




ASIMTools: A lightweight framework for scalable and reproducible atomic simulations

Mgcini Keith Phuthi¹, Emil Annevelink², and Venkatasubramanian Viswanathan¹

¹ University of Michigan ² Carnegie Mellon University

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

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

Summary

Atomic SIMulation Tools (ASIMTools) is a lightweight workflow and simulation manager for reproducible atomistic simulations on Unix-based systems. Within the framework, simulations can be transferred across computing environments, DFT codes, interatomic potentials and atomic structures. By using in-built or user-defined python modules (called asimmodules) and utilities, users can run simulation recipes and automatically scale them on slurm based clusters or locally on their console. The core idea is to separate the dependence of the atomistic potential/calculator, the computing environment and the simulation protocol thereby allowing the same simulation to be run with different calculators, structures or on different computers with just a change of one parameter in an input file after initial setup. This is increasingly necessary as benchmarking Machine Learning Interatomic Potentials has become a core part of computational materials science. Input and output files follow a simple standard format, usually yaml, providing a simple interface that also acts as a record of the parameters used in a simulation without having to edit python scripts. The minimal set of requirements means any materials science codes can be incorporated into an ASIMTools workflow in a unified way.

Statement of need

Atomic simulations are a key component of modern day materials science in both academia and industry. However, simulation protocols and workflows used by researchers are typically difficult to transfer to systems using different inputs, codes and environments. It often involves rewriting entire scripts in different languages to change from one type of atomistic potential or atomic structure to another. This leads to poor reproducibility and inefficient transfer of code from one researcher to the next. In addition, there exists a zoo of tools and packages for atomic simulation with more being developed every day (Walsh, 2024). There is however no unifying framework that can encompass all these tools without significant software development effort. Significant effort should not be necessary because while the source of the fundamental outputs of atomistic potentials such as energy, forces etc. may differ, simulation protocols built on these outputs should converge towards the most accurate and computationally efficient. ASIMTools focuses on this last aspect by introducing asimmodules which are simply Python functions that act as simulation protocols which have no dependence on a specific atomistic potential or computational environment or atomic structure. Through iteration and community input, these simulation protocols will hopefully converge towards best practice and ensure reproducibility of simulation results.

ASIMTools is for users interested in performing atomistic calculations on UNIX-like operating systems and/or on slurm based High Performance Computing clusters. By defining simulation protocols as functions in "asimmodules", they can be easily added to the library of provided asimmodules and iterated on. This will allow the community to develop a robust set of

42 shareable simulation protocols. The flexibility of ASIMTools allows integration of any kind of
43 simulation tools such as the heavily used Atomic Simulation Environment (Larsen et al., 2017)
44 pymatgen (Ong et al., 2013), LAMMPS (Thompson et al., 2022) etc. with examples provided.
45 With the asimmodules defined, users only need to provide a set of inputs in the form of yaml
46 files that define the parameters used for each simulation and are therefore a concrete record of
47 used parameters.

48 State of the Field

49 There exist a number of popular workflow tools for atomistic simulations such as Aiiida (Huber
50 et al., 2020), Fireworks (Jain et al., 2015) and many more. These tools provide frameworks for
51 constructing complex workflows with different underlying principles. Some managers enforce
52 strict rules that ensure that data obeys FAIR principles and emphasizes data provenance and
53 reproducibility. These methods however tend to be fairly large packages with steep learning
54 curves. ASIMTools provides a simple interface as a starting point that can transform any code
55 into ASIMTools compatible code by simply wrapping it in a function that returns a Python
56 dictionary. Any such code can work in ASIMTools and with a few extra steps, the protocol
57 can be made to support an arbitrary calculator and input structure.

58 In some workflow managers, such as Atomic Simulation Recipes (Gjerding et al., 2021), once
59 workflows are built, it can often be difficult to quickly change and iterate over key parameters
60 such as the choice of atomistic calculator or structure as they are intrinsically built into the
61 code. This is particularly challenging in an age where machine learning models are becoming
62 more popular. Workflows involving machine learning interatomic potentials tend to require
63 the ability to repeat the same calculations on different examples, using different calculators
64 on different hardware iteratively. This is where the value of ASIMTools lies in contrast to
65 more established workflows. ASIMTools is not designed to replace the more powerful workflow
66 managers but rather to supplement them. This is achieved by providing unified inputs that
67 can be easily integrated into, for example, Aiiida as Python functions/asimmodules while also
68 being a stand-alone lightweight workflow manager for simpler cases.

69 Usage To-Date

70 ASIMTools has been used in the benchmarking Machine Learning Interatomic Potentials
71 (Phuthi, Yao, et al., 2024) and creating a workflow for calculation of vibrational properties of
72 solids calculations (Phuthi, Huang, et al., 2024).

73 Examples

74 We present two examples of simulation protocols, many more can be found in the ASIMTools
75 documentation.

76 Example 1: Single point calculation of energy and forces

77 Many atomic simulations involve evaluations of energies, forces, dipoles etc. of an atomic
78 configuration. In Figure. Figure 1 we show how the singlepoint asimmodule, provided in
79 ASIMTools can be used and the input files needed to run the asimmodule with arbitrary input
80 structure, calculator in an arbitrary environment.

