids such as Si, Ge, Sb, Bi, etc.

**b. API Query**

* **API URL:**

bash

https://www.materialsproject.org/rest/v2/materials/{composition}/vasp?API\_KEY=TpNpzhEvC1i2wBwOS

* The API retrieves information on compounds formed by elements X, Y, and Z, with the stoichiometric ratio X2YZ.
* The response is filtered to retain only structures with space groups 216 (L21) or 225 (Xa).

**c. Collected Information**

Key properties retrieved from the API include:

* Chemical formula (pretty\_formula)
* Energy (energy and energy\_per\_atom)
* Volume (volume)
* Formation energy per atom (formation\_energy\_per\_atom)
* Space group (spacegroup)
* Band gap (band\_gap)
* Density (density)
* Total magnetization (total\_magnetization)

**d. Data Export**

* Results are saved into data.csv with columns corresponding to the collected information

**3. Code Details**

**Main Function:** get\_data(X, Y, Z)

* Sends RESTful requests to the Materials Project API for compositions X2YZX\_2YZX2​YZ.
* Filters compounds with space groups 216 or 225.
* Returns a list of collected data.

**Main Loop**

* Iterates through all possible combinations of elements in X\_list, Y\_list, and Z\_list.
* Sends API requests for each combination and collects the corresponding data.

**Request Timing:** time.sleep

* Introduces a 5-second delay between requests to avoid violating API rate limits.

**4. Usage Instructions**

**Run the Script**

* Execute the Python script directly:

bash

python3 ML\_get\_data\_Restful.py

**Prerequisites Before Running**

1. **API Key:**
   * Replace API\_KEY in the path2 variable with a valid API key from the Materials Project.
2. **Required Libraries:**
   * requests: For sending HTTP requests
   * pandas: For exporting data to a CSV file
   * json: For parsing JSON responses from the API

**Install Libraries (if not already installed):**

bash

pip install requests pandas

**Output**

* The collected results are saved in data.csv.

**5. Notes**

1. **API Limitations:**
   * Materials Project may impose limits on the number of API requests. Check and adhere to the API usage policy.
2. **API Key Security:**
   * The API key is embedded directly in the code. Protect it to prevent unauthorized access.

**USER GUIDE: ML\_getdata\_OQMD – Data Collection from the OQMD Database**

**Introduction**  
The ML\_getdata\_OQMD directory contains a Python script (ML\_getdata\_OQMD.py) designed to collect data from the Open Quantum Materials Database (OQMD). The program focuses on retrieving information about Heusler compounds (Fm-3m or F-43m structures) and exporting the data into .csv files.

**1. Directory Structure**

* **ML\_getdata\_OQMD.py:** Main Python script for data collection
* **workdir/:** Working directory, used for storing auxiliary or output files
* **ML\_getdata\_OQMD.sln and ML\_getdata\_OQMD.pyproj:** Project configuration files for development environments like Visual Studio

**2. Key Features of the Script**

**Libraries Used**

* requests: Sends HTTP requests
* pandas: Processes and stores data in table format
* BeautifulSoup: Parses HTML to extract links
* unicodedata.normalize: Cleans and normalizes text strings

**Data Collection Workflow**

1. **Element Lists (X, Y, Z):**
   * **X\_list:** Transition metals such as Mn, Fe, Co, Ni, etc.
   * **Y\_list:** Transition and main group metals such as Mg, Ru, Pd, Ir, etc.
   * **Z\_list:** Non-metals or metalloids such as Si, Sb, Bi, etc.
2. **Constructing Heusler Compounds:**
   * For each combination of X, Y, Z, the script checks the compound X2YZX\_2YZX2​YZ.
3. **Querying the OQMD Database:**
   * Primary query URL:

ruby

http://oqmd.org/materials/composition/{composition}

* + If the compound exists and belongs to the desired space groups (Fm-3m or F-43m), the script fetches its details.

1. **Fetching Detailed Information:**
   * From the compound’s detailed page, the script retrieves the following properties:
     + Formation energy (formation\_energy)
     + Total energy (energy)
     + Unit cell volume (volume)
     + Total magnetization (total\_magnetization)
     + Band gap (band\_gap)
     + Local magnetic moments for elements X, Y, Z
2. **Exporting Data:**
   * Each valid combination is saved as a separate .csv file, named in the format data\_oqmd\_X2YZ.csv.

**3. Key Functions**

**search\_data(X, Y, Z)**

* Sends a query for the compound X2YZX\_2YZX2​YZ.
* Checks the compound’s space group:
  + Only retains compounds in space groups Fm-3m or F-43m.
* Returns the status and basic information like energy, volume, and magnetization.

**get\_entry\_link(id)**

* Finds the link to the detailed page of a compound using its ID.

**get\_calculation\_data(X, Y, Z, link)**

* Retrieves detailed information from the compound’s page, including energy, band gap, and local magnetic moments.

**add\_data(X, Y, Z)**

* Aggregates all steps:
  + Sends search requests.
  + If the compound is valid, collects detailed information and returns it as a dictionary.

**4. Usage Instructions**

**Run the Script**

* Execute the script using Python:

bash

python3 ML\_getdata\_OQMD.py

**Prerequisite Installation**

* Install the required libraries:

bash

pip install requests pandas beautifulsoup4

**Output**

* The collected data will be saved in .csv files, e.g.:
  + data\_oqmd\_Mn2FeSi.csv
  + data\_oqmd\_Co2NiAl.csv

**5. Notes**

1. **Network Connection:**
   * If the network is unstable, the script will retry up to 5 times before skipping the current combination.
2. **Performance:**
   * Querying each combination can be time-consuming due to the large number of combinations and wait times between requests.
3. **Maintenance:**
   * If the structure of the OQMD website changes, the script may need updates to maintain compatibility.

**USER GUIDE: ML\_makedata – Creating a Custom Database**

**Introduction**  
ML\_makedata.py is designed to create a database of materials from structural information, elemental properties, and DFT calculation results from OUTCAR files. The results are saved in a .csv file with normalized attribute columns, facilitating machine learning and data analysis tasks.

**1. Directory Structure**

* **ML\_makedata.py:** Main script for processing data
* **workdir/:** Working directory containing required files such as Elements.csv, INPUT, and subdirectories with OUTCAR files
* **ML\_makedata.sln** and **ML\_makedata.pyproj:** Project configuration files for IDEs like Visual Studio or PyCharm

**2. Data Processing Workflow**

**Input Data**

1. **INPUT File:**
   * A list of structures to process, with each line in the format:

X Y Z

where X, Y, and Z are the chemical symbols of the elements.

1. **Elements.csv:**
   * Contains chemical and physical properties of elements. Important columns include:
     + Symbol: Chemical symbol of the element
     + Atomic Radius: Atomic radius (Å)
     + Electronegativity: Electronegativity
     + Density: Density (g/cm³)
     + Valence (s, p, d, f): Valence electron counts for different orbitals
2. **OUTCAR:**
   * Contains magnetic and energy data from DFT calculations, read by the read\_outcar() function.

**Data Processing Steps**

1. **Read the INPUT File:**
   * Lists the compound structures to process.
2. **Iterate Through Each Compound:**
   * Combines information from Elements.csv and OUTCAR to compute unique features.
3. **Feature Calculation:**
   * **Elemental Information:**
     + Extracts physical and chemical properties for each element (X, Y, Z).
   * **Atomic Radius Ratios:**
     + Calculates the ratios between atomic radii of elements: RatioXY,RatioXZ,RatioYZ\text{Ratio}\_{XY}, \text{Ratio}\_{XZ}, \text{Ratio}\_{YZ}RatioXY​,RatioXZ​,RatioYZ​
   * **OUTCAR Information:**
     + Extracts lattice constant (lat), total magnetization (tot\_mag), and total energy (tot\_en).
4. **Magnetization Value Transformation:**
   * If the compound’s total valence electron count is less than 24, reverses the sign of the magnetization for consistency.
5. **Export Data:**
   * Saves all processed information into a .csv file with normalized columns.

**3. Key Functions**

1. **read\_outcar(path)**
   * Reads the OUTCAR file to extract:
     + Lattice constant (lat)
     + Local magnetic moments of elements (x1\_mag, x2\_mag, etc.)
     + Total energy (tot\_en)
2. **Element Property Extraction from Elements.csv:**
   * Retrieves properties like atomic radius, electronegativity, and valence electron counts based on the element’s symbol.
3. **Compound Data Processing:**
   * For each X2YZX\_2YZX2​YZ structure:
     + Extracts properties of X, Y, and Z.
     + Calculates atomic radius ratios

(RatioXY,RatioXZ,RatioYZ\text{Ratio}\_{XY}, \text{Ratio}\_{XZ}, \text{Ratio}\_{YZ}RatioXY​,RatioXZ​,RatioYZ​).

