The Journal of Open Source Software

Fortitude: The Fortran Linter

Liam Pattinson [1*¶], **Peter Hill** [1*], **Connor Aird** [2], **Ava Dean** [1], **Cyril Gandon** [3], **Ian McInerney** [4], **Violet Sherratt** [5], **and Vincent Vanlaer** [6]

1 University of York, United Kingdom **2** University College London, United Kingdom **3** INRAE (French National Research Institute for Agriculture, Food and Environment), France **4** Imperial College London, United Kingdom **5** Met Office, United Kingdom **6** KU Leuven, Belgium ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.10570

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @fjebaker
- @mansurjisan

Submitted: 21 April 2026
Published: 22 July 2026

In partnership with

JOSS Special Issue for RSECon26 Research Software Papers

Summary

Static analysis tools are used by software developers to better understand and improve their software, and they achieve this by examining their source code and/or binaries without running the software. A 'linter' is a type of static analysis tool that offers an opinionated critique of software beyond its syntactic validity, highlighting bug-prone coding patterns, deviations from common style guides, use of outdated features, and, generally, suggestions to improve the code in some way.

Fortitude is a linter targeting Fortran, which has hitherto lacked a standard open-source solution. It is over an order of magnitude faster than other open-source offerings, is highly customisable, has robust parsing capabilities, can automatically fix many linting issues, and can integrate either into a CI/CD pipeline or directly into a user's editor or IDE.

Statement of need

The Fortran programming language, first released in 1957 (Backus et al., 1957), is the oldest high-level programming language still in common usage today (Kedward et al., 2022), and is widely used for high-performance research applications in fields such as climatology, quantum condensed matter physics, and fusion energy. Fortran powers the majority of the software run on Archer2, which until 2024 was the most powerful supercomputer in the United Kingdom (Turner, 2022).

Due to Fortran's longevity, much of the research software in use today is written to older standards and contains a lot of technical debt. This places a large burden on the maintainers of this research software, and porting Fortran codes to more modern languages is not typically worth the resources required or the risks of introducing new bugs or performance issues.

Fortitude is a Fortran linter designed to help developers of Fortran research software improve the quality of their code. Its linting rules are grouped under categories specifying what aspect of software they intend to improve:

- Correctness: Rules to find bug-prone coding patterns, helping developers catch errors early and improve the safety of their code.
- Obsolescent: Rules to flag features marked as obsolescent in the Fortran standard and recommend refactoring strategies to avoid them.
- Modernisation: Rules to recommend newer features that promote cleaner code. These are complementary to 'obsolescent' rules, and go beyond the strict recommendations of the Fortran standard.
- Style: Rules to make Fortran code more readable and help adhere to a common set of coding conventions.

- Portability: Rules to avoid compiler- or platform-specific features in favour of portable alternatives.

As an example, the correctness rule `nonportable-shortcircuit-inquiry` detects a subtle bug in which an optional argument is both checked and used within the same logical expression, e.g:

```
if (present(arg) .and. arg > 0) then
    print *, arg
end if
```

Unlike in languages such as C and Python, the order of operations in a logical Fortran expression is neither guaranteed to run left-to-right nor to exit once the result is known, and therefore may result in all operations within the expression being executed. It is therefore possible that this could result in a reference to invalid data and a critical error. Running Fortitude over this code with the rule activated delivers a diagnostic message to the user:

```
test.f90:12:32: C161 variable inquiry `present(arg)` and use in \
        same logical expression
   |
12 |         if (present(arg) .and. arg > 0) then
   |                                ^^^ C161
13 |             print *, arg
14 |         end if
   |
```

For another example, the modernisation rule `deprecated-relational-operator` flags the use of operators such as `.lt.` and `.ge.` in place of `<` and `>=`. Though the Fortran standard permits the use of either operator style, the latter is generally considered to be more readable, and is recommended in most style guides. Fortitude will report this to the user as follows:

```
test.f90:22:17: MOD021 [*] deprecated relational operator '.lt.', \
        prefer '<' instead
   |
22 |         if (arg .lt. 0) then
   |                 ^^^^ MOD021
23 |             print *, arg
24 |         end if
   |
   = help: Use '<'
```

The symbol `[*]` indicates that Fortitude can fix this issue automatically, which can be achieved by re-running with the `--fix` flag. This feature makes it much easier for developers to introduce Fortitude into existing projects, as a large proportion of linter warnings raised by Fortitude can be corrected instantaneously.

Fortitude may be used alongside other tooling in the Fortran ecosystem. For example, its settings may be specified in the `fpm.toml` files used by the Fortran Package Manager (FPM) (Fortran-Lang Collaboration, 2026). Fortitude's editor integration using the Language Server Protocol (LSP), by which it can provide suggestions inline with the user's code, is compatible with that of FortLS (Fortran-Lang Collaboration, 2025), a plugin that can aid users in navigating a project. It can also be used alongside the fprettify (Fortran-Lang Collaboration, 2020) code formatter without generating conflicts.

