The Journal of Open Source Software

ASSYST: A Python package for automated generation of training data for machine learning interatomic potentials

Marvin Poul [1] **and Jörg Neugebauer** [1]

1 Max Planck Institute for Sustainable Materials, Germany

DOI: 10.21105/joss.11133

Software

- Review
- Repository
- Archive

Editor: Neea Rusch

Reviewers:

- @victorhph8
- @davidmeunier79
- @shravanngoswamii

Submitted: 22 May 2026
Published: 07 November 2026

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

The accuracy of machine-learning interatomic potentials (MLIPs) depends on high-quality training data, yet existing automation tools are often monolithic and tied to specific electronic-structure codes. ASSYST is a lightweight, pure Python library that implements the Automated Small Symmetric Structure Training method. It has a modular design: small, orthogonal classes communicate via abstract interfaces and the Atomic Simulation Environment (ASE) Calculator API. The package can be easily integrated with workflow managers, e.g., Executorlib (Janssen et al., 2025) and pyiron (Janssen et al., 2019), command-line interfaces (CLI) or graphical user interfaces (GUI), enabling heterogeneous high-performance computing (HPC) deployment. Released under the BSD 3-Clause license with full documentation and continuous integration (CI) testing, ASSYST enables the flexible and extensible generation of MLIP training data across diverse research workflows.

Statement of Need

Generation of training data is the leading expense in developing new interatomic potentials, in terms of compute time as well as human interaction. Various approaches have been developed to automate this task via active learning, though the resulting code bases are often specific to certain MLIPs and particular applications. This complicates both the comparison of methods across models and systems and the construction of foundational datasets that confer wide transferability on their MLIPs.

Recently, we developed a method, the *Automated Small Symmetric Structure Training* or *ASSYST*. It produces training sets consisting of large numbers of small supercells, remaining agnostic to the MLIP architecture used to train on the generated data while yielding exceptionally transferable MLIPs (Poul et al., 2023, 2025). Meanwhile, the field is rapidly advancing: universal MLIPs now model significant parts of the periodic table (Lysogorskiy et al., 2026); extended formalisms handle additional degrees of freedom beyond geometry (Yang et al., 2026); and new reference data expand the scope of potentials (Herzog et al., 2024; Schran et al., 2019). All of these developments hint strongly that much more high-quality training data will be required in the near future, generated and ingested by new data sources and new MLIP architectures that are difficult to foresee. ASSYST therefore targets a design that facilitates interoperability with different (electronic-structure) reference data, MLIPs, or workflow managers.

State of the Field

There exist multiple software packages that aim to generate training data for MLIPs (and often the MLIPs themselves, too), offering integrated workflows to do so automatically or

semi-automatically, e.g., (Fletcher, 2026; Gelžinytė et al., 2023; Liu et al., 2025; Zeng et al., 2025; Zhang et al., 2020) aimed at solid-state materials. All of these packages are deeply integrated with the software required to run the underlying electronic structure calculations and MLIP fitting. While this can be convenient, it also makes setting up additional MLIPs or electronic data generation methods very difficult.

By comparison, ASSYST aims to be more compact and easier to extend, and to be less dependent on certain reference data or MLIPs. The integration of ASSYST has been demonstrated in multiple public workshops[1]. The corresponding supporting material illustrates the ease of integration with existing workflow tools, MLIP codes, and a GUI—something that would be cumbersome to implement with other CLI-based tools.

Software Design and Features

This package is the result of a refactoring of a previous implementation of the method (pyiron developers, 2024). The previous implementation followed an integrated design pattern, similar to alternative packages mentioned above. However, on attempting to integrate that previous code with workflow tools and execution models (Huber & Neugebauer, 2026; Janssen et al., 2025; pyiron developers, 2026), we found it difficult to cleanly express all options and capabilities of a monolithic code in these smaller and more flexible contexts without a lot of code duplication. Hence, we redesigned ASSYST from scratch by relying on:

1. small, orthogonal classes, and
2. clean, abstract interfaces between them.

The result is a lightweight code that can be quickly integrated into other packages, even without in-depth knowledge of the code base. Users of high-performance computing (HPC) systems can benefit from this particularly because the individual steps of a workflow producing an MLIP require quite different and heterogeneous compute resources. Initial preparation tasks may require minimal parallelization, if any, whereas electronic structure calculations are almost always heavily optimized across multiple compute nodes. Fitting the MLIP on the other hand is often best done on specialized GPU-accelerated nodes. In a loosely coupled code like ASSYST, users therefore have a lot of flexibility to submit different parts of the workflow to the best-fitting HPC resources, e.g., with Executorlib (Janssen et al., 2025).