* + - Aggregates magnetic and energy information from OUTCAR.

1. **Data Export:**
   * Saves processed data into a .csv file in table format.

**4. Output Data Structure**

| **Column** | **Description** |
| --- | --- |
| atom\_num\_X | Atomic number of element X |
| symbol\_X | Chemical symbol of element X |
| mass\_X | Atomic mass of element X |
| period\_X | Period of X in the periodic table |
| group\_X | Group of X in the periodic table |
| stable\_phase\_X | Stable crystal phase of X |
| atomic\_radius\_X | Atomic radius of X (Å) |
| negativity\_X | Electronegativity of X |
| density\_X | Density of X (g/cm³) |
| tot\_val\_X | Total valence electron count of X |
| s\_val\_X, p\_val\_X, ... | Valence electron counts for s, p, d, f orbitals of X |
| ... | Similar attributes for elements Y, Z |
| ratio\_XY, ratio\_XZ | Atomic radius ratios between X, Y, Z |
| lat | Lattice constant of the compound |
| tot\_mag | Total magnetization |
| tot\_en | Total energy of the compound |
| formula | Chemical formula (e.g., Mn2FeSi\text{Mn}\_2\text{FeSi}Mn2​FeSi) |
| num\_valence | Total valence electron count of the compound |

**5. Usage Instructions**

**Run the Script**

* Execute the script using Python:

bash

python3 ML\_makedata.py

**Input Requirements**

* **INPUT:** List of compound structures
* **Elements.csv:** Elemental property information
* **OUTCAR:** DFT calculation results

**Output**

* **data.csv:** Database containing all aggregated information

**6. Notes**

1. **Performance:**
   * Processing time depends on the number of structures and the size of OUTCAR files.
2. **Extensibility:**
   * Additional elemental properties or custom output attributes can be incorporated into the .csv file.

**USER GUIDE: ML\_trainNN – Neural Network Modeling**

**Introduction**  
ML\_trainNN.py is a Python script designed to:

1. Prepare training data from a compiled database.
2. Train an artificial neural network (NN) model.
3. Perform feature selection for the model.
4. Predict outputs for new datasets using the trained model.

The program is flexible, supports GPU acceleration, and provides various feature selection and model training options.

**1. Workflow**

**Step 1: Data Preparation**

* **Combine Databases:**
  + Merges multiple .csv files into a single database.
* **Remove Duplicates:**
  + Keeps the lowest total energy (tot\_en) for each chemical formula (formula).
* **Data Cleaning:**
  + Removes columns with constant or unique values.
  + Converts categorical data into numerical values.

**Step 2: Feature Selection**

* **Supported Methods:**
  1. **Visualization:**
     + **Pairplot:** Shows distributions and correlations between features.
     + **Covariance Matrix:** Displays correlations through a heatmap.
  2. **Automated Feature Selection:**
     + **Backward Elimination:** Removes features based on p-values.
     + **Recursive Feature Elimination (RFE):** Uses linear regression to eliminate the least important features.
     + **LassoCV:** Uses Lasso regression with cross-validation to remove features based on alpha regularization.
     + **Random Forest:** Measures feature importance based on error reduction.
     + **Permutation Importance:** Evaluates feature importance via random permutations.
     + **SHAP Values:** Explains feature importance in the model.

**Step 3: Model Training**

* **NN Configuration:**
  + Supports multiple hidden layers and activation functions (ReLU, Sigmoid, Linear, etc.).
  + Uses Adam Optimizer with adjustable learning rate and decay step.
* **Training Workflow:**
  + Splits data into training and testing sets based on a defined ratio (ratio\_train).
  + Trains the model over multiple runs (num\_run) with a fixed number of epochs.
  + Saves the best model weights.
  + Generates plots for loss and comparisons between true and predicted values.

**Step 4: Prediction**

* Converts prediction datasets into numerical format.
* Uses the trained model to predict outputs and saves results.

**2. Configuration**

**INPUT File Settings**

Adjustable parameters in the INPUT file include:

* **Hidden Layers and Nodes:**

makefile

num\_hidden\_layer=2

num\_nn=64,32

* **Activation Functions:**

makefile

act\_func=relu,linear

* **Feature Selection Options:**

graphql

use\_lasso=True

use\_randomforest=False

* **Input and Output Names:**

makefile

input\_names=ratio\_XY ratio\_XZ tot\_mag

output\_names=tot\_en

**Required Libraries**

* **TensorFlow/Keras:** For neural network modeling
* **Scikit-learn, SHAP:** For feature selection
* **Matplotlib, Seaborn:** For visualization

Install libraries if needed:

bash

pip install tensorflow scikit-learn shap matplotlib seaborn

**3. Key Functions**

1. **Data Preparation:**
   * combineData(): Merges all .csv files in a directory into one file.
   * prepare\_data(): Removes missing values (NaN), normalizes data, and splits into training/testing sets.
2. **Feature Selection:**
   * select\_feature(): Applies feature selection methods described above.
3. **Model Training:**
   * build\_and\_compile\_model(): Builds and compiles the NN model.
   * train(): Trains the model, saves the best weights, and plots training loss.
4. **Prediction:**
   * predict(): Uses the trained model to predict outputs for new data.

**4. Key Features**

1. **High Customizability:**
   * Supports flexible configurations for hidden layers, nodes, and activation functions.
   * Automatically adjusts learning rates with a decay schedule.
2. **Robust Feature Selection:**
   * Combines statistical and machine learning-based feature selection methods.
3. **GPU Acceleration:**
   * Speeds up training when GPU is available.
4. **Visualization:**
   * Generates plots for feature correlations, training loss, and true vs. predicted values.

**5. Usage Instructions**

1. Adjust the INPUT file to configure settings.
2. Run the script:

bash

python3 ML\_trainNN.py

**Output Results**

* **Processed Databases:**
  + train\_dataset.csv, test\_dataset.csv
* **Model Weights:**
  + Saved in a directory containing weights.ckpt
* **Plots:**
  + Training loss: loss.png
  + True vs. Predicted: true\_vs\_test.png
* **Prediction Results:**
  + Saved as predict\_results.csv

**6. Applications**

* Predicting material properties like energy, magnetization, and other attributes.
* Analyzing and optimizing material property databases.

**HƯỚNG DẪN SỬ DỤNG CHƯƠNG TRÌNH TÍNH TOÁN**

**DÀNH CHO PHÂN TÍCH VẬT LIỆU**

**Hướng dẫn sử dụng CrystalPlot: Code vẽ cấu trúc ô đơn vị**

**1. Giới thiệu**

CrystalPlot là một ứng dụng mở rộng (app) được phát triển trên nền tảng MATLAB nhằm trực quan hóa cấu trúc tinh thể từ dữ liệu định dạng POSCAR. Ứng dụng này hỗ trợ người dùng tạo ra hình ảnh đồ họa chất lượng cao cho cấu trúc ô đơn vị, đặc biệt phù hợp để sử dụng trong các nghiên cứu khoa học vật liệu và xuất bản bài báo.

**Tính năng nổi bật**:

* Hỗ trợ nhập file POSCAR từ phần mềm VASP.
* Trực quan hóa cấu trúc tinh thể với đồ họa đẹp và tùy chỉnh màu sắc.
* Lưu trữ hình ảnh đồ họa chất lượng cao cho các mục đích báo cáo và xuất bản.

**2. Yêu cầu hệ thống**

* MATLAB (phiên bản 2019a trở lên).
* Gói ứng dụng **CrystalPlot.mlappinstall**.

**3. Cài đặt**

1. Mở MATLAB.
2. Chọn **Apps** trên thanh công cụ.
3. Nhấn **Install App** và chọn file CrystalPlot.mlappinstall từ thư mục chứa mã nguồn.
4. Sau khi cài đặt, ứng dụng sẽ xuất hiện trong phần **Apps** với biểu tượng của CrystalPlot.

**4. Hướng dẫn sử dụng**

1. **Khởi chạy ứng dụng**:
   * Truy cập **Apps** trên MATLAB và chọn **CrystalPlot**.
2. **Nhập dữ liệu**:
   * Nhấn **Import** trong giao diện CrystalPlot.
   * Chọn file cấu trúc POSCAR từ thư mục của bạn.
3. **Tùy chỉnh hình ảnh**:
   * Tùy chỉnh các thông số hiển thị như màu sắc, kích thước nguyên tử, và kiểu hiển thị liên kết (nếu hỗ trợ).
4. **Xuất hình ảnh**:
   * Nhấn **Export** để lưu hình ảnh với các định dạng phổ biến như PNG, JPG, hoặc SVG.

