The Journal of Open Source Software

Fiats: Functional inference and training for surrogates

Damian Rouson [1], **Dan Bonachea** [1], **Brad Richardson** [1], **Jordan A. Welsman** [1], **Jeremiah Bailey** [1], **Ethan D Gutmann** [2], **David Torres** [3], **Katherine Rasmussen** [1], **Baboucarr Dibba** [1], **Yunhao Zhang** [1], **Kareem Weaver** [1], **Zhe Bai** [1], **and Tan Nguyen** [1]

1 Lawrence Berkeley National Laboratory, United States **2** NSF National Center for Atmospheric Research, United States **3** Northern New Mexico College, United States

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Evan Spotte-Smith

Reviewers:

- @jwallwork23
- @niccolozanotti

Submitted: 02 July 2025
Published: unpublished

Summary

Fiats provides a platform for research on the training and deployment of neural-network surrogate models for computational science. Fiats also supports exploring, advancing, and combining functional, object-oriented, and parallel programming patterns in Fortran 2023 (Fortran Standards Committee JTC1/SC22/WG5, Nov 2023). As such, the Fiats name has dual expansions: "Functional Inference And Training for Surrogates" or "Fortran Inference And Training for Science." Fiats inference and training procedures are `pure` and therefore satisfy a language constraint imposed on procedure invocations inside Fortran's parallel loop construct: `do concurrent`. Furthermore, the Fiats training procedures are built around a `do concurrent` parallel reduction. Several compilers can automatically parallelize `do concurrent` on Central Processing Units (CPUs) or Graphics Processing Units (GPUs). Fiats thus aims to achieve performance portability through standard language mechanisms.

In addition to an `example` subdirectory with illustrative codes, the Fiats `demo/app` subdirectory contains three demonstration applications:

1. One trains a cloud-microphysics surrogate for the Berkeley Lab fork of the Intermediate Complexity Atmospheric Research (ICAR) model.
2. Another calculates input- and output-tensor statistics for ICAR's physics-based microphysics models.
3. A third performs batch inference using an aerosols surrogate for the Energy Exascale Earth Systems Model (E3SM).

Ongoing research explores how Fiats can exploit multi-image execution, a set of Fortran features for Single-Program, Multiple-Data (SPMD) parallel programming with a Partitioned Global Address Space (PGAS) (Numrich, 2018), where the PGAS features center around "coarray" distributed data structures.

To explore how new language features and novel uses of longstanding features can power deep learning, Fiats contributors work to advance Fortran by

- Participating in the Fortran standardization process and
- Contributing to compiler development through
  - Writing unit tests (Rasmussen et al., 2022),
  - Studying performance (Rouson, Bai, Bonachea, Ergawy, et al., 2025),
  - Isolating and reporting compiler bugs and fixing front-end bugs,
  - Publishing and updating the Parallel Runtime Interface for Fortran (PRIF) (Bonachea et al., 2024a, 2024b), and
  - Developing the first PRIF implementation: Caffeine (Bonachea et al., 2025; Rouson & Bonachea, 2022).

Fiats thus facilitates studying deep learning for science and studying programming paradigms and patterns for deep learning in Fortran 2023.

Statement of need

Fortran 2008 introduced two forms of parallelism: `do concurrent` for loop-level parallelism and multi-image execution for SPMD/PGAS parallelism in shared or distributed memory. Fortran 2018 and 2023 expanded and refined these features by adding, for example,

1. `Do concurrent` iteration locality specifiers, including reductions and
2. Collective subroutines, image teams, events (semaphores), atomic subroutines, and more,

which creates a need for libraries and frameworks that support users who adopt these features. For example, one requirement impacting library design stems from the aforementioned language constraint allowing only side-effect-free (`pure`) procedure invocations inside `do concurrent`.

All intrinsic functions defined in the Fortran 2023 standard are `simple`, an attribute that implies `pure` plus additional constraints. Libraries that export `pure` procedures thus behave like extensions of the language. To wit, the Fortran 2023 standard states: "It is expected that most library procedures will conform to the constraints required of pure procedures, and so can be declared pure and referenced in `do concurrent` constructs… and within user-defined `pure` procedures."

