






# foamlib: A modern Python package for working with OpenFOAM

Gabriel S. Gerlero <sup>1,2</sup> and Pablo A. Kler <sup>1,3</sup>

**1** Centro de Investigación en Métodos Computacionales (CIMEC), UNL-CONICET, Argentina **2** Universidad Nacional de Rafaela (UNRaf), Argentina **3** Departamento de Ingeniería en Sistemas de Información, Universidad Tecnológica Nacional (UTN), Facultad Regional Santa Fe, Argentina ¶ Corresponding author

DOI: [10.21105/joss.07633](https://doi.org/10.21105/joss.07633)

## Software

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

Editor: [Mojtaba Barzegari](#) 

## Reviewers:

- [@AndreWeiner](#)
- [@Failxxx](#)
- [@paugier](#)

Submitted: 13 November 2024

Published: 19 May 2025

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

## Summary

foamlib is an open-source Python package that provides a high-level, modern, object-oriented programming interface for working with OpenFOAM cases and their files. It is designed to simplify the development of workflows that involve running OpenFOAM simulations, as well as pre- and post-processing steps. foamlib understands OpenFOAM's file formats and case structures, and provides an ergonomic, Pythonic API for manipulating cases and files, including a full parser and in-place editor for the latter. It also includes support for running OpenFOAM cases asynchronously and on Slurm-based HPC clusters.

## Statement of need

foamlib is a Python package that offers a high-level programming interface for manipulating OpenFOAM cases and files, including a full standalone parser and in-place editor for the latter. It is not a thin wrapper library around existing OpenFOAM commands, nor does it provide Python bindings for OpenFOAM's C++-based code. Except for workflows that specifically involve running OpenFOAM solvers or utilities, foamlib can be used by itself without requiring an installation of OpenFOAM, e.g., allowing for pre- or post-processing steps to be performed on a different system than is used to run the simulations.

Dealing with OpenFOAM simulations from Python can be challenging, as (i) OpenFOAM uses its own non-standard file format that is not trivial to parse, and (ii) actually running OpenFOAM cases programmatically can require substantial boilerplate code for determining the correct commands to use, and then invoking said commands while accounting for other relevant considerations such as avoiding oversubscription of CPU resources when executing multiple cases at the same time. foamlib aims to address these challenges by providing a modern Python interface for interacting with OpenFOAM cases and files. By abstracting away the details of OpenFOAM's file formats, case structures, and recipes for execution, foamlib makes it easier to create Python-based workflows that involve running OpenFOAM simulations, as well as their pre- and post-processing steps. The ultimate goal of foamlib is that code for common OpenFOAM workflows, such as running parallel or HPC-based optimization loops, can be easily written in a concise, readable, and composable manner.

The closest existing software to foamlib is PyFoam ([Gschaider, 2023](#)), which is an established package that provides an alternative approach for working with OpenFOAM from Python. We believe that foamlib offers several advantages over it, notably including compatibility with current versions of Python, transparent support for fields stored in binary format, a more Pythonic fully type-hinted API with PEP 8-compliant naming, as well as support for other modern Python features such as asynchronous operations. We would also like to mention here

other Python packages with similar functionality—most notably including the ability to parse OpenFOAM output files—: `fluidfoam` (CyrilleBonamy et al., 2025), `fluidsimfoam` (Augier & Danaeifar, 2025) and `ofpp` (Xianghua, 2023).

## Features

### Requirements and installation

`foamlib` is published on PyPI and conda-forge, meaning that it can be easily installed using either `pip` or `conda`. The only prerequisite is having an installation of Python 3.7 and later, with 3.7 being chosen due to the fact that many high-performance computing (HPC) environments do not provide more up-to-date Python versions. However, in contrast with `PyFoam`, which is not currently compatible with Python releases newer than 3.11, `foamlib` works and is tested with all supported Python versions up to the current Python 3.13.

Besides the recommended packaged installs, official Docker images are also made available (with variants with or without OpenFOAM provided).