**5. Mẹo sử dụng**

* Đảm bảo rằng file POSCAR được định dạng đúng theo tiêu chuẩn của VASP để ứng dụng nhận diện chính xác cấu trúc.
* Sử dụng các công cụ zoom và xoay trong MATLAB để điều chỉnh góc nhìn trước khi lưu hình ảnh.

**6. Khắc phục lỗi phổ biến**

* **Không mở được ứng dụng**:
  + Kiểm tra xem MATLAB đã được cài đặt đúng phiên bản hay chưa.
  + Đảm bảo đã chọn đúng file CrystalPlot.mlappinstall khi cài đặt.
* **Lỗi khi nhập file POSCAR**:
  + Xác nhận rằng file POSCAR không bị lỗi và có thông tin đầy đủ về lattice vectors và atomic positions.

**7. Tham khảo**

* Để biết thêm thông tin chi tiết hoặc cần hỗ trợ, vui lòng liên hệ nhóm phát triển qua email hoặc tài liệu kèm theo sản phẩm.

**Hướng dẫn sử dụng ML\_AHC\_cal: Code tính tự động cho AHC**

**1. Giới thiệu**

ML\_AHC\_cal là một mã nguồn được viết bằng Python, hỗ trợ tính toán tự động Độ dẫn Hall bất thường (Anomalous Hall Conductivity - AHC) cho các hợp chất từ dữ liệu cấu trúc và thông số DFT. Chương trình này tích hợp các công cụ như **VASP** và **Wannier90** để thực hiện các bước tính toán chính xác và tự động hóa quy trình.

**2. Yêu cầu hệ thống**

* **Hệ điều hành**: Linux (khuyến nghị vì hỗ trợ tốt các công cụ tính toán khoa học).
* **Phần mềm cài đặt**:
  + **Python 3.x** (các thư viện cần thiết: os, shutil, numpy, re).
  + **VASP** (phiên bản 5.4.4 hoặc mới hơn) cài đặt tại đường dẫn được khai báo trong code.
  + **Wannier90** (phiên bản 3.1.0 hoặc mới hơn) cài đặt tại đường dẫn được khai báo trong code.

**3. Thông tin đầu vào**

1. **Cấu trúc thư mục**:
   * **Thư mục dự án**: Mỗi hợp chất cần tính AHC phải có một thư mục riêng chứa:
     + **step1**: Các file POSCAR, POTCAR, INCAR, KPOINTS cho bước tính DFT đầu tiên.
2. **Các tệp dữ liệu**:
   * **structures.in**: Danh sách tên thư mục chứa các cấu trúc hợp chất cần tính toán. Nếu tệp này không tồn tại, chương trình sẽ tự động liệt kê các thư mục trong đường dẫn hiện tại.
   * File POSCAR trong mỗi thư mục chứa thông tin về cấu trúc tinh thể.

**4. Quy trình thực hiện**

Quy trình tự động được chia thành 5 bước:

1. **Step 2 - Chuẩn bị dữ liệu Wannier90**:
   * Sử dụng file CHGCAR từ **step1** để tạo file **wannier90.win** với thông số lattice vector, vị trí nguyên tử, và năng lượng Fermi.
   * Tính toán SOC (Spin-Orbit Coupling) sử dụng **VASP**.
2. **Step 3 - Tối ưu hóa cấu trúc Wannier90**:
   * Tối ưu hóa các tham số của Wannier90 để cải thiện độ chính xác.
3. **Band\_w90 - Tính toán dải năng lượng bằng Wannier90**:
   * Sử dụng file tối ưu từ bước 3 để tính toán và trực quan hóa dải năng lượng sử dụng **Wannier90**.
4. **Band\_vasp - Tính toán dải năng lượng bằng VASP**:
   * Tạo file KPOINTS để thực hiện tính toán dải năng lượng qua **VASP**.
5. **Step 4 - Tính toán AHC**:
   * Sử dụng **Wannier90** để tính độ cong Berry và AHC trên lưới k-mesh.

**5. Sử dụng mã nguồn**

1. **Chuẩn bị môi trường**:
   * Đảm bảo đã cài đặt **VASP** và **Wannier90**.
   * Khai báo đường dẫn tới các tệp thực thi của VASP và Wannier90 trong mã nguồn:

python

vasp\_link = '/path/to/vasp/bin'

wannier90\_link = '/path/to/wannier90/bin'

1. **Chạy chương trình**:
   * Lưu mã nguồn vào file ML\_AHC\_cal.py.
   * Chạy chương trình:

bash

python ML\_AHC\_cal.py

1. **Đầu ra**:
   * **Kết quả AHC**: Được lưu trong tệp wannier90.wout tại thư mục step4 của từng hợp chất.
   * **Dải năng lượng Wannier90**: Tệp output từ band\_w90 để vẽ phổ năng lượng.
   * **Dải năng lượng VASP**: Tệp kết quả từ band\_vasp.

**6. Lưu ý**

* Đảm bảo cấu trúc và file đầu vào trong thư mục hợp chất đầy đủ (POSCAR, POTCAR, KPOINTS, INCAR).
* Kiểm tra file output.log để theo dõi quá trình tính toán và phát hiện lỗi.
* Các file checkpoint giúp tiếp tục tính toán từ bước trước nếu chương trình bị gián đoạn.

**7. Khắc phục lỗi phổ biến**

* **Lỗi không tìm thấy checkpoint**:
  + Đảm bảo thư mục hợp chất có các tệp kết quả từ bước trước.
* **Lỗi đường dẫn không hợp lệ**:
  + Kiểm tra và khai báo chính xác đường dẫn tới VASP và Wannier90.

**8. Kết quả**

Chương trình tự động hóa toàn bộ quy trình từ tính toán DFT đến xuất kết quả AHC, cung cấp dữ liệu độ cong Berry và phổ năng lượng, hỗ trợ trực tiếp cho các nghiên cứu tính chất Hall bất thường trong vật liệu.

**Hướng dẫn sử dụng: ML\_Combine\_OQMD.py**

**Mục đích**

ML\_Combine\_OQMD.py là mã nguồn Python dùng để tạo cơ sở dữ liệu (database) theo định dạng OQMD từ các tập dữ liệu đầu vào. Kết quả là một tập tin .csv chứa các thuộc tính vật liệu, phục vụ các nghiên cứu vật liệu và ứng dụng machine learning.

**Yêu cầu**

1. **Môi trường:**
   * Python (phiên bản >= 3.8).
   * Các thư viện Python: pandas, numpy.
2. **Thư mục và tệp cần thiết:**
   * Thư mục databases: chứa các tệp .csv với thông tin vật liệu từ các nguồn dữ liệu khác nhau.
   * Tệp Elements.csv: chứa thông tin nguyên tố hóa học (bao gồm các thuộc tính như số nguyên tử, bán kính nguyên tử, độ âm điện,...).

**Cách sử dụng**

1. **Chuẩn bị dữ liệu:**
   * Đảm bảo thư mục databases chứa các tệp .csv đầu vào cần kết hợp.
   * Đảm bảo tệp Elements.csv có đầy đủ thông tin các nguyên tố liên quan.
2. **Chạy chương trình:**
   * Mở terminal hoặc IDE hỗ trợ Python.
   * Điều hướng đến thư mục chứa tệp ML\_Combine\_OQMD.py.
   * Thực thi lệnh:

bash

python ML\_Combine\_OQMD.py

1. **Quy trình xử lý trong mã:**
   * **Kết hợp dữ liệu (combineData):**
     + Gộp tất cả các tệp .csv trong thư mục databases thành một tệp oqmd\_data.csv.
   * **Xử lý dữ liệu (processing\_data):**
     + Loại bỏ cột band\_gap (nếu có).
     + Chuyển đổi thể tích thành hằng số mạng (lattice constant).
     + Thay giá trị NaN trong cột moment từ bằng 0.
     + Kết quả được lưu trong processed\_oqmd\_data.csv.
   * **Tạo dữ liệu đầu vào (make\_input):**
     + Đọc dữ liệu từ processed\_oqmd\_data.csv và kết hợp thông tin từ Elements.csv.
     + Tính toán các thuộc tính liên quan đến nguyên tố X, Y, Z (số nguyên tử, bán kính nguyên tử, độ âm điện,...).
     + Tính toán các tỉ lệ bán kính nguyên tử giữa X, Y, và Z (ratio\_XY, ratio\_XZ, ratio\_YZ).
     + Lưu dữ liệu đầy đủ vào oqmd\_data\_plus.csv.

