Project 1: Atomic Environment Classification as a Tool for Crystal Growth and Phase Transition Visualization *[Unsupervised Learning, Molecular Descriptors]*

Reference: João Paulo Almeida de Mendonça

Objective: You will use machine learning techniques to classify atomic environments in molecular dynamics (MD) trajectories and visualize crystal growth and phase transitions, trying to get a better visualization approach that can identify phases more precisely than traditional metrics like coordination number.

Background: In simulations of crystallization or solid-solid transitions, local atomic environments shift significantly. Traditional descriptors like coordination number, effective coordination number (ECN) or common neighbor analysis (CNA) offer some insight but may not be the best for every system. By applying molecular descriptors and clustering/classification techniques, you can identify and visualize distinct atomic environments automatically.

Tools & Libraries:

- Everything can be done in a Notebook, using !pip to install packages.
- ASE or Ovito are helpful to manipulate the structures and read the MD files.
- Descriptors: Coulomb Matrix, SOAP, ACSF… All are available via DScribe, or you can use any manual feature engineering/code you want.
- CN, ECN, and CNA: Compute them as you see fit. Ovito and ASE already have some of those implemented.
- ML: Scikit-learn, PCA or t-SNE (for dimensionality reduction, if necessary), k-means or DBSCAN for the clustering.
- Visualization: matplotlib, Ovito, ase.view, VMD, POVRAY,… Be creative!

Data Provided:

- MD trajectories and structures with phase transitions (solid-solid and solid-liquid) or interfaces (nucleation).
  https://filesender.renater.fr/?s=download&token=50ae6a36-79de-4df9-9b87-bd97d4f7eb9d

Goals:

- Extract meaningful descriptors of atomic environments
- Apply clustering or classification to label phases or intermediates
- Compare with coordination/CNA analysis in clarity and interpretability
- Generate 2D plots or animations of structural evolution

Project 2: Benchmarking DFT Methods for Small Hydrocarbons *[Workflow, DFT]*

Reference: João Paulo Almeida de Mendonça

Objective: Evaluate the performance of different Density Functional Theory (DFT) methods in computing the properties of small organic molecules. Identify systematic trends in energies and electronic properties across functionals.

Background: Different DFT functionals (LDA, PBE, B3LYP, etc.) yield varying results, and their reliability can depend on molecular type and on the property that you're trying to predict. By benchmarking key properties such as atomization energy, HOMO-LUMO gaps, and Mulliken charges, we can understand how consistent different methods are and identify potential trade-offs between accuracy and computational cost.

Tools & Libraries:

- Python + ASE + PySCF: This combo will allow you to run all these simulations in a notebook. It isn't the most computational effective for production, but is very practical for implementation! We recommend this one for the practice here.
- Bash + Quantum Espresso/ORCA/GAMESS… : This is the standard combo people use in the HPC facilities. If you have experience on this approach, be free to go for it!
- Professional workflows using tools like AiiDA/PyIRON/ASE workflows... are the state of the art for this kind of application, but can be challenging for a few hours practice. We are listing it here just for your information.
- Matplotlib and Seaborn for data analysis and plotting! Be free also to visualize the molecules if you wish to do so.

Molecule Set: Use a filtered version of ase.collections.g2, keeping only molecules composed of C, H, O, and N atoms. ~10–20 molecules recommended.

Functionals to Compare: LDA, PBE, PBE0, B3LYP… If you already use a DFT approach in your work, you can add it here!
Properties to Analyze: Atomization energy, HOMO-LUMO gap, Mulliken charges, computation time and number of SCF steps (as a convergence/cost metric). Feel free to explore the outputs of your calculations and use any other property that you think are interesting.

Questions to Explore

- Do atomization energies correlate well across functionals?
- Electronic properties like HOMO-LUMO gaps are functional dependent in the same way?
- Do SCF convergence steps suggest some molecules are inherently harder to solve?
- Can we say the same for functionals? I.e., are some functionals harder to converge than others?