### Rationale for building a standalone library

`foamlib` is designed to be a standalone library that can be used independently of OpenFOAM itself. Notably, it does not expose or use the OpenFOAM C++ API itself. This allows `foamlib` to avoid any kind of dependence on any version or distribution of OpenFOAM, which is especially relevant considering that OpenFOAM is available in two major, incrementally diverging distributions. This design choice also allows for the development of workflows that involve OpenFOAM simulations to be performed on a different system than the one used to run the simulations. The major disadvantage of this approach is that `foamlib` needs to maintain its own implementation of an OpenFOAM file parser. However, this is mitigated by the fact that OpenFOAM's file formats are not expected to change frequently, and that `foamlib`'s parser is designed to be flexible and easily extensible.

### OpenFOAM distribution support

`foamlib` is tested with both newer and older OpenFOAM versions from both major distributions (i.e., [openfoam.com](https://openfoam.com) and [openfoam.org](https://openfoam.org)). Nevertheless, as mentioned before, OpenFOAM itself is not a required dependency of `foamlib`, being only necessary for actually running OpenFOAM solvers and utilities.

### OpenFOAM case manipulation

`foamlib` provides an object-oriented interface for interacting with OpenFOAM cases. The main classes for this purpose are `FoamCaseBase`, `FoamCase`, `AsyncFoamCase`, and `AsyncSlurmFoamCase`; all of which are presented below.

#### FoamCaseBase class

The `FoamCaseBase` class is the base class for all OpenFOAM case manipulation classes in `foamlib`. It takes the path to an OpenFOAM case on construction, and provides methods for inspecting and manipulating the case structure, whether before, during or after running the case. `FoamCaseBase` behaves as a sequence of `FoamCaseBase.TimeDirectory` objects, each representing a time directory in the case. `FoamCaseBase.TimeDirectory` objects themselves are mapping objects that provide access to the field files present in each time directory (as `FoamFieldFiles`—read below for information on file manipulation). Note, as of now, even in the case of a decomposed case, the `TimeDirectory` object will iterate over the reconstructed time directories.

### FoamCase class

The `FoamCase` class is a subclass of `FoamCaseBase` that adds functionality for running OpenFOAM cases. It is meant to be the default class used for interacting with OpenFOAM cases in `foamlib`. The following methods are present in `FoamCase` objects:

- `run()`: runs an OpenFOAM case
- `clean()`: removes files generated during the case run
- `copy()`: copies the case to a new location
- `clone()`: makes a clean copy of the case (equivalent to `copy()` followed by `clean()`—but may be more efficient)

Notably, the `run()` method can automatically determine how to run a case based on inspecting its contents and applying some heuristics. If a `run` or `Allrun` (or `Allrun-parallel`) script is present in the case directory, it will be used to run the case. Otherwise, the `run()` method can execute `blockMesh`, invoke an `Allrun.pre` script, restore the “0” directory from a backup, run `decomposePar`, and/or run the required solver (as set in the case’s `controlDict`) in serial or parallel mode, as required by the case. The behavior can be customized by passing specific arguments to the `run()` method.

By default, the `run()` method creates log files in the case directory that capture the output of the invoked commands. Besides being included within the log files, the contents of the standard error stream (`stderr`) are also stored in memory so that they can be shown in the exception message if a command fails, for easier debugging.

The `clean()` method removes all files generated during the case run. It uses a `clean` or `Allclean` script if present, or otherwise invokes logic that removes files and directories that can be re-created by running the case.

`foamlib` is also designed in a way that it can be directly used within Python-based `(All)run` and `(All)clean` scripts, without risk of calls to `run()` and `clean()` causing infinite recursion.

Besides being usable as regular methods, both `copy()` and `clone()` also permit usage as context managers (i.e., with Python’s `with` statement), which can be used to create temporary copies of a case, e.g., for testing or optimization runs. Cases created this way are automatically deleted when exiting the relevant code block.

### AsyncFoamCase class

`AsyncFoamCase` is an alternative subclass of `FoamCaseBase` that provides an asynchronous interface for running OpenFOAM cases. It is designed to work with Python’s `asyncio` library, allowing for the execution of multiple OpenFOAM cases concurrently in a single Python process.

In `AsyncFoamCase`, all of the `run()`, `clean()`, `copy()`, and `clone()` methods are asynchronous coroutines, which can be simply awaited from other asynchronous code. These methods otherwise retain the same semantics as their synchronous counterparts in `FoamCase`.