This flexibility is also crucial to future-proof the package to changes in MLIP architecture or reference data. ASSYST will require very few or no changes to deal with other MLIPs or data, while more monolithic designs, e.g., those discussed in the State of the Field section, are likely to require more modifications.

In Figure 1, we show the program design that enables this flexibility. Green boxes depict individual workflow steps. Between them data flows as lists of ASE `Atoms` objects (Larsen et al., 2017). `Atoms` is a well-known and widely used data structure in computational materials science that enables the easy distribution of the individual workflow steps. In our previous GUI implementations[2], each of the boxes is represented by an interactive 'window' or 'node'. The ochre-colored boxes highlight possible and common configuration options for each step, e.g., passing `VolumeRelax("BFGS", pressure=0)` will instruct `relax(...)` to minimize the energy of each structure at zero pressure using the well-known BFGS algorithm. The violet boxes in turn mark where in the workflow ASSYST relies on external codes to generate reference data or to act as a force engine. These interactions occur either via the widely employed ASE Calculator interface or by overloading a minimal set of documented methods, which naturally integrate into the main workflow.

Finally, where ASSYST relies on random initialization or perturbation, the state of the

[1] See DPG2025 and DPG2026.
[2] See DPG2025 and DPG2026.

underlying random number generators is exposed to enable fully reproducible runs. Each generated structure is assigned an ID, and the relationships between derived structures are tracked. Together they establish the provenance of each data point in a training set. This property is verified in the test suite.

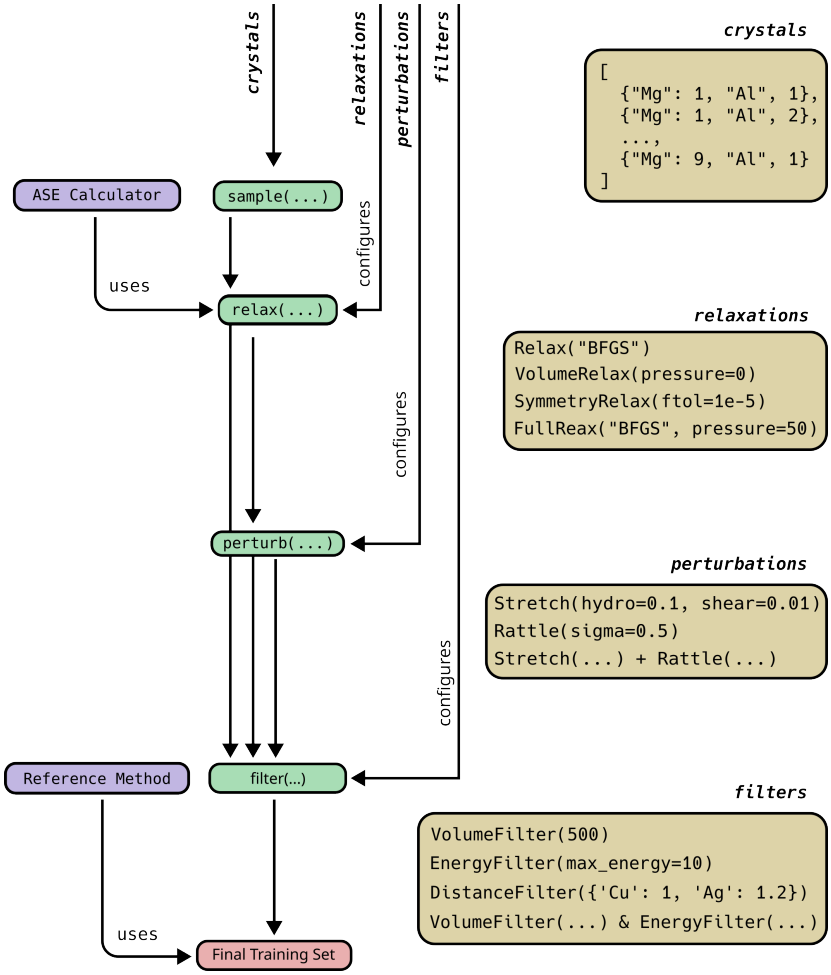


Figure 1: Outline of data and control flow in the package. Green boxes are the cleanly separated workflow steps. Arrows among them follow the flow of generated structures. Ochre boxes present example code of the small modular classes that are used to configure the individual workflow steps. Violet boxes locate where ASSYST loosely couples to external codes.