81 While requiring four input files for such a simple calculation can seem cumbersome at first, the
82 benefit of this approach becomes apparent with repeated use and in high-throughput workflows.
83 The global calc_input.yaml and env_input.yaml configuration files describe the possible
84 calculators and environments respectively and are accessible to all simulations. Therefore
85 they only need to be modified when new environments or calculators are configured, which is

86 rare. The asimodule, a python module, forms the heart of the approach. A large library of
87 asimodules and corresponding examples are provided within ASIMTools for standard atomic
88 calculations. Users can also create their own asimodules with the only restriction being that
89 they are python modules with a correspondingly named function that returns a dictionary.
90 These asimodules should eventually converge to the most efficient and robust simulation
91 protocol over time. Finally the sim_input.yaml is the file that is used on a daily basis to run
92 the simulation and contains the parameters for the simulation protocol to be performed.

93 It becomes very simple therefore to take any simulation protocol such as the singlepoint.py
94 asimodule and to change the calculator from a cheaper Lennard-Jones potential or universal
95 force field to a more expensive DFT calculator defined in calc_input.yaml, useful for testing.
96 Similarly, changing the specified env_id changes the environment from running the protocol
97 inline in a console to running it using the slurm scheduler without the need to write a job
98 script. Just as well, the input image can be provided from any source e.g. a file, constructed
99 using ASE as in Figure. Figure 1 or downloaded from Materials Project. This modularity
100 without the need to touch any code or alter the simulation code and potentially introducing
101 bugs, is useful for benchmarks and comparing parameter choices.

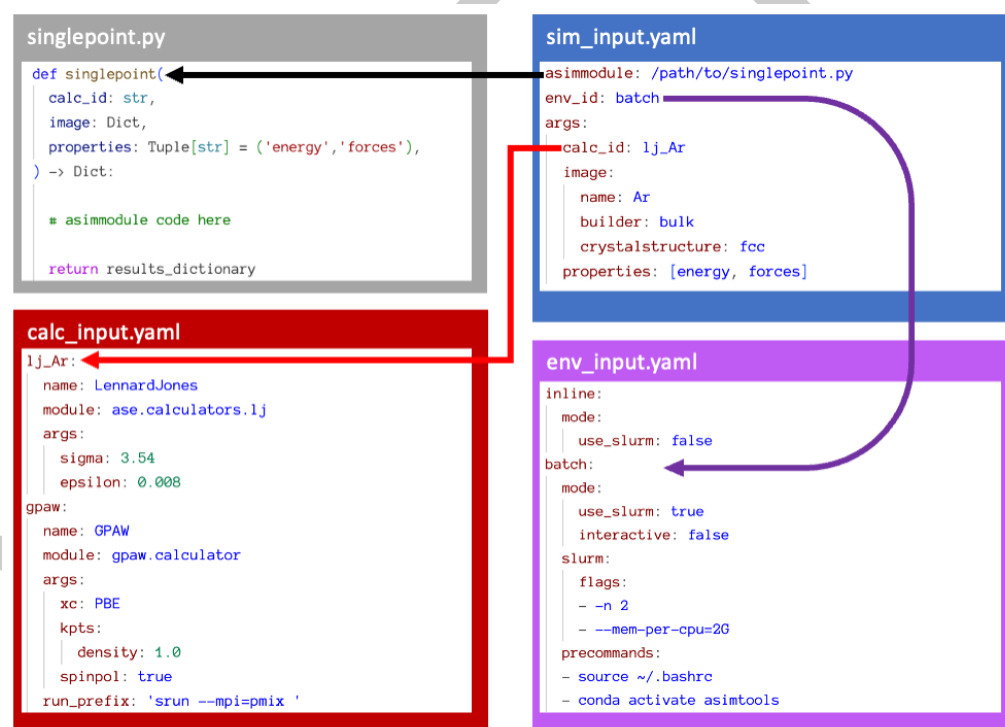


Figure 1: Schematic showing the connection between the modular input yaml files. The sim_input.yaml is the main input file which specifies the environment, calculator (if used) and asimodule to be run.

Example 2: Iterating over arguments of a simulation protocol

Given the infrastructure setup for running one simulation protocol like the singlepoint calculation, ASIMTools makes it straightforward to immediately scale. Through a set of workflow tools, simulations can be distributed in parallel across different simulation parameters as shown in the second example. ASIMTools automatically knows to submit jobs in parallel in slurm where possible without user intervention. Additionally, simulation protocols can be chained together if they have dependent results irrespective of whether they are run in slurm in different environments or in a console without user Intervention with multiple examples provided. This means that once an asimodule is written to perform a single simulation protocol using

111 ASIMTools principles, it can immediately be run with any defined input structure, calculator
112 on any computing environment at scale.