Conversely, multi-image execution in a library places a requirement on the client code. The Fortran standard defines steps for synchronized image launch, synchronized normal termination, and single-image initiation of global error termination. Multi-image execution in a library thus requires support for multi-image execution in the main program.

A surrogate model's utility hinges upon inference calculations executing faster than the physics-based model the surrogate replaces. This commonly restricts the surrogate neural network to a few thousand tunable parameters. For networks of modest size, useful insights can sometimes be gleaned from visually inspecting the network parameters. Fiats therefore stores networks in human-readable JavaScript Object Notation (JSON) format. The Fiats companion package Nexport exports Fiats JSON files PyTorch.

State of the field

At least six open-source software packages provide deep learning services to Fortran. Three provide Fortran application programming interfaces (APIs) that wrap C++ libraries:

- Fortran-TF-Lib is a Fortran API for TensorFlow,
- FTorch is a Fortran API for libtorch, the PyTorch back-end, and
- TorchFort is also a Fortran API for libtorch.

As of this writing, recursive searches in the root directories of the these three projects find no `pure` procedures. Procedures are `pure` if declared as such or if declared `simple` or if declared `elemental` without the `impure` attribute. Because any procedure invoked within a `pure` procedure must also be `pure`, the absence of `pure` procedures precludes the use of these APIs anywhere in the call stack inside a `do concurrent` construct. Also, as APIs backed by C++ libraries, none use Fortran's multi-image execution features.

Three packages supporting deep learning in Fortran are themselves written in Fortran:

- Athena (Taylor, 2024)
- Fiats (Rouson, Bai, Bonachea, Ergawy, et al., 2025)
- neural-fortran (Curcic, 2019)

The Journal of Open Source Software

Searching the Athena, Fiats, and neural-fortran `src` subdirectories finds that over half of the procedures in each are `pure`, including 75% of Fiats procedures. Included in these tallies are procedures explicitly marked as `pure` along with `simple` procedures and `elemental` procedures without the `impure` attribute. Athena, Fiats, and neural-fortran each employ `do concurrent` extensively. Only Fiats, however, leverages the locality specifiers introduced in Fortran 2018 and expanded in Fortran 2023 to include parallel reductions.

Of the APIs and libraries discussed here, only neural-fortran and Fiats use multi-image features: neural-fortran in its core library and Fiats in a demonstration application. Both use multi-image features minimally, leaving considerable room for researching parallelization strategies.

Each of the Fortran deep learning APIs and libraries discussed in this paper is actively developed except Fortran-TF-Lib. Fortran-TF-Lib's most recent commit was in 2023 and no releases have been posted. The other mentioned projects have most-recent commits no older than two months as of June 2025.

Recent research and scholarly publications

Fiats supports research in training surrogate models and parallelizing batch inference calculations for atmospheric sciences. This research has generated two peer-reviewed paper submissions: one accepted to appear in workshop proceedings (Rouson, Bai, Bonachea, Ergawy, et al., 2025) and one in open review (Rouson, Bai, Bonachea, Dibba, et al., 2025). Four programs in the Fiats repository played significant roles in these two papers:

1. `example/concurrent-inferences.f90`,
2. `example/learn-saturated-mixing-ratio.f90`,
3. `app/demo/infer-aerosols.f90`, and
4. `app/demo/train-cloud-microphysics.f90`.

Rouson, Bai, Bonachea, Ergawy, et al. (2025) used program 1 to study automatically parallelizing batch inferences via `do concurrent`. Rouson, Bai, Bonachea, Dibba, et al. (2025) used programs 2–4 to study neural-network training for cloud microphysics and inference for atmospheric aerosols. The derived types in the Unified Modeling Language (UML) class diagram in Figure 1 enabled these studies.