GUI Example

We demonstrate the ease of integrating ASSYST with a small GUI for a partial training set workflow in the listing below. It uses `pyironflow` version 0.1.4 and `pyiron_workflow` version 0.20.3. The modular structure allows integrators to wrap each workflow step separately according to the requirements of the surrounding framework. This way exactly the wanted functionality can be exposed or small incompatibilities glued over easily. Because each GUI element cleanly maps onto a native workflow step, GUIs or workflow managers can easily control the execution and persistence of the steps.

```python
import pyiron_workflow as pwf
from ase import Atoms
from pyironflow import PyironFlow

from assyst import crystals as cr
from assyst import perturbations as pt
from assyst.filters import DistanceFilter, FilterBase
from assyst.plot import volume_histogram

def formulas(element: str = "Cu", max_atoms: int = 4) -> cr.Formulas:
    formulas = cr.Formulas.range(element, 1, max_atoms + 1)
    return formulas

def sample(
    formulas: cr.Formulas, max_structures: int = 100, seed: int = 42
) -> list[Atoms]:
    structures = list(cr.sample(
        formulas, max_structures=max_structures,
        shuffle=True, rng=seed,
    ))
    return structures

def perturb(
    structures: list[Atoms],
    perturbation: pt.PerturbationABC,
    filters: FilterBase,
) -> list[Atoms]:
    structures = list(pt.perturb(
        structures, [perturbation], filters=filters
    ))
    return structures

def plot_volumes(structures: list[Atoms]):
    import matplotlib.pyplot as plt

    fig = plt.figure()
    volume_histogram(structures)
    return fig

wf = pwf.Workflow("assyst")
wf.formulas = pwf.node(formulas)
wf.sample = pwf.node(sample, formulas=wf.formulas)
wf.rattle = pwf.node(
    pt.Rattle, sigma=0.1, create_supercells=True, rng=42
)
wf.min_dist = pwf.node(DistanceFilter, radii=0.8)
wf.perturb = pwf.node(
    perturb, structures=wf.sample, perturbation=wf.rattle,
    filters=wf.min_dist,
)
wf.volumes = pwf.node(plot_volumes, structures=wf.perturb)

pf = PyironFlow([wf], library_roots=None)
pf.gui
```

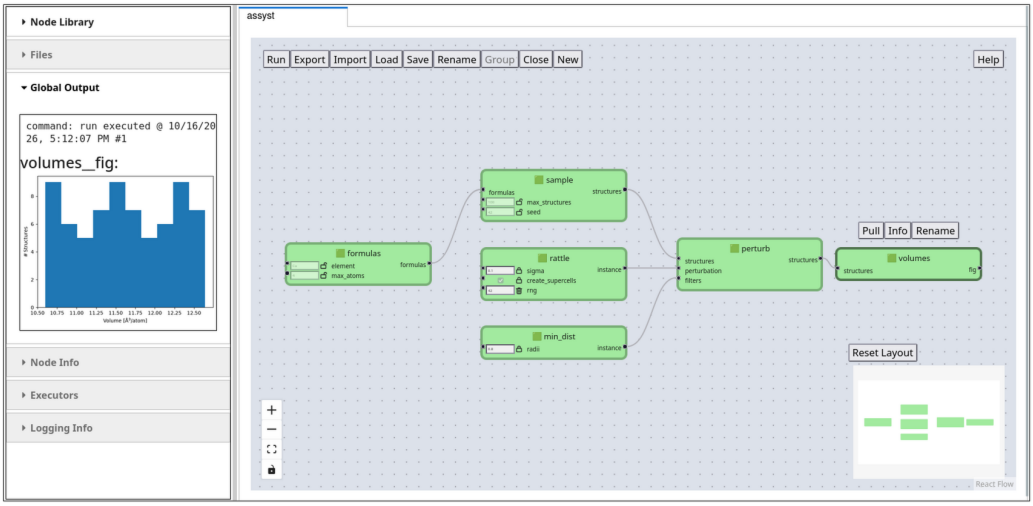


Figure 2: Example of a GUI for part of an ASSYST pipeline. Each box maps directly to a native workflow step in Figure 1.