State of the field

Fortitude was developed due to notable deficiencies in existing open-source Fortran linting tools, most of which are some combination of unmaintained, have few linting rules, use unreliable

Fortran parsers, and/or have poor performance. Features such as automatic fixes and editor integration were also absent. The decision to write a new linter rather than contribute to existing solutions was inspired by the higher quality of linters in other languages; it was easier to adapt those solutions to work with Fortran than to upgrade the existing Fortran linters to meet the state-of-the-art. Fortitude borrows many language-agnostic assets from the Python linter Ruff (Ruff Collaboration, 2026), including much of the command-line interface, output formats, and configuration file design. While these structural elements could be reused directly, the linting rules themselves had to be written specifically for Fortran.

Other popular open-source Fortran linting tools include:

- CamFort: Primarily a code refactoring tool, it features some linting rules (Orchard & Rice, 2013). Written in Haskell, it features its own Fortran parser.
- Flint: A linter written in Python. Uses lizard to analyse cyclomatic complexity and related metrics, and regular expressions otherwise (CERFACS Collaboration, 2022).
- Stylist: A linter written in Python and using FortranParser to generate an AST (Met Office Collaboration, 2023).

To compare Fortitude's performance to these tools, each was used to lint 72 files in the GS2 source code, a turbulence modelling tool used in magnetically-confined fusion energy research (GS2 Collaboration, 2024). Each tool was set up as follows:

- Fortitude v0.9.0: Activated all 102 rules.
- Stylist v0.5.dev (latest on GitHub): Activated all 11 rules.
- CamFort v1.3.dev (latest on GitHub): Activated the 5 rules in the `basic-checks` command.
- Flint v0.7.1: Activated the default set of 38 rules. One file had to be excluded due to a parse error.

The runtime of each tool is shown in Figure 1. In all cases, the tests were run on a simple laptop featuring a 4-core 11th generation Intel i5 CPU running at 2.60GHz. Despite implementing many more checks than the alternatives, Fortitude was found to run over 25 times more quickly than its fastest competitor.

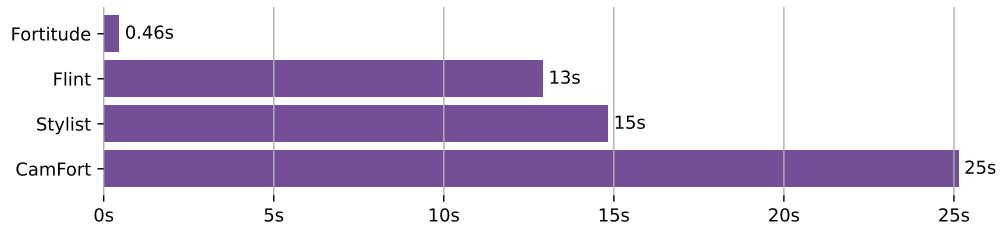

Figure 1: Time taken for Fortran linters to lint 72 files in the GS2 project.

Software design

As linting is effectively a solved problem in other languages, the design of Fortitude was guided by the reuse of existing resources. The core architecture and user interface of Fortitude was inspired primarily by Ruff (Ruff Collaboration, 2026), a widely-used Python linter, and, where possible, Ruff's language-agnostic features directly repurposed thanks to its open and permissive licensing. This reuse of a tried-and-tested design enabled more rapid development than would otherwise be possible, and resulted in a user experience that can be readily understood by those already familiar with Ruff.

The generation of an abstract/concrete syntax tree from a Fortran file is no simple task, especially given the high degree of backwards compatibility in the Fortran standards. Therefore,

this was achieved with the pre-existing TreeSitter parsing framework, which provides a fast and robust solution with Rust bindings (Tree-sitter Collaboration, 2026). The Fortran extension, tree-sitter-fortran, has itself received numerous upgrades thanks to the experience gained working on Fortitude (Stadelman & Hill, 2026).

Research impact statement

Fortitude started as an in-house tool for refactoring legacy Fortran codes within plasma physics, and is now used for code quality control by teams all over the world, with users ranging from individual researchers to government institutions. The earliest known external use was for FTorch (Atkinson et al., 2025), a library for running neural network models in Fortran. The Met Office uses Fortitude for various projects, including LFRic (Met Office Collaboration, 2026b), CASIM (Met Office Collaboration, 2026a), and Socrates (Met Office Collaboration, 2026c). It is also used by the quantum chemistry and solid state physics package cp2k (Kühne et al., 2020) and the stellar astrophysics package MESA (Paxton et al., 2011).

The number of contributors to Fortitude has grown from the original two authors to over 20, and more community members have raised feature requests and bug reports. It receives over 4,000 downloads per month via PyPI, and a further unknown number of downloads of platform-specific binaries using a provided installer script. With over 200 GitHub stars, Fortitude appears to be the most popular open-source Fortran linting tool currently in use.

AI usage disclosure

Some of the code in Fortitude was created with the assistance of the generative AI tool GitHub Copilot, but none has been generated wholesale, and all code has been verified by at least two human developers prior to deployment. The software is thoroughly tested, and all tests have been designed manually. Other than spelling and grammar checking, this paper was written without AI assistance.