**Kết quả**

* **Đầu ra:**
  + oqmd\_data.csv: Tệp .csv chứa dữ liệu từ thư mục databases được gộp lại.
  + processed\_oqmd\_data.csv: Tệp .csv sau khi dữ liệu được xử lý.
  + oqmd\_data\_plus.csv: Tệp .csv chứa đầy đủ thông tin vật liệu, được định dạng theo kiểu OQMD.

**Lưu ý**

* Nếu tệp hoặc thư mục cần thiết bị thiếu, chương trình sẽ không hoạt động chính xác. Hãy kiểm tra dữ liệu đầu vào trước khi chạy.
* Dữ liệu đầu ra có thể được sử dụng cho các nghiên cứu tiếp theo, bao gồm machine learning hoặc phân tích vật liệu.

**Hướng dẫn sử dụng: auto\_fleur (ML\_SHC+OHC\_calc)**

**Giới thiệu**

Folder auto\_fleur chứa mã nguồn Python (auto\_shc\_wannier90\_fleur.py) được thiết kế để tính toán các đặc tính như Spin Hall Conductivity (SHC) và Orbital Hall Conductivity (OHC) tự động, sử dụng phần mềm **FLEUR** và **Wannier90**. Đây là một phần quan trọng trong việc mô phỏng vật liệu dựa trên DFT.

**Yêu cầu hệ thống**

1. **Phần mềm cần thiết:**
   * FLEUR.
   * Wannier90.
   * Python (>= 3.8).
   * Gói numpy, shutil, và xml.etree.ElementTree.
2. **Thư mục và tệp liên quan:**
   * **Tệp elements.list:** Chứa danh sách các nguyên tố và thông số liên quan (tên, cấu trúc mạng, hằng số mạng).
   * **Tệp run.sh:** Tệp bash để chạy chương trình Python.

**Cách sử dụng**

1. **Chuẩn bị dữ liệu:**
   * Đảm bảo thư mục auto\_fleur chứa đủ các tệp sau:
     + auto\_shc\_wannier90\_fleur.py
     + elements.list
     + run.sh
   * Tệp elements.list có định dạng:

php

<Tên nguyên tố> <Loại mạng (fcc/bcc/hcp)> <Hằng số mạng>

Ví dụ:

Fe bcc 2.87

Co hcp 2.51

Ni fcc 3.52

1. **Cách chạy chương trình:**
   * Sử dụng terminal để điều hướng đến thư mục auto\_fleur.
   * Thực thi lệnh sau:

bash

bash run.sh

1. **Chi tiết quá trình tính toán:**
   * **Step 1: Cấu trúc SCF.**
     + Tạo tệp POSCAR, KPOINTS và POTCAR từ dữ liệu nguyên tố.
     + Chạy DFT SCF bằng **VASP** để tối ưu hóa cấu trúc và tính toán mật độ điện tử.
   * **Step 2: Nhập liệu cho FLEUR.**
     + Dữ liệu từ bước 1 được chuyển thành tệp inp của FLEUR.
     + Tính toán SCF bằng **FLEUR**.
   * **Step 3: Tạo WF (Wannier Functions).**
     + Tạo tệp projgen\_inp để khởi tạo hàm Wannier từ FLEUR.
     + Chạy các bước projgen, prepwan90, và matrixamn.
   * **Step 4: Tối ưu WF.**
     + Chạy tối ưu hóa hàm Wannier bằng **Wannier90**.
   * **Step 5: Tính SHC và OHC.**
     + Sử dụng các thông số Wannier để tính độ dẫn spin và orbital bằng **Wannier90**.
2. **Log chương trình:**
   * Quá trình tính toán được ghi lại trong tệp output.log.
   * Sau khi tính toán hoàn thành, kết quả SHC/OHC được lưu tại thư mục step5.

**Đầu ra**

* **Thư mục chứa dữ liệu tính toán:**
  + Các thư mục step1, step2,..., step5\_tb được tạo ra để lưu kết quả từng bước.
* **Các giá trị SHC/OHC:** Lưu trong tệp kết quả trong thư mục step5 hoặc step5\_tb.

**Lưu ý**

1. **Phụ thuộc phần mềm:** Các phiên bản **VASP**, **FLEUR**, và **Wannier90** phải tương thích và được cấu hình đúng trên máy tính.
2. **Cấu trúc mạng:** Tệp elements.list cần được kiểm tra kỹ để đảm bảo đúng loại mạng và hằng số mạng.
3. **Hiệu chỉnh thông số:**
   * Có thể thay đổi giá trị encut, kmesh, và các tham số khác trong mã nguồn để phù hợp với nghiên cứu của bạn.

**Hướng dẫn sử dụng: auto\_vasp (ML\_SHC+OHC\_calc)**

**Giới thiệu**

Thư mục auto\_vasp chứa mã Python (auto\_shc\_wannier90\_vasp.py) để thực hiện tính toán tự động Spin Hall Conductivity (SHC) và Orbital Hall Conductivity (OHC) dựa trên VASP và Wannier90. Đây là công cụ mạnh mẽ để mô phỏng vật liệu thông qua các phương pháp **DFT (Density Functional Theory)** kết hợp với các mô hình Wannierized.

**1. Thành phần trong auto\_vasp:**

1. **Tệp mã nguồn chính:**
   * auto\_shc\_wannier90\_vasp.py: Mã Python chính để tự động hóa quy trình tính toán SHC/OHC.
2. **Tệp dữ liệu:**
   * elements.list: Danh sách các nguyên tố và các thông số liên quan như mạng tinh thể, hằng số mạng.
3. **Tệp lệnh thực thi:**
   * run.sh: Tệp shell để chạy mã Python.

**2. Cấu trúc elements.list**

Tệp này chứa danh sách các nguyên tố, được sử dụng để tự động hóa quy trình. Định dạng như sau:

php

<Tên nguyên tố> <Loại mạng (fcc/bcc/hcp)> <Hằng số mạng> <Dự đoán Wannier (1/0)> <Tính AHC (1/0)>

**Ví dụ:**

Fe bcc 2.87 1 1

Co hcp 2.51 0 0

Ni fcc 3.52 1 0

* **1/0 trong cột thứ tư:** Quy định có sử dụng hàm Wannier để tối ưu hóa (1: Có, 0: Không).
* **1/0 trong cột thứ năm:** Xác định có tính AHC (Anomalous Hall Conductivity) (1: Có, 0: Không).

**3. Hướng dẫn sử dụng**

**3.1. Chuẩn bị môi trường**

1. Cài đặt các phần mềm cần thiết:
   * **VASP:** Phiên bản hỗ trợ tính toán SOC (spin-orbit coupling).
   * **Wannier90:** Phiên bản >= 3.1.0.
   * **Python:** Cần các thư viện như numpy và shutil.
2. Kiểm tra đường dẫn phần mềm trong mã:
   * Đường dẫn đến vasp, wannier90, và wannier4hall được định nghĩa:

python

vasp\_link = '/home/user2/thiho/app/vasp.5.4.4\_wan3'

wannier90\_link = '/home/user2/thiho/app/wannier90\_OHC/wannier90-3.1.0'

wan4hall\_link = '/home/user2/thiho/app/wannier4hall'

**3.2. Chạy mã**

1. Điều hướng đến thư mục auto\_vasp bằng terminal:

bash

cd /path/to/auto\_vasp

1. Chạy lệnh sau để thực thi:

bash

bash run.sh

**4. Chi tiết từng bước tính toán**

**4.1. Step 1: Chuẩn bị dữ liệu (SCF Calculation)**

* Mã tự động tạo các tệp **POSCAR**, **INCAR**, **KPOINTS**, và **POTCAR**.
* Chạy SCF bằng VASP với srun.

**4.2. Step 2: Tính toán SOC và tối ưu hóa hàm Wannier**

* Tính toán SOC (spin-orbit coupling) và chuẩn bị tệp đầu vào cho Wannier90.
* Nếu cần, mã sẽ tối ưu hóa các hàm Wannier để mô phỏng trạng thái vật liệu.

**4.3. Step 3: Khởi tạo hàm Wannier**

* Tạo tệp wannier90.win và chạy tối ưu hóa hàm Wannier thông qua **wannier90**.

**4.4. Step 4: Tính SHC/OHC**

* Xác định các giá trị Spin Hall Conductivity (SHC) hoặc Anomalous Hall Conductivity (AHC) dựa trên dữ liệu Wannier.

**4.5. Step 5: Tính toán bổ sung với Wannier4Hall**

* Sử dụng công cụ wannier4hall để tính các giá trị OHC hoặc mô phỏng dải năng lượng (band structure).

**4.6. Band Structure (optional)**

* Tạo các đồ thị dải năng lượng bằng VASP hoặc Wannier90.