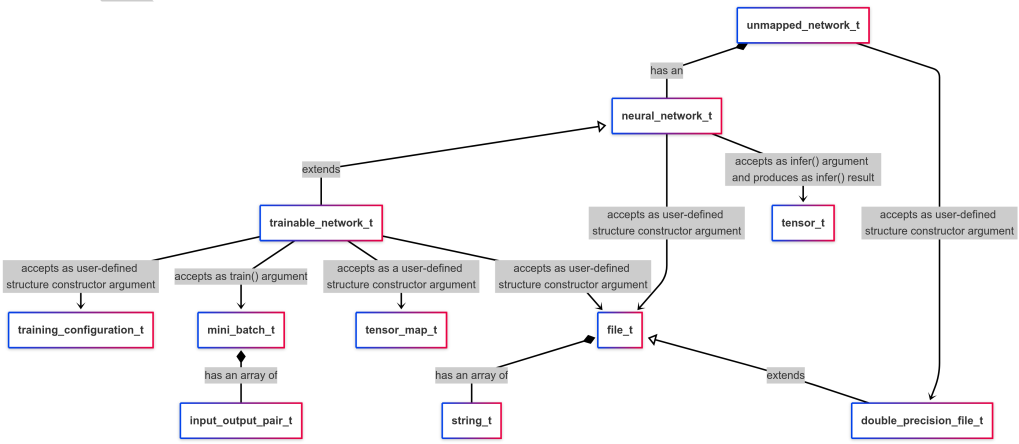

Figure 1: Class diagram: derived types (named in bordered white boxes), type relationships (connecting lines), type extension (open triangles), composition (solid diamonds), or directional relationship (arrows). Read relationships as sentences wherein the type named at the base of an arrow is the subject followed by an annotation (in an unbordered gray box) followed by the type named at the arrow's head as the object. Type extension reads with the type adjacent to the open triangle as the subject. Composition reads with the type adjacent to the closed diamond as the subject.

The Journal of Open Source Software

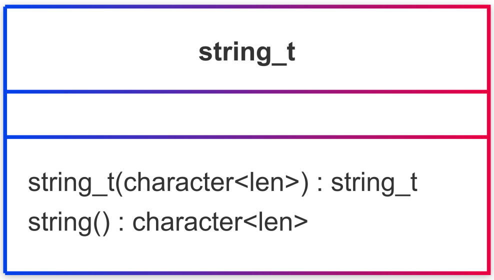

Figure 2: String class diagram

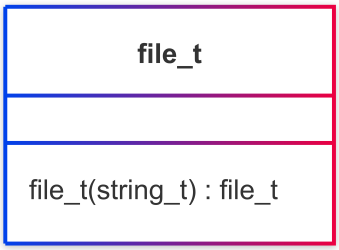

Figure 3: File class diagram

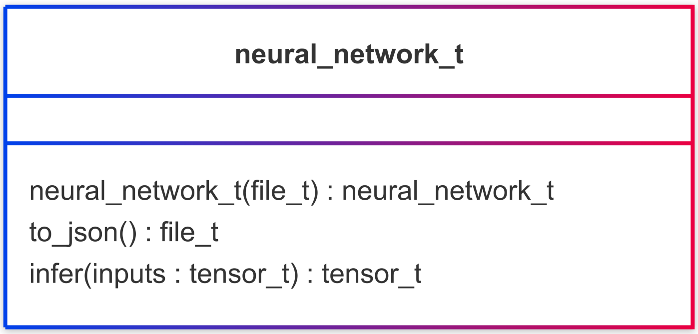

Figure 4: Neural network class diagram

Figure 1 includes two of the Julienne correctness-checking framework's derived types, `string_t` and `file_t`. These are included because other parts of the figure reference these types. The rightmost four types in Figure 1 exist primarily to support inference. The leftmost six support training. Because inference is considerably simpler, it makes sense to describe the right side of the diagram before the left side.

The `concurrent-inferences` example program performs batch inference using the `string_t`, `file_t`, and `neural_network_t` types. Figure 2 through Figure 4 show class diagrams with more details on these types. Each detailed diagram displays a top panel listing the type name, an empty middle panel with private components omitted, and a bottom panel listing public procedure bindings.