113 Below is an example `sim_input.yaml` for performing a number of singlepoint calculations on
114 structures with different lattice constants. Because the `calc_input.yaml`, `env_input.yaml` and
115 `asimmodule` were already defined before, they need not be defined again. The `key_sequence`
116 determines the key that will be distributed with the values in `array_values`. In the example
117 below, the sequence defines the lattice parameter `a` that is nested within `image` that is itself in
118 `args`.

```
119 asimmodule: workflows.sim_array
120 args:
121   key_sequence: [args, image, a]
122   array_values: [5.1,5.2,5.3,5.4,5.5]
123   template_sim_input:
124     asimmodule: singlepoint
125     args:
126       calc_id: lj_Ar
127       image:
128         name: Ar
129         crystalstructure: fcc
130       a: null # Iterate over this value
```

131 Conclusion and Availability

132 The ASIMTools package is a powerful tool for building and executing atomic simula-
133 tion protocols locally and at scale. The code is hosted on a public Github repository
134 (<https://github.com/BattModels/asimtools>) with a number of examples. Interested users
135 are encouraged to submit issues, contact developers and make pull requests, particularly for
136 adding new simulation protocols to the library.

137 Author Contribution Statement

138 Conceptualization by Keith Phuthi. Coding and development by Keith Phuthi and Emil
139 Annevelink. Paper writing by Keith Phuthi. Project management by all.

140 Acknowledgements

141 We acknowledge feedback from Kian Pu, Lance Kavalsky, Hancheng Zhao and Ziqi Wang.

142 Gjerding, M., Skovhus, T., Rasmussen, A., Bertoldo, F., Larsen, A. H., Mortensen, J. J., &
143 Thygesen, K. S. (2021). Atomic Simulation Recipes: A Python framework and library for
144 automated workflows. *Computational Materials Science*, 199, 110731. <https://doi.org/10.1016/j.commatsci.2021.110731>

146 Huber, S. P., Zoupanos, S., Uhrin, M., Talirz, L., Kahle, L., Häuselmann, R., Gresch, D., Müller,
147 T., Yakutovich, A. V., Andersen, C. W., Ramirez, F. F., Adorf, C. S., Gargiulo, F., Kumbhar,
148 S., Passaro, E., Johnston, C., Merkys, A., Cepellotti, A., Mounet, N., ... Pizzi, G. (2020).
149 AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and
150 data provenance. *Scientific Data*, 7(1), 300. <https://doi.org/10.1038/s41597-020-00638-4>

151 Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G.,
152 Rignanese, G.-M., Hautier, G., Gunter, D., & Persson, K. A. (2015). FireWorks: A dynamic
153 workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17), 5037–5059. <https://doi.org/10.1002/cpe.3505>

- 155 Larsen, A. H., Mortensen, J. J., Blomqvist, J., Castelli, I. E., Christensen, R., Duřak, M.,
156 Friis, J., Groves, M. N., Hammer, B., Hargus, C., Hermes, E. D., Jennings, P. C.,
157 Jensen, P. B., Kermode, J., Kitchin, J. R., Kolsbjerg, E. L., Kubal, J., Kaasbjerg, K.,
158 Lysgaard, S., ... Jacobsen, K. W. (2017). The atomic simulation environment—a Python
159 library for working with atoms. *Journal of Physics: Condensed Matter*, 29(27), 273002.
160 <https://doi.org/10.1088/1361-648X/aa680e>
- 161 Ong, S. P., Richards, W. D., Jain, A., Hautier, G., Kocher, M., Cholia, S., Gunter, D., Chevrier,
162 V. L., Persson, K. A., & Ceder, G. (2013). Python Materials Genomics (pymatgen): A
163 robust, open-source python library for materials analysis. *Computational Materials Science*,
164 68, 314–319. <https://doi.org/10.1016/j.commatsci.2012.10.028>
- 165 Phuthi, M. K., Huang, Y., Widom, M., & Viswanathan, V. (2024). *Vibrational Entropy*
166 *and Free Energy of Solid Lithium using Covariance of Atomic Displacements Enabled by*
167 *Machine Learning*. arXiv. <http://arxiv.org/abs/2406.15491>
- 168 Phuthi, M. K., Yao, A. M., Batzner, S., Musaelian, A., Guan, P., Kozinsky, B., Cubuk, E. D.,
169 & Viswanathan, V. (2024). Accurate Surface and Finite-Temperature Bulk Properties of
170 Lithium Metal at Large Scales Using Machine Learning Interaction Potentials. *ACS Omega*,
171 9(9), 10904–10912. <https://doi.org/10.1021/acsomega.3c10014>
- 172 Thompson, A. P., Aktulga, H. M., Berger, R., Bolintineanu, D. S., Brown, W. M., Crozier, P.
173 S., Veld, P. J. in 't, Kohlmeyer, A., Moore, S. G., Nguyen, T. D., Shan, R., Stevens, M. J.,
174 Tranchida, J., Trott, C., & Plimpton, S. J. (2022). LAMMPS - a flexible simulation tool
175 for particle-based materials modeling at the atomic, meso, and continuum scales. *Comp.*
176 *Phys. Comm.*, 271, 108171. <https://doi.org/10.1016/j.cpc.2021.108171>
- 177 Walsh, A. (2024). Open computational materials science. *Nature Materials*, 23(1), 16–17.
178 <https://doi.org/10.1038/s41563-023-01699-7>