In order to avoid oversubscription of the available computational resources, `AsyncFoamCase` defines a mutable class attribute named `max_cpus` (defaulting to the number of available CPUs) that limits how many cases can be run concurrently. If a `run()` call being awaited requires more CPUs than are available, the case will not be executed immediately but will rather wait until enough CPUs are freed up by the completion of other `run()` calls.

Besides its obvious use to orchestrate parallel optimization loops, `AsyncFoamCase` can also be used to speed up testing workflows that involve running multiple OpenFOAM cases by running them concurrently. For instance, it can be used in conjunction with the `pytest` (Krekel et al., 2004) framework and the `pytest-asyncio-cooperative` (Thiart, 2024) plugin, as the authors of this paper currently do to test several OpenFOAM-based projects (Gerlero et al., 2021, 2024; Gerlero, 2024; Gerlero & Kler, 2024).

### AsyncSlurmFoamCase class

AsyncSlurmFoamCase is a direct subclass of AsyncFoamCase that adds support for running OpenFOAM cases on Slurm-based HPC clusters. When using this class, every call to `run()` by default will run the case by submitting one or more Slurm jobs to the cluster. An optional fallback keyword argument can be set to `True` to run the case locally if Slurm is not available, allowing for seamless local and cluster-based execution.

## OpenFOAM file manipulation

foamlib also provides an object-oriented interface for reading from and writing to OpenFOAM files. The main classes for this purpose are `FoamFile` and `FoamFieldFile`, which are described below.

### FoamFile class

The `FoamFile` class offers high-level facilities for reading and writing OpenFOAM files, providing an interface similar to that of a Python dict. `FoamFile` understands OpenFOAM's common input/output file formats, and is able to edit file contents in place without disrupting formatting and comments. Most types of OpenFOAM “FoamFile” files are supported, meaning that `FoamFile` can be used for both pre- and post-processing tasks.

OpenFOAM data types stored in files are mapped to built-in Python or NumPy ([Harris et al., 2020](#)) types as much as possible, making it easy to work with OpenFOAM data in Python. [Table 1](#) shows the mapping of OpenFOAM data types to Python data types with `foamlib`. Also, disambiguation between Python data types that may represent different OpenFOAM data types (e.g., a scalar value and a uniform scalar field) is resolved by `foamlib` at the time of writing by considering their contextual location within the file. The major exception to this preference for built-ins is posed by the `FoamFile.SubDict` class, which is returned for sub-dictionaries contained in `FoamFiles`, and allows for one-step modification of entries in nested dictionary structures—as is commonly required when configuring OpenFOAM cases.

For clarity and additional efficiency, `FoamFile` objects can be used as context managers to make multiple reads and writes to the same file while minimizing the number of filesystem and parsing operations required.

Finally, we note that all OpenFOAM file formats are transparently supported by `foamlib`, including ASCII, double- and single-precision binary formats, as well as compressed files.

**Table 1:** Mapping of OpenFOAM data types to Python data types with `foamlib`.

OpenFOAM	foamlib (accepts and returns)	foamlib (also accepts)
scalar	float	
vector/tensor	numpy.ndarray   list[float]	Sequence[float]
label	int	
switch	bool	
word	str	
string	str (quoted)	
multiple tokens	tuple	
list	list	Sequence
dictionary	FoamFile.SubDict   dict	Mapping
dictionary entry	tuple	
uniform field	float   numpy.ndarray	Sequence[float]
non-uniform field	numpy.ndarray	Sequence[float]   Sequence[Sequence[float]]

OpenFOAM	foamlib (accepts and returns)	foamlib (also accepts)
dimension set dimensioned	FoamFile.DimensionSet FoamFile.Dimensioned	Sequence[float]   numpy.ndarray

### FoamFieldFile class

FoamFieldFile is a convenience subclass of FoamFile that simply adds properties for accessing data expected to be present in files that represent OpenFOAM fields, such as `internal_field`, `dimensions`, and `boundary_field`.

## Documentation and examples

Examples of foamlib usage are provided in the [README file](#) of the project. Additionally, Sphinx-based documentation covering the entirety of the public API is available at [foamlib.readthedocs.io](https://foamlib.readthedocs.io).