The bottom panel also lists what the Fortran 2023 standard describes as user-defined structure constructors, which are generic interfaces through which to invoke functions that define a result of the named type (Fortran Standards Committee JTC1/SC22/WG5, Nov 2023). We henceforth refer to these as "constructors." From the bottom of the class hierarchy in Figure 1, the `concurrent-inferences` program does the following:

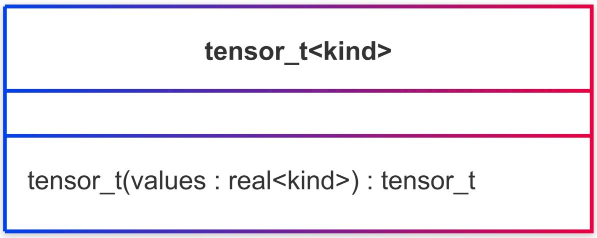

Figure 5: Tensor class diagram

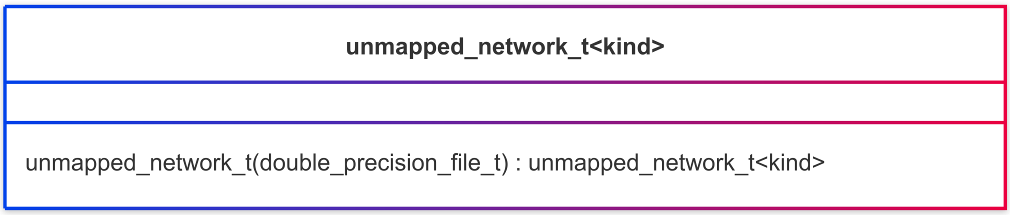

Figure 6: Unmapped network class diagram

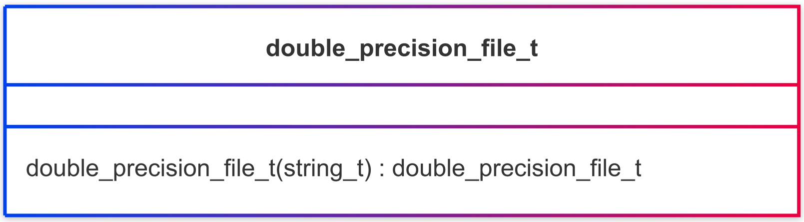

Figure 7: Double precision file class diagram

1. Gets a `character` file name from the command line,
2. Passes the name to a `string_t` constructor,
3. Passes the resulting `string_t` object to a `file_t` constructor, and
4. Passes the resulting `file_t` object to a `neural_network_t` constructor.

The program then repeatedly invokes the `infer` type-bound procedure on a three-dimensional (3D) array of `tensor_t` objects (see Figure 5) using OpenMP directives or `do concurrent` or an array statement. The array statement takes advantage of `infer` being `elemental`. Line 101 of `example/concurrent-inferences.f90` at git tag `joss-line-references` demonstrates neural-network construction from a file. Line 109 demonstrates using the network for inference.

The `infer-aerosols` program performs inferences by invoking `double precision` versions of the `infer` generic binding on an object of type `unmapped_network_t` (see Figure 6), a parameterized derived type (PDT) that has a `kind` type parameter. To match the expected behavior of the aerosol model, which was trained in PyTorch, the `unmapped_network_t` implementation ensures the use of raw network input and output tensors without the normalizations and remappings that are performed by default for a `neural_network_t` object. The `double_precision_file_t` (see Figure 7) type controls the interpretation of the JSON network file: JSON does not distinguish between categories of numerical values such as `real`, `double precision`, or even `integer`, so something external to the file must determine the interpretation of the numbers in a JSON file.