**5. Đầu ra**

* **Tệp log:** output.log ghi lại trạng thái từng bước tính toán.
* **Kết quả:**
  + Giá trị SHC/OHC, lưu trong thư mục step4 hoặc step4\_tb.
  + Dải năng lượng (band structure) được lưu ở band\_vasp hoặc band\_w90.

**6. Lưu ý**

1. **Kiểm tra tính khả dụng của phần mềm:**
   * Đảm bảo tất cả các phần mềm (VASP, Wannier90, v.v.) được cài đặt đúng và đường dẫn chính xác.
2. **Cấu trúc nguyên tố:** Kiểm tra tệp elements.list để đảm bảo các thông tin đầu vào chính xác.
3. **Thông số tính toán:**
   * Có thể thay đổi các tham số như kmesh, encut, hoặc num\_iter trong mã để phù hợp với mục tiêu nghiên cứu.

**Hướng dẫn sử dụng: plot\_figure (ML\_SHC+OHC\_calc)**

**Giới thiệu**

Thư mục plot\_figure chứa các tệp MATLAB hỗ trợ trực quan hóa kết quả từ các tính toán SHC/OHC được thực hiện trong thư mục auto\_fleur và auto\_vasp. Các tập lệnh MATLAB này có thể xử lý dữ liệu đầu ra từ VASP, Fleur, hoặc Wannier90 để tạo ra các đồ thị, bao gồm:

* Cấu trúc vùng Brillouin (BZ),
* Dải năng lượng (band structure),
* Spin Hall Conductivity (SHC),
* Orbital Hall Conductivity (OHC).

**1. Thành phần trong plot\_figure:**

1. **Các tập lệnh MATLAB chính:**
   * **get\_inside\_BZ.m:** Xác định các điểm k nằm trong vùng Brillouin Zone (BZ).
   * **plot\_bands\_berry\_fleur.m:** Vẽ đồ thị dải năng lượng (band structure) từ kết quả Fleur.
   * **plot\_bands\_combine\_vasp.m:** Kết hợp dữ liệu dải năng lượng từ VASP và các giá trị SHC/OHC.
   * **plot\_data.m:** Hiển thị dữ liệu đầu ra từ VASP hoặc Wannier90.
   * **plot\_scan.m:** Quét Fermi energy và vẽ kết quả SHC/OHC.
2. **Tệp dữ liệu hỗ trợ:**
   * **colormap\_ud.mat:** Lưu định nghĩa colormap để sử dụng trong các đồ thị.
3. **Tệp chức năng bổ sung:**
   * **import\_procar\_ohc.m:** Nhập dữ liệu từ tệp PROCAR hoặc các tệp tương tự để phân tích.
   * **plot\_3dBZ\_fleur.m:** Tạo đồ thị vùng Brillouin 3D từ dữ liệu Fleur.

**2. Hướng dẫn sử dụng MATLAB**

**2.1. Chuẩn bị môi trường MATLAB**

1. Mở MATLAB.
2. Thay đổi thư mục làm việc sang thư mục plot\_figure:

matlab

cd /path/to/plot\_figure

1. Tải các tệp dữ liệu (nếu cần) từ các bước tính toán trước (VD: đầu ra từ VASP, Fleur).

**2.2. Ví dụ sử dụng**

**a. Xác định các điểm k trong vùng Brillouin Zone (BZ)**

Tập lệnh: get\_inside\_BZ.m

**Cách sử dụng:**

matlab

[id, knew] = get\_inside\_BZ(kpoints, hkl, G, inv\_lat, true);

**Đầu vào:**

* kpoints: Danh sách các điểm k ban đầu (4 cột: kx, ky, kz, chỉ số hiệu lực).
* hkl: Tập hợp các vectơ mạng trong không gian đảo.
* G: Ma trận vectơ đảo từ reciprocal lattice.
* inv\_lat: Ma trận nghịch đảo của mạng cơ sở (lattice).
* conv: Xác định có cần chuyển đổi tọa độ k sang không gian reciprocal hay không (true/false).

**Đầu ra:**

* id: Chỉ số của các điểm k nằm trong BZ.
* knew: Tập hợp các điểm k đã chọn (trong không gian reciprocal).

**b. Vẽ dải năng lượng từ Fleur**

Tập lệnh: plot\_bands\_berry\_fleur.m

**Cách sử dụng:**

matlab

plot\_bands\_berry\_fleur('path\_to\_data');

**Đầu vào:**

* path\_to\_data: Đường dẫn tới tệp dữ liệu đầu ra của Fleur.

**Kết quả:**

* Đồ thị dải năng lượng với các giá trị SHC/OHC được đánh dấu (nếu có).

**c. Vẽ đồ thị dải năng lượng từ VASP**

Tập lệnh: plot\_bands\_combine\_vasp.m

**Cách sử dụng:**

matlab

plot\_bands\_combine\_vasp('path\_to\_band\_data', 'path\_to\_SHC\_data');

**Đầu vào:**

* path\_to\_band\_data: Đường dẫn tới dữ liệu dải năng lượng từ VASP.
* path\_to\_SHC\_data: Dữ liệu SHC được xuất từ Wannier90 hoặc VASP.

**Kết quả:**

* Đồ thị dải năng lượng với sự kết hợp các kết quả SHC.

**d. Hiển thị dữ liệu OHC**

Tập lệnh: plot\_data.m

**Cách sử dụng:**

matlab

plot\_data('path\_to\_OHC\_data');

**Đầu vào:**

* path\_to\_OHC\_data: Đường dẫn tới tệp dữ liệu OHC.

**Kết quả:**

* Đồ thị minh họa giá trị Orbital Hall Conductivity trong không gian reciprocal hoặc dọc theo trục năng lượng.

**e. Quét Fermi energy để phân tích SHC**

Tập lệnh: plot\_scan.m

**Cách sử dụng:**

matlab

plot\_scan('path\_to\_SHC\_scan\_data');

**Đầu vào:**

* path\_to\_SHC\_scan\_data: Tệp chứa dữ liệu SHC khi quét Fermi energy.

**Kết quả:**

* Đồ thị SHC theo Fermi energy.

**3. Ghi chú**

1. **Cấu trúc thư mục:**
   * Đảm bảo dữ liệu đầu ra từ VASP/Fleur được lưu trữ ở vị trí có thể truy cập từ MATLAB.
2. **Môi trường MATLAB:**
   * Cài đặt các toolbox MATLAB cần thiết để đảm bảo các hàm đồ thị hoạt động.
3. **Kiểm tra dữ liệu:**
   * Đảm bảo các tệp đầu vào đúng định dạng yêu cầu (VD: PROCAR, wannier90.win).

**ML\_Tc\_calc: Tính toán Nhiệt độ Curie (Tc)**

**Giới thiệu:**

Thư mục ML\_Tc\_calc chứa mã Python để tính toán nhiệt độ Curie (Tc) của vật liệu từ dựa trên các kết quả từ **OpenMX** và **Vampire**. Tập lệnh chính, ML\_Tc\_cal.py, thực hiện các bước tự động để trích xuất thông tin từ hệ thống từ tính và chạy các mô phỏng thông qua OpenMX và Vampire.

**1. Cấu trúc thư mục**

* **ML\_Tc\_cal.py**: Tập lệnh Python chính để chạy toàn bộ quy trình tính toán.
* **run.sh**: Tập lệnh Bash để chạy ML\_Tc\_cal.py trên hệ thống tính toán cụm.
* **examples/**: Thư mục chứa các ví dụ cấu trúc và kết quả tính toán.
* **Các tệp kết quả (.png, .fig, .eps)**: Bao gồm đồ thị hoặc dữ liệu minh họa nhiệt độ Curie của các vật liệu mẫu.

**2. Quy trình tính toán**

Tập lệnh ML\_Tc\_cal.py thực hiện các bước chính sau:

**Bước 1: Chuẩn bị và đọc dữ liệu**

* **Tệp cấu trúc đầu vào**:
  + Tên các thư mục cấu trúc được đọc từ tệp structures.in.
  + Nếu tệp không tồn tại, chương trình sẽ tự động lấy danh sách thư mục hiện tại và lưu lại.
* **Dữ liệu trong thư mục cấu trúc**:
  + POSCAR: Chứa thông tin mạng cơ sở (lattice) và tọa độ nguyên tử.
  + OUTCAR: Chứa kết quả tính toán từ tính.

**Bước 2: Kiểm tra hệ thống từ tính**

* **Chức năng**: check\_mag(path)
* Phân tích từ tính của các nguyên tử trong hệ thống thông qua OUTCAR.
* Nếu hệ thống không có từ tính (các moment từ thấp hơn ngưỡng mag\_thres), tính toán dừng lại.