Acknowledgements

Fortitude's development has been funded by the PlasmaFAIR project, EPSRC Grant EP/V051822/1.

The authors are also grateful for the support of the Software Sustainability Institute.

We acknowledge contributions from Lawrence Dior at the University of York; Jack Atkinson and Tom Meltzer at the University of Cambridge; Andrew Browne and Austen Rainer at Queen's University Belfast; and John Collins at SimCon Ltd.

References

Atkinson, J., Elafrou, A., Kasoar, E., Wallwork, J. G., Meltzer, T., Clifford, S., Orchard, D., & Edsall, C. (2025). FTorch: A library for coupling PyTorch models to Fortran. *Journal of Open Source Software*, *10*(107), 7602. https://doi.org/10.21105/joss.07602

Backus, J. W., Beeber, R. J., Best, S., Goldberg, R., Haibt, L. M., Herrick, H. L., Nelson, R. A., Sayre, D., Sheridan, P. B., Stern, H., & others. (1957). The FORTRAN automatic coding system. *Papers Presented at the February 26-28, 1957, Western Joint Computer Conference: Techniques for Reliability*, 188–198.

CERFACS Collaboration. (2022). Flint. In *GitLab repository*. CERFACS. https://gitlab.com/cerfacs/flint

Fortran-Lang Collaboration. (2020). Fprettify: Auto-formatter for modern Fortran source code. In *GitHub repository*. Fortran-Lang. https://github.com/fortran-lang/fprettify

Fortran-Lang Collaboration. (2025). Fortls: Fortran language server. In *GitHub repository*. Fortran-Lang. https://github.com/fortran-lang/fortls

Fortran-Lang Collaboration. (2026). FPM: Fortran package manager. In *GitHub repository*. Fortran-Lang. https://github.com/fortran-lang/fpm

GS2 Collaboration. (2024). GS2: A gyrokinetic plasma simulation code. In *Bitbucket repository*. GS2. https://doi.org/10.5281/zenodo.2551066

Kedward, L. J., Aradi, B., Čertík, O., Curcic, M., Ehlert, S., Engel, P., Goswami, R., Hirsch, M., Lozada-Blanco, A., Magnin, V., Markus, A., Pagone, E., Pribec, I., Richardson, B., Snyder, H., Urban, J., & Vandenplas, J. (2022). The state of Fortran. *Computing in Science & Engineering*, *24*(2), 63–72. https://doi.org/10.1109/MCSE.2022.3159862

Kühne, T. D., Iannuzzi, M., Del Ben, M., Rybkin, V. V., Seewald, P., Stein, F., Laino, T., Khaliullin, R. Z., Schütt, O., Schiffmann, F., & others. (2020). CP2K: An electronic structure and molecular dynamics software package - quickstep: Efficient and accurate electronic structure calculations. *The Journal of Chemical Physics*, *152*(19). https://doi.org/10.1063/5.0007045

Met Office Collaboration. (2023). Stylist: Style checking for Fortran, PSyclone DSL, etc. In *GitHub repository*. Met Office. https://github.com/MetOffice/stylist

Met Office Collaboration. (2026a). CASIM: The Cloud and AeroSol Interacting Microphysics (CASIM) component of the Met Office-NERC cloud (MONC) model. In *GitHub repository*. Met Office. https://github.com/MetOffice/casim

Met Office Collaboration. (2026b). LFRic. In *GitHub repository*. Met Office. https://github.com/MetOffice/lfric_core

Met Office Collaboration. (2026c). Socrates: Suite of community radiative transfer codes based on Edwards and Slingo. In *GitHub repository*. Met Office. https://github.com/MetOffice/socrates

Orchard, D., & Rice, A. (2013). Upgrading Fortran source code using automatic refactoring. *Proceedings of the 2013 ACM Workshop on Workshop on Refactoring Tools*, 29–32. https://doi.org/10.1145/2541348.2541356

Paxton, B., Bildsten, L., Dotter, A., Herwig, F., Lesaffre, P., & Timmes, F. (2011). Modules for Experiments in Stellar Astrophysics (MESA). *192*, 3. https://doi.org/10.1088/0067-0049/192/1/3

Ruff Collaboration. (2026). Ruff: An extremely fast Python linter and code formatter, written in Rust. In *GitHub repository*. Astral. https://github.com/astral-sh/ruff

Stadelman, M., & Hill, P. (2026). Tree-sitter-fortran: Fortran grammar for tree-sitter. In *GitHub repository*. https://github.com/stadelmanma/tree-sitter-fortran

Tree-sitter Collaboration. (2026). Tree-sitter: An incremental parsing system for programming tools. In *GitHub repository*. Tree-sitter. https://github.com/tree-sitter/tree-sitter

Turner, A. (2022). Software usage data on ARCHER2. In *Blog post*. EPCC. https://www.archer2.ac.uk/news/2022/02/07/software-usage-data.html