Project 3: Clustering and Visualizing Relationships Among 200 DFT Functionals *[Unsupervised Learning, Data visualization]*

Reference: João Paulo Almeida de Mendonça

Objective: You will explore how different density functional theory (DFT) methods relate to each other using the extensive dataset of ~200 functionals evaluated over an extensive dataset combination for main group chemistry properties (MGCDB84). Use clustering to detect families of related functionals and embed them in 2D for visualization.

Background: In an impressive review paper, Mardirossian & Head-Gordon benchmarked functionals across diverse chemical datasets for properties like non-covalent interactions, thermochemistry, barrier heights, etc. The supplementary data provides per-functional errors on each dataset. This rich, labeled dataset allows an unprecedented comparative analysis across methods.

Tools & Libraries:

- Python, pandas, NumPy
- Scikit-learn (clustering: hierarchical, k-means, DBSCAN)
- Dimensionality reduction: PCA, t-SNE, UMAP
- Visualization: matplotlib, seaborn, or Plotly

Data Provided: https://www.tandfonline.com/doi/suppl/10.1080/00268976.2017.1333644.

Goals

- Load and preprocess per-functional RMSD or MAE vectors across all datasets that compose MGCDB84. Apply clustering to try to detect groups of functionals that behave similarly.
- Reduce to 2D embedding to visually map clusters and inspect the result for relationships. Labeling the functionals or coloring the scatter plot in terms of the functional type or MRSD on targeted datasets can help. Try to interpret the clustering physically.

Challenges & Considerations:

- You will have to choose on doing or not some kind of normalization before clustering, since the errors on some quantities have different physical meanings and orders of magnitude.
- Try to optimize the clustering parameters (number of clusters of k-means, density for DBSCAN, etc.) based on some quantitative criteria and examine if this is enough sensitivity to have a clustering that is still physically relevant.
- If the clustering method or the visualization highlights outliers, try to check what makes those cases special. For instance, is this method particularly bad/good for a given property?

Project 4: Coding a DIY MLIP *[Supervised Learning, Molecular Descriptors, Molecular Dynamics]*

Reference: João Paulo Almeida de Mendonça

Objective: Your work will be doing by hand what you saw many codes doing this week: Use the MD data from this week (for instance, the one in aluminium) and convert the structures in local environment descriptors, code a neural network regressor and try to learn the energy as function of those local descriptors.

Background: Many codes in the literature already do this training process for us, but there is a value in doing it by hand: Comprehending each detail of the process, including data manipulations and metaparameters choice. Also, learning only the energies make this a problem handable in the time we have this afternoon.

Tools & Libraries:

- Everything can be done in a Notebook, using !pip to install packages.
- Use the notebooks from this week to get the MD data. You can use LAMMPS with a simple potential like EAM or even Lennard Jones.
- Descriptors: Ewald sum matrix, SOAP (via DScribe), Behler-Parrinello, ACSF, or any manual feature engineering.
- ML: Scikit-learn has a multi layer perceptron that can do the trick. Otherwise, check Torch or Tensorflow.
- Visualization: matplotlib, Ovito, ase.view, VMD, POVRAY,… Be creative!

Goals:

- Generate MD data.
- Build a full MLIP model, including descriptor and neural network.
- Do the training and check the learning curves. At this point, part of the group can work in the next goal while one takes care of testing different hyperparameters.
- Validate the result: Remember to plot the learning curve. You can validate the results by also plotting the predicted vs real energies in the validation set; and the potential energy of two atoms isolated in vacuum as function of the distance between them.

Project 5: MLIP on Aluminium complete database *[Supervised Learning, MLIP, Molecular Dynamics]*

Reference: Irina Piazza

Objective: Discriminate between physical results from ML artefact applied to an AIMD Aluminium dataset.