## Implementation details

### Type hints

foamlib is fully typed using Python's type hints, which makes it easier to understand and use the library, as well as enabling automatic checks for type errors using tools like mypy ([Lehtosalo, 2024](#)).

### Parsing

foamlib contains a full parser for OpenFOAM files, which is able to understand and write to the different types of files used by OpenFOAM. The parser is implemented using the `pyparsing` ([McGuire, 2024](#)) library, which provides a powerful and flexible way to define parsing grammars.

A special case parser is internally used for non-uniform OpenFOAM fields, which can commonly contain very large amounts of data in either ASCII or binary formats. The specialized parser uses regular expressions to extract these data, which results in greatly improved parsing performance—a more than 25x speedup versus PyFoam, as measured on a MacBook Air (Apple Inc., Cupertino, Calif., USA) with an M1 processor and 8 GB of system RAM—, while not sacrificing any of the generality of the parsing grammar. For extra efficiency and convenience, these fields map to NumPy arrays in Python.

### Asynchronous support

Methods of FoamCase and AsyncFoamCase have been carefully implemented in a way that greatly avoids duplication of code between the synchronous and asynchronous versions by factoring out the common logic into a helper intermediate class.

## Continuous integration

Continuous integration of foamlib is performed automatically using GitHub Actions, and includes:

- Testing with all supported Python versions (currently 3.7 to 3.13)
- Testing with multiple OpenFOAM distributions and versions (currently 9, 12, 2006 and 2406)
- Testing on a Slurm environment
- Code coverage tracking with Codecov

- Type checking with mypy
- Code linting and formatting with Ruff
- Package building with uv

## References

- Augier, P., & Danaeifar, P. (2025). fluidsimplam: Python framework for OpenFOAM. In *Heptapod repository*. Heptapod. <https://foss.heptapod.net/fluiddyn/fluidsimplam>
- CyrilleBonamy, jchauchat, QuentinClemencot, Montellà, E. P., Höhn, P., mathieu7an, AliBernard, Gonçalves, G., MarieSkorlic, gnikit, & remichassagne. (2025). *Fluiddyn/fluid-foam: v0.2.9* (Version v0.2.9). Zenodo. <https://doi.org/10.5281/zenodo.14893673>
- Gerlero, G. S. (2024). OpenFOAM.app: Native OpenFOAM for macOS. In *GitHub repository*. GitHub. <https://github.com/gerlero/openfoam-app>
- Gerlero, G. S., Guerenstein, Z. I., Franck, N., Berli, C. L. A., & Kler, P. A. (2024). Comprehensive numerical prototyping of paper-based microfluidic devices using open-source tools. *Talanta Open*, 10, 100350. <https://doi.org/10.1016/j.talo.2024.100350>
- Gerlero, G. S., & Kler, P. A. (2024). reagency: A simple, extensible reaction model for OpenFOAM. In *GitHub repository*. GitHub. <https://github.com/gerlero/reagency>
- Gerlero, G. S., Márquez Damián, S., & Kler, P. A. (2021). electroMicroTransport v2107: Open-source toolbox for paper-based electromigrative separations. *Computer Physics Communications*, 269, 108143. <https://doi.org/10.1016/j.cpc.2021.108143>
- Gschaidner, B. (2023). PyFoam: Python utilities for OpenFOAM. In *PyPI project*. Python Package Index. <https://pypi.org/project/PyFoam/>
- Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., & others. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laughner, B., & Bruhin, F. (2004). *pytest 8.3*. <https://github.com/pytest-dev/pytest>
- Lehtosalo, J. (2024). mypy: Optional static typing for python. In *GitHub repository*. GitHub. <https://github.com/python/mypy>
- McGuire, P. (2024). *pyparsing: Python library for creating PEG parsers*. <https://github.com/pyparsing/pyparsing>
- Thiart, W. (2024). *pytest-asyncio-cooperative: Run all your asynchronous tests cooperatively*. <https://github.com/willemt/pytest-asyncio-cooperative>
- Xianghua, X. (2023). Ofpp: OpenFOAM Python Parser. In *GitHub repository*. GitHub. <https://github.com/xu-xianghua/ofpp>