Research Impact Statement

ASSYST implements a method to generate training data that we described in detail in earlier publications (Poul et al., 2023, 2025). While this implementation was not used in the original publications, those papers prompted multiple independent investigations (Bienvenu et al., 2025; Brunner et al., 2026; Ito et al., 2025) using derived training sets. The ASSYST package was created in part also to define a reference implementation and facilitate future method extensions like these from a common core. ASSYST has been used in multiple public teaching workshops as well.

Documentation and Source Code

Documentation including worked examples in notebooks is available at https://assyst.readthedocs.io/stable. The source code can be obtained under the BSD 3-Clause license at https://github.com/eisenforschung/assyst.

AI Usage Disclosure

Generative AI was used to generate the tests, parts of the documentation, and a small number of features. The majority of AI contributions are clearly marked as such by the commit information as authored by `google-labs-jules[bot]` (Google, 2025), `Claude <noreply@anthropic.com>` or `claude[bot]`. `Jules` was used with Gemini 2.5 (Gemini Team, Google DeepMind, 2025) and 3 (Google DeepMind, 2025); for `Claude` we used Sonnet (Anthropic, 2026b) and Opus (Anthropic, 2026a, 2026c, 2026d). Prompts are linked in PR descriptions where applicable. Additionally, most of the test suite was AI-generated but committed manually in early commits, and is therefore not marked. We ensured high quality by manual revision of all generated code and documentation. These revisions were done openly on GitHub and can be transparently reviewed in the corresponding pull request comments. The architecture of the package has been entirely designed by us, and the clean and orthogonal design supports the safe use of AI because it enables small localized changes that are easily reviewed by human eyes.

Acknowledgements

MP and JN acknowledge funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) through the Collaborative Research Center 1394 (SFB 1394, No. 409476157).

References

Anthropic. (2026a). *System card: Claude Opus 4.6*. Large language model system card. https://anthropic.com/claude-opus-4-6-system-card

Anthropic. (2026b). *System card: Claude Sonnet 4.6*. Large language model system card. https://anthropic.com/claude-sonnet-4-6-system-card

Anthropic. (2026c). *System card: Claude Opus 4.7*. Large language model system card. https://anthropic.com/claude-opus-4-7-system-card

Anthropic. (2026d). *System card: Claude Opus 5.5*. Large language model system card. https://anthropic.com/claude-opus-5-5-system-card

Bienvenu, B., Todorova, M., Neugebauer, J., Raabe, D., Mrovec, M., Lysogorskiy, Y., & Drautz, R. (2025). Development of an atomic cluster expansion potential for iron and its oxides. *npj Computational Materials*, *11*(1). https://doi.org/10.1038/s41524-025-01574-w

Brunner, F., Fioravanti, D., & Maresca, F. (2026). Liquid metal embrittlement in Fe-Zn: What can we learn from atomistic simulations? *Acta Materialia*, *308*, 121989. https://doi.org/10.1016/j.actamat.2026.121989

Fletcher, V. G. (2026). *NS_database_builder*. Accessed Nov, 2026. https://github.com/VGFletcher/NS_database_builder

Gelžinytė, E., Wengert, S., Stenczel, T. K., Heenen, H. H., Reuter, K., Csányi, G., & Bernstein, N. (2023). wfl Python toolkit for creating machine learning interatomic potentials and related atomistic simulation workflows. *The Journal of Chemical Physics*, *159*(12). https://doi.org/10.1063/5.0156845

Gemini Team, Google DeepMind. (2025). *Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities*. Google DeepMind. https://storage.googleapis.com/deepmind-media/gemini/gemini_v2_5_report.pdf

Google. (2025). *Jules using Gemini 2.5 Pro & Gemini 3 Pro*. https://jules.google/

Google DeepMind. (2025). *Gemini 3 Pro model card*. Large language model card. https://deepmind.google/models/model-cards/gemini-3-pro/

Herzog, B., Gallo, A., Hummel, F., Badawi, M., Bučko, T., Lebègue, S., Grüneis, A., & Rocca, D. (2024). Coupled cluster finite temperature simulations of periodic materials via machine learning. *npj Computational Materials*, *10*(1), 68. https://doi.org/10.1038/s41524-024-01249-y

Huber, L., & Neugebauer, J. (2026). *pyiron_workflow*. Zenodo. https://doi.org/10.5281/zenodo.19114207