Background: linux command, python, code, linux/win/mac terminal

Instructions:

- Download input.nn is stored in the repository under the path MLIP_n2p2/input_data
- Download input.data for Aluminium again, but this input.data is a collection of structure of Aluminium in different conditions.
- Download the apptainer ovito image and n2p2 ovito converter to convert input.data from n2p2 format to xyz format, which can be read ovito. It helps you to visualize Alumimium structures inside input.data.
- Open the notebook used on Colab and upload your input data correctly. **Don't forget to run the calculation into a specific folder that you can define by yourself but you should be aware where you run your calculation!**
- Change input.nn hyperparameters:
    - Change global_nodes_short = 2 2 and epochs = 100
    - Change global_nodes_short = 2 2 and epochs = 1000
    - Change global_nodes_short = 2 2 and epochs = 10000
    - Change global_nodes_short = 5 5 and epochs = 100
    - Change global_nodes_short = 5 5 and epochs = 1000
    - Change global_nodes_short = 10 10 and epochs = 50
    - Change global_nodes_short = 2 2 and epochs = 100
    - Change global_nodes_short = 2 2 and epochs = 200

Have a look at how best weights change and how RDF changes compare with the results obtained previously using the animation of RDF.

Tools & Libraries: Apptainer image, Ovito, Colab

Goals:

- **To Learn a MLIP method**
- Have a look at how best weights change and how RDF changes and you shoul compare with the results.
- Highlight any ML artefact, overfitting.

Project 6: MLIP n2p2 workflow (WF) *[Supervised Learning, MLIP, Molecular Dynamics]*

Reference: Irina Piazza

Objective: Add validation step into a n2p2 workflow.

Background: linux command, python, code, linux/win/mac terminal

Instructions:

- Download the MLIP_n2p2 workflow.
- Run the main file run_simulation.py
    - You can run under the terminal and modify files using a simple note software
    - You can run using Visual studio code, that you should install if you don't have it.
- As you can see in the main there is:
    - a library that contains all information to **generate the data** md_simulation_library.py
    - A library to do the scaling nnp_scaling_manager.py
    - A library to do the training nnp_training_manager.py

You should create another library that makes the validation step.

Tools & Libraries: Apptainer image, Ovito, Colab, subprocess python

Goals:

- **To build a WF**

Project 7: Physics-Informed Neural Operators for the Allen–Cahn Equation

[Physics-Informed Neural Networks]

Reference: Chih-Kang Huang

Objective: In this exercise, you will explore different neural operator architectures—such as DeepONet, Fourier Neural Operator (FNO), and U-Net —and implement them using JAX and Equinox. The operators will then be trained within a physics-informed framework to learn solutions of the Allen–Cahn equation, a widely studied nonlinear PDE in materials science.

Background: While Physics-Informed Neural Networks (PINNs) have shown strong potential in solving PDEs, they face an important limitation: each time the initial conditions, boundary conditions, or PDE coefficients change, the network needs to be retrained. Neural Operators, on the other hand, are designed to learn mappings between function spaces and can naturally take problem parameters as inputs, making them better suited for parametric PDEs. Recent work (Li et al., 2021) introduced Physics-Informed Neural Operators (PINOs), which combine the strengths of data-driven training with physics constraints to approximate the solution operator of a whole family of parametric PDEs.

Tools & Libraries:

- Programming: Python
- Model building: JAX, Equinox, Optax
- Visualization: Matplotlib

Goals:

- Implement different neural operator architectures (DeepONet, FNO, UNet, etc.) using JAX and Equinox.
- Train the neural operators on the Allen–Cahn equation, either directly through PDE residuals or via a variational formulation such as the Deep Ritz method.
- Compare the approaches and discuss their advantages over standard PINNs in handling parametric PDE problems.

References:

- https://arxiv.org/abs/2111.03794 Li et al., Physics-Informed Neural Operator for Learning Partial Differential Equations
- Fourier Neural Operators (FNO) in JAX, Machine Learning & Simulations
- https://arxiv.org/abs/2302.13368 Li et al., Phase-Field DeepONet: Physics-informed deep operator neural network for fast simulations of pattern formation governed by gradient flows of free-energy functionals