**Bước 3: Tính toán với OpenMX**

* **Chức năng**: run\_step2(path)
* Tạo tệp input.dat cho OpenMX dựa trên thông tin cấu trúc (POSCAR) và moment từ (OUTCAR).
* Tạo ma trận tương tác từ (Jij) sử dụng thông số mạng và tương tác từ.

**Bước 4: Mô phỏng với Vampire**

* **Chức năng**: run\_step4(path)
* Tạo tệp đầu vào cho Vampire:
  + vampire.UCF: Mô tả cấu trúc mạng.
  + vampire.mat: Thông tin vật liệu từ.
  + input: Thiết lập các tham số mô phỏng (VD: kích thước hệ thống, phạm vi nhiệt độ).
* Vampire chạy mô phỏng từ tính, quét nhiệt độ và xác định nhiệt độ Curie.

**3. Thông số chính trong mã**

**a. Thông số OpenMX**

* **Hàm sóng (PAO):**
  + Các hàm sóng sử dụng chuẩn PBE19 (danh sách vps\_pao và std\_pao).
* **Cắt năng lượng và lưới k-point:**
  + cutoff = 400.0 Ry
  + kpoints\_openmx = [31, 31, 31]
* **Ngưỡng từ tính:**
  + mag\_thres = 0.1

**b. Thông số Vampire**

* **Kích thước hệ thống:**
  + system\_size\_x = 3.0 nm, system\_size\_y = 3.0 nm, system\_size\_z = 3.0 nm
* **Phạm vi nhiệt độ:**
  + max\_temp = 1500 K, min\_temp = 0 K
* **Bước nhảy nhiệt độ:**
  + Nếu không sử dụng làm lạnh bằng trường (field-cool), bước nhảy là 25 K.

**4. Ví dụ sử dụng**

**a. Chạy tập lệnh Python**

Chạy trên hệ thống máy chủ cụm:

bash

bash run.sh

Hoặc trực tiếp từ Python:

bash

python3 ML\_Tc\_cal.py

**b. Đầu ra**

* **Kết quả mô phỏng:** Lưu trong thư mục step4/ của từng cấu trúc:
  + stdout: Nhật ký quá trình chạy Vampire.
  + output: Dữ liệu nhiệt độ Curie và từ hóa.
* **Biểu đồ:** Tệp .png hoặc .fig minh họa đồ thị từ hóa theo nhiệt độ.

**5. Ghi chú**

1. **Tương thích hệ thống**:
   * Cần cài đặt **OpenMX** và **Vampire** với đường dẫn phù hợp trong mã (openmx\_link, vampire\_link).
2. **Dữ liệu đầu vào**:
   * Đảm bảo các thư mục cấu trúc chứa đầy đủ POSCAR và OUTCAR.
3. **Tuỳ chỉnh**:
   * Có thể thay đổi các thông số như cutoff, kpoints\_openmx, hoặc system\_size\_x/y/z trong mã để phù hợp với từng hệ thống.

**ML\_get\_data\_Restful: Thu thập dữ liệu từ Materials Project API**

**Giới thiệu:**

Thư mục ML\_get\_data\_Restful chứa mã Python ML\_get\_data\_Restful.py để thu thập dữ liệu từ trang web [Materials Project](https://materialsproject.org) thông qua giao diện RESTful API. Chương trình lấy dữ liệu về các hợp chất Heusler (L21 hoặc cấu trúc Xa) và xuất dữ liệu thành tệp .csv.

**1. Cấu trúc thư mục**

* **ML\_get\_data\_Restful.py**: Mã Python chính để thu thập dữ liệu.
* **workdir/**: Thư mục làm việc, có thể dùng để lưu trữ các tệp dữ liệu phụ trợ hoặc tệp đầu ra.
* **ML\_get\_data\_Restful.sln và ML\_get\_data\_Restful.pyproj**: Các tệp giải pháp và dự án liên quan, thường dành cho môi trường phát triển tích hợp (VD: Visual Studio).

**2. Quy trình thu thập dữ liệu**

**a. Danh sách các nguyên tố:**

Ba danh sách nguyên tố X, Y, và Z được định nghĩa trước:

* **X\_list**: Các nguyên tố kim loại chuyển tiếp như Mn, Fe, Co, Ni,...
* **Y\_list**: Các nguyên tố nhóm chính và kim loại chuyển tiếp như Mg, Ni, Cu, Ag,...
* **Z\_list**: Các nguyên tố phi kim hoặc bán kim như Si, Ge, Sb, Bi,...

**b. Truy vấn API**

* **URL API**:
* https://www.materialsproject.org/rest/v2/materials/{composition}/vasp?API\_KEY=TpNpzhEvC1i2wBwOS
* API lấy thông tin từ các hợp chất được tạo bởi các nguyên tố X, Y, và Z, với tỷ lệ phối tử X2YZ.
* Thông tin trả về được lọc để giữ lại các cấu trúc có nhóm không gian **216 (L21)** hoặc **225 (Xa)**.

**c. Thông tin thu thập**

Các thuộc tính chính được lấy từ API:

* Công thức hóa học (pretty\_formula)
* Năng lượng (energy và energy\_per\_atom)
* Thể tích (volume)
* Năng lượng hình thành trên mỗi nguyên tử (formation\_energy\_per\_atom)
* Nhóm không gian (spacegroup)
* Khe vùng cấm (band\_gap)
* Mật độ (density)
* Từ hóa tổng (total\_magnetization)

**d. Xuất dữ liệu**

* Kết quả được lưu vào data.csv dưới dạng bảng với các cột tương ứng với thông tin thu thập.

**3. Chi tiết mã nguồn**

**Hàm chính: get\_data(X, Y, Z)**

* Gửi yêu cầu RESTful đến API Materials Project với thành phần X2YZ.
* Lọc các hợp chất thỏa mãn nhóm không gian 216 hoặc 225.
* Trả về danh sách dữ liệu được thu thập.

**Vòng lặp chính**

* Lặp qua tất cả các tổ hợp có thể của X, Y, và Z từ danh sách được cung cấp.
* Mỗi tổ hợp gửi một yêu cầu đến API và thu thập dữ liệu tương ứng.

**Thời gian chờ (time.sleep)**

* Giữa các lần gửi yêu cầu, chương trình chờ 5 giây để tránh vi phạm giới hạn tốc độ của API.

**4. Hướng dẫn sử dụng**

**Chạy tập lệnh**

Chạy trực tiếp tập lệnh Python:

bash

python3 ML\_get\_data\_Restful.py

**Yêu cầu trước khi chạy**

1. **API Key**: Đảm bảo thay thế API\_KEY trong path2 bằng khóa API hợp lệ từ Materials Project.
2. **Thư viện cần thiết**:
   * requests: Gửi yêu cầu HTTP.
   * pandas: Xuất dữ liệu ra tệp CSV.
   * json: Phân tích cú pháp dữ liệu trả về từ API.

Cài đặt các thư viện nếu chưa có:

bash

pip install requests pandas

**Đầu ra**

* Kết quả thu thập được lưu trong data.csv.

**5. Ghi chú**

1. **Giới hạn API**:
   * Materials Project có thể giới hạn số lượng yêu cầu API. Đảm bảo kiểm tra và tuân thủ giới hạn sử dụng API.
2. **Bảo mật API Key**:
   * API Key được nhúng trực tiếp trong mã, cần bảo vệ để tránh lạm dụng.

**ML\_getdata\_OQMD: Thu thập dữ liệu từ cơ sở dữ liệu OQMD**

**Giới thiệu**

Thư mục ML\_getdata\_OQMD chứa mã Python ML\_getdata\_OQMD.py để thu thập dữ liệu từ cơ sở dữ liệu [OQMD (Open Quantum Materials Database)](http://oqmd.org). Chương trình này tập trung lấy dữ liệu của các hợp chất Heusler (cấu trúc Fm-3m hoặc F-43m), xuất thông tin ra các tệp .csv.

**1. Cấu trúc thư mục**

* **ML\_getdata\_OQMD.py**: Mã Python chính để thu thập dữ liệu.
* **workdir/**: Thư mục làm việc, có thể được sử dụng để lưu trữ các tệp dữ liệu.
* **ML\_getdata\_OQMD.sln và ML\_getdata\_OQMD.pyproj**: Các tệp cấu hình dự án trong môi trường phát triển như Visual Studio.

**2. Chức năng chính trong mã**

**Các thư viện được sử dụng**

* **requests**: Gửi yêu cầu HTTP.
* **pandas**: Xử lý và lưu trữ dữ liệu dưới dạng bảng.
* **BeautifulSoup**: Phân tích cú pháp HTML để trích xuất liên kết.
* **unicodedata.normalize**: Chuẩn hóa và làm sạch chuỗi văn bản.