Ito, K., Yokoi, T., Hyodo, K., & Mori, H. (2025). Transferability of machine-learning interatomic potential to α-Fe nanocrystalline deformation. *International Journal of Mechanical Sciences*, *291–292*, 110132. https://doi.org/10.1016/j.ijmecsci.2025.110132

Janssen, J., Surendralal, S., Lysogorskiy, Y., Todorova, M., Hickel, T., Drautz, R., & Neugebauer, J. (2019). pyiron: An integrated development environment for computational materials science. *Computational Materials Science*, *163*, 24–36. https://doi.org/10.1016/

j.commatsci.2018.07.043

Janssen, J., Taylor, M., Yang, P., Neugebauer, J., & Perez, D. (2025). Executorlib—Up-scaling Python workflows for hierarchical heterogenous high-performance computing. *The Journal of Open Source Software (JOSS)*, *10*(108). https://doi.org/10.21105/joss.07782

Larsen, A. H., Mortensen, J. J., Blomqvist, J., Castelli, I. E., Christensen, R., Dułak, M., Friis, J., Groves, M. N., Hammer, B., Hargus, C., Hermes, E. D., Jennings, P. C., Jensen, P. B., Kermode, J., Kitchin, J. R., Kolsbjerg, E. L., Kubal, J., Kaasbjerg, K., Lysgaard, S., … Jacobsen, K. W. (2017). The atomic simulation environment—a Python library for working with atoms. *Journal of Physics: Condensed Matter*, *29*(27), 273002. https://doi.org/10.1088/1361-648X/aa680e

Liu, Y., Morrow, J. D., Ertural, C., Fragapane, N. L., Gardner, J. L. A., Naik, A. A., Zhou, Y., George, J., & Deringer, V. L. (2025). An automated framework for exploring and learning potential-energy surfaces. *Nature Communications*, *16*(1), 7666. https://doi.org/10.1038/s41467-025-62510-6

Lysogorskiy, Y., Bochkarev, A., & Drautz, R. (2026). Graph atomic cluster expansion for foundational machine learning interatomic potentials. *npj Computational Materials*. https://doi.org/10.1038/s41524-026-01979-1

Poul, M., Huber, L., Bitzek, E., & Neugebauer, J. (2023). Systematic atomic structure datasets for machine learning potentials: Application to defects in magnesium. *Physical Review B*, *107*(10). https://doi.org/10.1103/PhysRevB.107.104103

Poul, M., Huber, L., & Neugebauer, J. (2025). Automated generation of structure datasets for machine learning potentials and alloys. *npj Computational Materials*, *11*(1). https://doi.org/10.1038/s41524-025-01669-4

pyiron developers. (2024). *pyiron_potentialfit*. Version 0.3.4. https://github.com/pyiron/pyiron_potentialfit

pyiron developers. (2026). *pyiron_core*. Accessed Oct, 2026. https://github.com/pyiron/pyiron_core

Schran, C., Behler, J., & Marx, D. (2019). Automated fitting of neural network potentials at coupled cluster accuracy: Protonated water clusters as testing ground. *J. Chem. Theory Comput.*, *16*(1), 88–99. https://doi.org/10.1021/acs.jctc.9b00805

Yang, J., Li, B., Mattoso, S., Abdelkawy, A., Todorova, M., & Neugebauer, J. (2026). Macro-dipole-constrained learning of atomic charges for accurate electrostatic potentials at electrochemical interfaces. *Physical Review Letters*, *136*(11), 118001. https://doi.org/10.1103/5hh2-wypm

Zeng, J., Zhang, D., Peng, A., Zhang, X., He, S., Wang, Y., Liu, X., Bi, H., Li, Y., Cai, C., Zhang, C., Du, Y., Zhu, J.-X., Mo, P., Huang, Z., Zeng, Q., Shi, S., Qin, X., Yu, Z., … Wang, H. (2025). DeePMD-kit v3: A multiple-backend framework for machine learning potentials. *J. Chem. Theory Comput.*, *21*(9), 4375–4385. https://doi.org/10.1021/acs.jctc.5c00340

Zhang, Y., Wang, H., Chen, W., Zeng, J., Zhang, L., Wang, H., & E, W. (2020). DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models. *Computer Physics Communications*, *253*, 107206. https://doi.org/10.1016/j.cpc.2020.107206