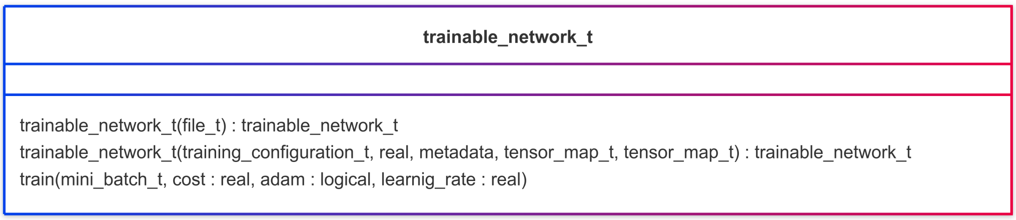

Figure 8: Trainable network class diagram

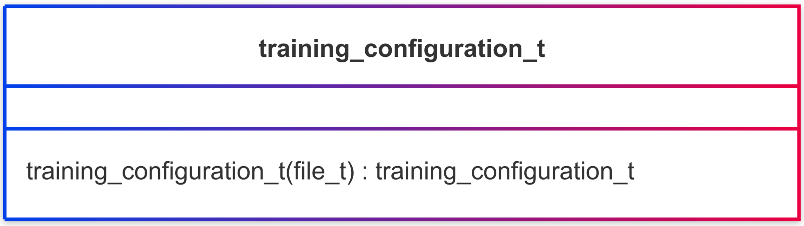

Figure 9: Training configuration class diagram

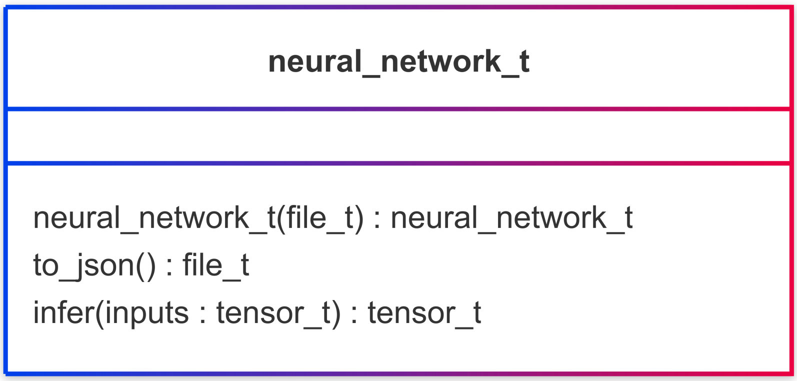

Figure 10: Mini-batch class diagram

The `learn-saturated-mixing-ratio` and `train-cloud-microphysics` programs focus on using a `trainable_network_t` object (see Figure 8) for training. The former trains neural network surrogates for a thermodynamic function from ICAR: the saturated mixing ratio, a scalar function of temperature and pressure. The latter trains surrogates for the complete cloud microphysics models in ICAR – models implemented in thousands of lines of code. Whereas diagrammed relationships of `neural_network_t` reflect direct dependencies of only two types (`file_t` and `tensor_t`), even describing the basic behaviors of `trainable_network_t` requires showing dependencies on five types:

- A `training_configuration_t` object (see Figure 9), which holds hyperparameters such as the learning rate and choice of optimization algorithms,
- A `file_t` object from which the `training_configuration` is read inside the `trainable_network_t` constructor,
- A `mini_batch_t` object (see Figure 10) that stores an array of `input_output_pair` objects (see Figure 11) from the training data set,
- Two `tensor_map_t` objects (see Figure 12) storing the linear functions that map inputs to the training data range and map outputs from the training data range back to the

application range, and

- A parent `neural_network_t` object storing the network architecture, including weights, biases, layer widths, etc.

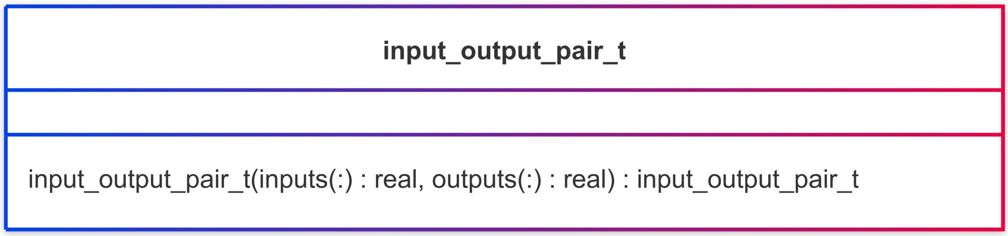

Figure 11: Input/Output tensor pair class diagram

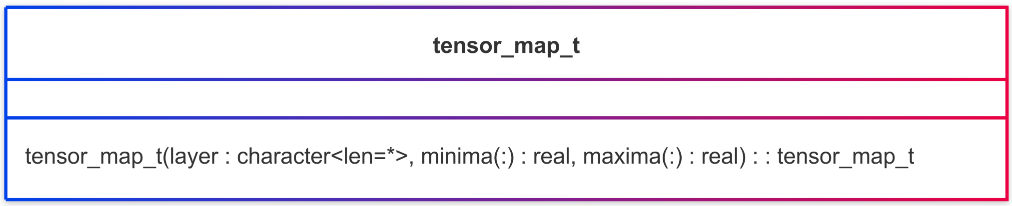

Figure 12: Tensor map class diagram

The `trainable_network_t` type stores a `workspace_t` (not shown) as a scratch-pad for training purposes. The workspace is not needed for inference. During each training step, a `trainable_network_t` object passes its `workspace_t` to a `learn` procedure binding (not shown) on its parent `neural_network_t`. Lines 388–396 of `demo/app/train-cloud-microphysics.f90` at git tag `joss-line-references` demonstrate:

1. A loop over epochs,
2. The shuffling of the `input_output_pair_t` objects at the beginning of each epoch,
3. The grouping of `input_output_pair_t` objects into `mini_batch_t` objects, and
4. The invocation of the `train` procedure for each mini-batch,

where steps 2 and 3 express deep learning's stochastic gradient descent algorithm.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of Nuclear Physics, Scientific Discovery through Advanced Computing (SciDAC) Next-Generation Scientific Software Technologies (NGSST) programs under Contract No. DE-AC02-05CH11231. This material is also based on work supported by Laboratory Directed Research and Development (LDRD) funding from Lawrence Berkeley National Laboratory, provided by the Director, Office of Science, of the U.S. DOE under Contract No. DE-AC02-05CH11231. This manuscript has been authored by an author at Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. The U.S. Government retains, and the publisher, by accepting the article for publication, acknowledges, that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

References

Bonachea, D., Rasmussen, K., Richardson, B., & Rouson, D. (2024a). Parallel Runtime Interface for Fortran (PRIF): A Multi-Image Solution for LLVM Flang. *Tenth Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC2024)*. https://doi.org/10.25344/S4N017

Bonachea, D., Rasmussen, K., Richardson, B., & Rouson, D. (2024b). *Parallel runtime interface for Fortran (PRIF) specification (rev. 0.5)*. Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA (United States). https://doi.org/10.25344/S4CG6G

Bonachea, D., Rasmussen, K., Richardson, B., & Rouson, D. (2025). Caffeine: A parallel runtime library for supporting modern Fortran compilers. *Journal of Open Source Software*, *10*(107), 7895. https://doi.org/10.21105/joss.07895

Curcic, M. (2019). A parallel Fortran framework for neural networks and deep learning. *Acm Sigplan Fortran Forum*, *38*, 4–21. https://doi.org/10.1145/3323057.3323059

Fortran Standards Committee JTC1/SC22/WG5. (Nov 2023). *Information technology – programming languages – ISO/IEC 1539-1:2023*. International Organization for Standardization (ISO).

Numrich, R. W. (2018). *Parallel Programming with Co-arrays*. CRC Press. https://doi.org/doi:10.1201/9780429437182

Rasmussen, K., Rouson, D., George, N., Bonachea, D., Kadhem, H., & Friesen, B. (2022). *Agile acceleration of LLVM flang support for Fortran 2018 parallel programming*. Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA (United States). https://doi.org/10.25344/S4CP4S

Rouson, D., Bai, Z., Bonachea, D., Dibba, B., Gutmann, E., Rasmussen, K., Torres, D., Welsman, J., & Zhang, Y. (2025). Cloud microphysics training and aerosol inference with the Fiats deep learning library. *2025 Improving Scientific Software Conference*, (in review).

Rouson, D., Bai, Z., Bonachea, D., Ergawy, K., Gutmann, E., Klemm, M., Rasmussen, K., Richardson, B., Shende, S., Torres, D., & Zhang, Y. (2025). Automatically parallelizing batch inference on deep neural networks using Fiats and Fortran 2023 "do concurrent." *5th International Workshop on Computational Aspects of Deep Learning (CADL)*. https://doi.org/10.25344/S4VG6T

Rouson, D., & Bonachea, D. (2022). Caffeine: CoArray Fortran Framework of Efficient Interfaces to Network Environments. *2022 IEEE/ACM Eighth Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, 34–42. https://doi.org/10.25344/S4459B

Taylor, N. T. (2024). ATHENA: A Fortran package for neural networks. *Journal of Open Source Software*, *9*(99), 6492. https://doi.org/10.21105/joss.06492