**Quy trình thu thập dữ liệu**

1. **Danh sách nguyên tố X, Y, Z**:
   * **X\_list**: Kim loại chuyển tiếp như Mn, Fe, Co, Ni,...
   * **Y\_list**: Kim loại chuyển tiếp và nhóm chính như Mg, Ru, Pd, Ir,...
   * **Z\_list**: Phi kim hoặc bán kim như Si, Sb, Bi,...
2. **Xây dựng hợp chất Heusler**:
   * Với mỗi tổ hợp X, Y, Z, chương trình kiểm tra hợp chất X₂YZ.
3. **Truy vấn cơ sở dữ liệu OQMD**:
   * Địa chỉ truy vấn chính:

http://oqmd.org/materials/composition/{composition}.

* + Nếu hợp chất tồn tại và thuộc nhóm không gian mong muốn (**Fm-3m hoặc F-43m**), chương trình tiếp tục thu thập chi tiết.

1. **Lấy thông tin chi tiết**:
   * Từ trang thông tin chi tiết của hợp chất, mã lấy các thuộc tính sau:
     + **Năng lượng hình thành** (formation\_energy).
     + **Năng lượng toàn phần** (energy).
     + **Thể tích ô mạng** (volume).
     + **Từ hóa tổng** (total\_magnetization).
     + **Khe vùng cấm** (band\_gap).
     + **Mômen từ cục bộ của các nguyên tố X, Y, Z**.
2. **Xuất dữ liệu**:
   * Mỗi tổ hợp được tìm thấy sẽ được lưu vào tệp .csv riêng biệt, có tên data\_oqmd\_X2YZ.csv.

**3. Các hàm quan trọng**

**search\_data(X, Y, Z)**

* Gửi yêu cầu tìm kiếm hợp chất X2YZ.
* Kiểm tra nhóm không gian của hợp chất:
  + Chỉ lấy các hợp chất thuộc nhóm **Fm-3m** hoặc **F-43m**.
* Trả về trạng thái (status) và các thông tin cơ bản như năng lượng, thể tích, và từ hóa.

**get\_entry\_link(id)**

* Tìm liên kết đến trang chi tiết của hợp chất qua id.

**get\_calculation\_data(X, Y, Z, link)**

* Truy xuất thông tin chi tiết từ liên kết của hợp chất, bao gồm năng lượng, khe vùng cấm, mômen từ cục bộ.

**add\_data(X, Y, Z)**

* Tổng hợp tất cả các bước trên:
  + Gửi yêu cầu tìm kiếm.
  + Nếu hợp chất hợp lệ, thu thập thông tin chi tiết và trả về dưới dạng từ điển.

**4. Hướng dẫn sử dụng**

**Chạy tập lệnh**

Sử dụng Python để chạy chương trình:

bash

python3 ML\_getdata\_OQMD.py

**Yêu cầu cài đặt**

Cài đặt các thư viện cần thiết:

bash

pip install requests pandas beautifulsoup4

**Đầu ra**

Dữ liệu thu thập được sẽ được lưu vào các tệp .csv, ví dụ:

* data\_oqmd\_Mn2FeSi.csv
* data\_oqmd\_Co2NiAl.csv

**5. Ghi chú**

1. **Kết nối mạng**:
   * Nếu kết nối mạng không ổn định, mã sẽ thử lại tối đa 5 lần trước khi bỏ qua tổ hợp hiện tại.
2. **Hiệu năng**:
   * Quá trình truy vấn từng tổ hợp nguyên tố có thể mất thời gian vì số lượng tổ hợp lớn và chờ đợi giữa các yêu cầu.
3. **Tính bảo trì**:
   * Nếu cấu trúc của trang web OQMD thay đổi, mã có thể cần được cập nhật.

**ML\_makedata: Tạo cơ sở dữ liệu tùy chỉnh**

**Giới thiệu**

Tập tin ML\_makedata.py được thiết kế để tạo cơ sở dữ liệu từ các hợp chất dựa trên thông tin cấu trúc, thuộc tính nguyên tố, và kết quả tính toán DFT từ tệp OUTCAR. Kết quả được lưu trong một tệp .csv với các cột thuộc tính được chuẩn hóa, giúp thuận tiện cho các bài toán học máy (machine learning) và phân tích dữ liệu.

**1. Cấu trúc thư mục**

* **ML\_makedata.py**: Tập tin mã chính để xử lý dữ liệu.
* **workdir/**: Thư mục làm việc, có thể chứa các tệp dữ liệu cần thiết như Elements.csv, INPUT, và các thư mục con chứa tệp OUTCAR.
* **ML\_makedata.sln và ML\_makedata.pyproj**: Tệp cấu hình dự án cho IDE như Visual Studio hoặc PyCharm.

**2. Quy trình xử lý**

**Nhập dữ liệu**

1. **Tệp INPUT**:
   * Danh sách các cấu trúc cần xử lý, mỗi dòng có định dạng:

Sao chép mã

X Y Z

Trong đó, X, Y, Z là ký hiệu hóa học của các nguyên tố.

1. **Tệp Elements.csv**:
   * Chứa thuộc tính hóa học và vật lý của các nguyên tố (số nguyên tử, khối lượng, bán kính nguyên tử,...).
   * Các cột quan trọng bao gồm:
     + **Symbol**: Ký hiệu nguyên tố.
     + **Atomic Radius**: Bán kính nguyên tử.
     + **Electronegativity**: Độ âm điện.
     + **Density**: Khối lượng riêng.
     + **Valence (s, p, d, f)**: Số electron hóa trị của các phân lớp.
2. **Tệp OUTCAR**:
   * Chứa thông tin từ tính và năng lượng tính toán DFT, được đọc bởi hàm read\_outcar().

**Xử lý dữ liệu**

1. **Đọc tệp INPUT**:
   * Liệt kê các cấu trúc hợp chất cần xử lý.
2. **Duyệt qua từng hợp chất**:
   * Kết hợp thông tin từ Elements.csv và dữ liệu từ OUTCAR để tính toán các thuộc tính đặc trưng.
3. **Tính toán đặc trưng**:
   * **Thông tin nguyên tố**:
     + Thuộc tính vật lý và hóa học của từng nguyên tố (X, Y, Z).
   * **Tỷ lệ bán kính nguyên tử**:
     + Tính tỷ lệ bán kính nguyên tử giữa các nguyên tố: RatioXY,RatioXZ,RatioYZ\text{Ratio}\_{XY}, \text{Ratio}\_{XZ}, \text{Ratio}\_{YZ}RatioXY​,RatioXZ​,RatioYZ​.
   * **Thông tin từ OUTCAR**:
     + Lấy hằng số mạng (lat), mômen từ (tot\_mag), năng lượng toàn phần (tot\_en).
4. **Chuyển đổi giá trị từ hóa**:
   * Nếu tổng số electron hóa trị của hợp chất nhỏ hơn 24, đảo ngược dấu của mômen từ để đảm bảo tính đồng nhất.
5. **Xuất dữ liệu**:
   * Lưu tất cả thông tin vào tệp .csv với các cột được chuẩn hóa.

**3. Các hàm chính**

**read\_outcar(path)**

* Đọc tệp OUTCAR để trích xuất:
  + **Hằng số mạng (lat)**.
  + **Mômen từ cục bộ của các nguyên tố** (x1\_mag, x2\_mag,...).
  + **Năng lượng toàn phần (tot\_en)**.

**Tính toán thuộc tính từ Elements.csv**

* Các thuộc tính như bán kính nguyên tử, độ âm điện, và valence được trích xuất từ Elements.csv dựa trên ký hiệu nguyên tố (Symbol).

**Xử lý dữ liệu hợp chất**

* Với mỗi cấu trúc X2YZ:
  + Trích xuất thông tin của nguyên tố X,Y,Z.
  + Tính toán tỷ lệ bán kính nguyên tử (RatioXY,RatioXZ,RatioYZ\text{Ratio}\_{XY}, \text{Ratio}\_{XZ}, \text{Ratio}\_{YZ}RatioXY​,RatioXZ​,RatioYZ​).
  + Tổng hợp thông tin từ tính và năng lượng từ OUTCAR.

**Lưu dữ liệu**

* Xuất dữ liệu ra tệp .csv với cấu trúc bảng.

**4. Cấu trúc dữ liệu đầu ra**

**Các cột trong tệp data.csv**

| **Cột** | **Mô tả** |
| --- | --- |
| atom\_num\_X | Số nguyên tử của nguyên tố X |
| symbol\_X | Ký hiệu hóa học của nguyên tố X |
| mass\_X | Khối lượng nguyên tử của X |
| period\_X | Chu kỳ của X trong bảng tuần hoàn |
| group\_X | Nhóm của X trong bảng tuần hoàn |
| stable\_phase\_X | Pha tinh thể ổn định của X |
| atomic\_radius\_X | Bán kính nguyên tử của X (Å) |
| negativity\_X | Độ âm điện của X |
| density\_X | Khối lượng riêng của X (g/cm³) |
| tot\_val\_X | Tổng số electron hóa trị của X |
| s\_val\_X, p\_val\_X,... | Số electron hóa trị của các phân lớp (s, p, d, f) của X |
| ... | Tương tự cho các nguyên tố Y, Z |
| ratio\_XY, ratio\_XZ | Tỷ lệ bán kính nguyên tử giữa X, Y, Z |
| lat | Hằng số mạng của hợp chất |
| tot\_mag | Mômen từ tổng |
| tot\_en | Năng lượng toàn phần của hợp chất |
| formula | Công thức hóa học (vd: Mn₂FeSi) |
| num\_valence | Tổng số electron hóa trị của hợp chất |

**5. Hướng dẫn sử dụng**

**Chạy chương trình**

Sử dụng Python để chạy tập lệnh:

bash

python3 ML\_makedata.py

**Yêu cầu tệp đầu vào**

* **INPUT**: Danh sách các cấu trúc hợp chất.
* **Elements.csv**: Thông tin hóa học của các nguyên tố.
* **OUTCAR**: Kết quả tính toán DFT.

**Đầu ra**

* **Tệp data.csv**: Cơ sở dữ liệu chứa tất cả thông tin đã tổng hợp.

**6. Ghi chú**

* **Hiệu năng**:
  + Quá trình xử lý phụ thuộc vào số lượng cấu trúc và kích thước tệp OUTCAR.
* **Mở rộng**:
  + Có thể thêm thuộc tính nguyên tố hoặc tùy chỉnh các thuộc tính đầu ra trong tệp CSV.

**ML\_trainNN: Mô hình hóa sử dụng mạng nơ-ron nhân tạo (NN)**

**Giới thiệu**

Tập tin ML\_trainNN.py cung cấp các công cụ để:

1. Chuẩn bị dữ liệu huấn luyện từ cơ sở dữ liệu tổng hợp.
2. Huấn luyện mô hình mạng nơ-ron nhân tạo (NN).
3. Lựa chọn đặc trưng (feature selection) cho mô hình.
4. Dự đoán đầu ra cho tập dữ liệu mới bằng mô hình đã huấn luyện.

Chương trình linh hoạt, hỗ trợ GPU và cung cấp nhiều phương pháp lựa chọn đặc trưng và huấn luyện mô hình.

**1. Quy trình xử lý**

**Bước 1: Chuẩn bị dữ liệu**

* **Kết hợp cơ sở dữ liệu (combineData)**:
  + Tập hợp nhiều tệp .csv vào một cơ sở dữ liệu duy nhất.
* **Loại bỏ giá trị trùng lặp**:
  + Giữ giá trị năng lượng nhỏ nhất (tot\_en) cho mỗi công thức hóa học (formula).
* **Xử lý dữ liệu**:
  + Loại bỏ các cột không có giá trị thay đổi hoặc giá trị duy nhất.
  + Chuyển đổi dữ liệu ký hiệu (object) thành số nguyên.

**Bước 2: Lựa chọn đặc trưng**

* **Các phương pháp hỗ trợ**:
  1. **Trực quan hóa**:
     + **Biểu đồ phân phối (pairplot)**: Hiển thị phân phối và mối tương quan giữa các đặc trưng.
     + **Ma trận hiệp phương sai (covariance matrix)**: Hiển thị mối quan hệ giữa các đặc trưng qua hệ số tương quan.
  2. **Lựa chọn đặc trưng tự động**:
     + **Backward Elimination**: Loại bỏ các đặc trưng dựa trên giá trị p-value.
     + **Recursive Feature Elimination (RFE)**: Sử dụng hồi quy tuyến tính để loại bỏ đặc trưng ít quan trọng nhất.
     + **LassoCV**: Phương pháp nhúng loại bỏ đặc trưng dựa trên điều chỉnh alpha.
     + **Random Forest**: Đo lường tầm quan trọng của đặc trưng bằng độ giảm sai số.
     + **Permutation Importance**: Đánh giá ảnh hưởng của từng đặc trưng thông qua hoán vị ngẫu nhiên.
     + **SHAP Values**: Giải thích tầm quan trọng của đặc trưng trong mô hình.

**Bước 3: Huấn luyện mô hình**

* **Cấu hình mạng nơ-ron**:
  + Hỗ trợ nhiều lớp ẩn và hàm kích hoạt (ReLU, Sigmoid, Linear,...).
  + Sử dụng Adam Optimizer với tỷ lệ học (learning rate) và bước giảm (decay step) có thể điều chỉnh.
* **Quy trình huấn luyện**:
  + Chia tập dữ liệu thành tập huấn luyện và kiểm tra theo tỷ lệ quy định (ratio\_train).
  + Huấn luyện mô hình trong nhiều vòng lặp (num\_run) và số epoch cố định.
  + Kiểm tra và lưu trữ trọng số mô hình tốt nhất.
  + Vẽ biểu đồ tổn thất (loss) và so sánh giá trị thực và giá trị dự đoán.

**Bước 4: Dự đoán**

* Chuyển đổi tập dữ liệu dự đoán sang định dạng số.
* Sử dụng mô hình đã huấn luyện để dự đoán đầu ra và lưu kết quả.

**2. Cấu hình**

**Tệp INPUT**

Người dùng điều chỉnh các thông số qua tệp INPUT, với các tùy chọn:

* **Số lớp ẩn, số node trong mỗi lớp**:

makefile

Sao chép mã

num\_hidden\_layer=2

num\_nn=64,32

* **Hàm kích hoạt**:

makefile

Sao chép mã

act\_func=relu,linear

* **Các lựa chọn đặc trưng**:

graphql

Sao chép mã

use\_lasso=True

use\_randomforest=False

* **Tên đầu vào và đầu ra**:

makefile

Sao chép mã

input\_names=ratio\_XY ratio\_XZ tot\_mag

output\_names=tot\_en

**Yêu cầu thư viện**

* **TensorFlow/Keras**: Cho NN.
* **Scikit-learn, SHAP**: Cho lựa chọn đặc trưng.
* **Matplotlib, Seaborn**: Để vẽ biểu đồ.

**3. Các hàm chính**

**1. Chuẩn bị dữ liệu**

* **combineData()**:
  + Tập hợp tất cả các tệp .csv trong thư mục vào một tệp duy nhất.
* **prepare\_data()**:
  + Loại bỏ giá trị thiếu (NaN), chuẩn hóa dữ liệu, và chia tập dữ liệu.

**2. Lựa chọn đặc trưng**

* **select\_feature()**:
  + Thực hiện lựa chọn đặc trưng bằng các phương pháp đã mô tả.

**3. Huấn luyện**

* **build\_and\_compile\_model()**:
  + Xây dựng và biên dịch mô hình NN.
* **train()**:
  + Huấn luyện mô hình, lưu trọng số tốt nhất và vẽ biểu đồ tổn thất.

**4. Dự đoán**

* **predict()**:
  + Dự đoán giá trị đầu ra cho tập dữ liệu mới.

**4. Tính năng nổi bật**

1. **Tùy chỉnh cao**:
   * Hỗ trợ cấu hình số lớp ẩn, số node và các hàm kích hoạt.
   * Tự động điều chỉnh tốc độ học với lịch trình giảm tỷ lệ học.
2. **Lựa chọn đặc trưng mạnh mẽ**:
   * Sử dụng nhiều phương pháp lựa chọn đặc trưng từ thống kê đến học máy.
3. **Hỗ trợ GPU**:
   * Tăng tốc huấn luyện mô hình khi có GPU.
4. **Trực quan hóa**:
   * Vẽ biểu đồ tương quan đặc trưng, tổn thất huấn luyện, và so sánh giá trị thực với dự đoán.

**5. Hướng dẫn sử dụng**

**Chạy chương trình**

1. Điều chỉnh tệp INPUT theo yêu cầu.
2. Chạy lệnh sau:

bash

Sao chép mã

python3 ML\_trainNN.py

**Kết quả đầu ra**

* **Cơ sở dữ liệu đã xử lý**: train\_dataset.csv, test\_dataset.csv.
* **Trọng số mô hình**: Thư mục chứa tệp weights.ckpt.
* **Biểu đồ**: loss.png, true\_vs\_test.png.
* **Kết quả dự đoán**: predict\_results.csv.

**6. Ứng dụng**

* Dự đoán năng lượng, mômen từ, và các thuộc tính của vật liệu.
* Phân tích và tối ưu hóa cơ sở dữ liệu vật liệu.