

Fiats: Functional inference and training for surrogates

Damian Rouson¹, Dan Bonachea¹, Brad Richardson¹, Jordan A. Welsman¹, Jeremiah Bailey¹, Ethan D Gutmann², David Torres³, Katherine Rasmussen¹, Baboucarr Dibba¹, Yunhao Zhang¹, Kareem Weaver¹, Zhe Bai¹, and Tan Nguyen¹

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

DOI: [10.21105/joss.08785](https://doi.org/10.21105/joss.08785)

Software

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

Editor: [Evan Spotte-Smith](#)

Reviewers:

- [@jwallwork23](#)
- [@niccolozanotti](#)

Submitted: 02 July 2025

Published: 05 December 2025

License

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

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, 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 Lawrence Berkeley National Laboratory [fork](#) of the Intermediate Complexity Atmospheric Research (ICAR) model.
2. Another calculates input- and output-tensor statistics for ICAR’s physics-based micro-physics 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.

Statement of need

Developers of computational science software lack widespread support for adopting Fortran’s built-in parallel programming features. Those features take two forms: `do concurrent` for loop-level parallelism and multi-image execution for SPMD/PGAS parallelism in shared or distributed memory. Fiats addresses this problem by providing inference and training procedures that are compatible with both forms of parallel language features. Fiats thus facilitates studying deep learning for science and programming paradigms and patterns for deep learning in Fortran 2023. Fiats also provides a vehicle for contributors to ensure that Fortran supports the algorithms that are central to the emerging field of deep learning and to drive improvements in the Fortran language to best support this domain. The next section covers related work.

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](#) ([Atkinson et al., 2025](#)) 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](#))

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. These annotations make it more tractable for compilers to correctly parallelize code on processors or offload code to accelerators such as GPUs.

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.

Recent research and scholarly publications

Fiats supports research in training surrogate models and parallelizing batch inference calculations for atmospheric sciences. This research recently generated two peer-reviewed papers described in this section. Four programs in the Fiats repository played significant roles in these 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](#) 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 five 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. From the bottom of the class hierarchy in [Figure 1](#), the concurrent-inferences program does the following:

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 each element of a three-dimensional (3D) array of `tensor_t` objects using OpenMP directives or `do concurrent` or an array statement. The array statement takes advantage of infer being elemental. The following line example/concurrent-inferences.f90 demonstrates neural-network construction from a file:

```
neural_network = neural_network_t(file_t(network_file_name))
```

In the same example, the following line demonstrates using the network for inference:

```
outputs(i,k,j) = neural_network%infer(inputs(i,k,j))
```

The infer-aerosols program performs inferences by invoking double precision versions of the infer generic binding on an object of type `unmapped_network_t`, 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` 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.

The learn-saturated-mixing-ratio and train-cloud-microphysics programs focus on using a `trainable_network_t` object 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, which holds hyperparameters such as the learning rate and choice of optimization algorithms,
- A `file_t` object representing a JSON input file from which the training configuration can alternatively be read inside the `trainable_network_t` constructor,
- A `mini_batch_t` object that stores an array of `input_output_pair` objects from the training data set,
- Two `tensor_map_t` objects 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.

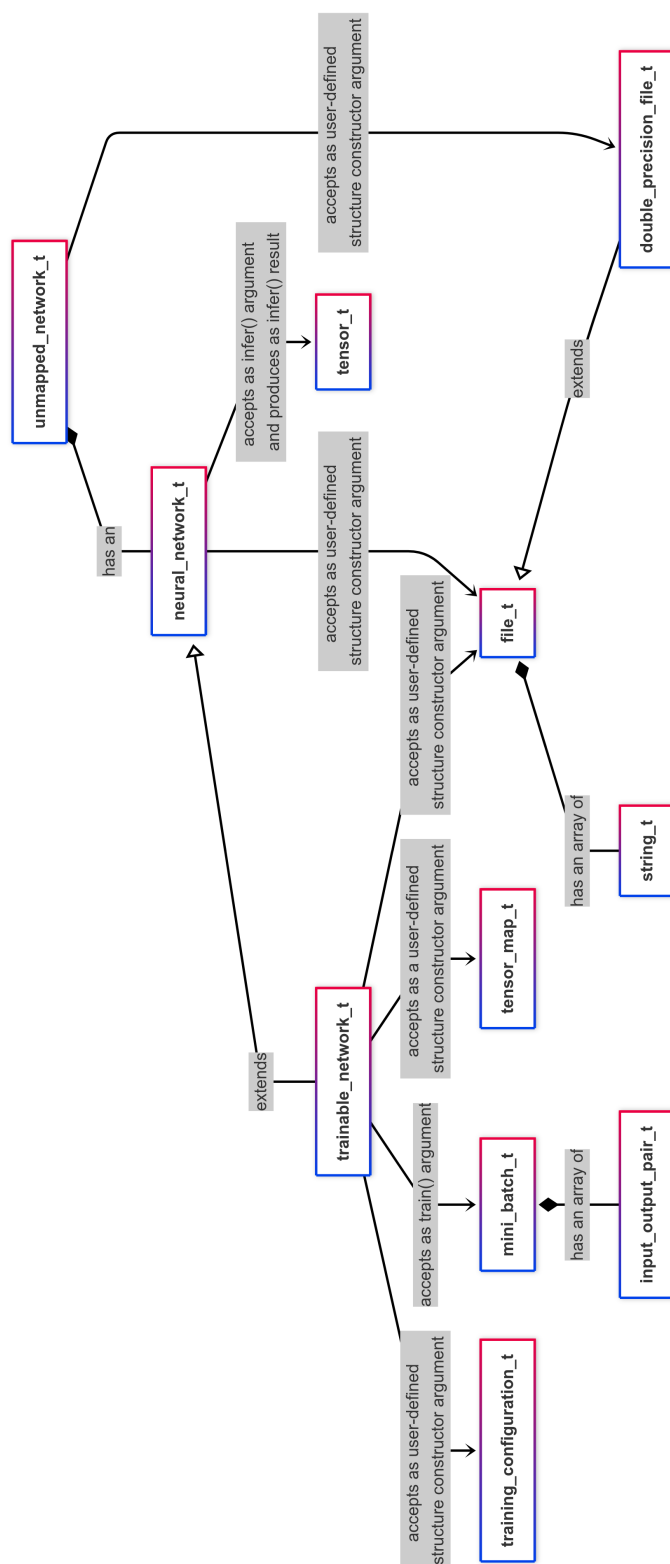


Figure 1: Class diagram: derived types (named in bordered white boxes), type relationships (connecting lines), type extension (open triangles), composition (solid diamonds), or directional relationship (arrows). Read relationships as sentences wherein the type named at the base of an arrow is the subject 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 `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

- 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>
- 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. (2023). *Information technology – programming languages – Fortran, ISO/IEC 1539-1:2023*. International Organization for Standardization (ISO).
- Numrich, R. W. (2018). *Parallel Programming with Co-arrays*. CRC Press. <https://doi.org/10.1201/9780429437182>
- 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. *Improving Scientific Software Conference (ISS)*. <https://doi.org/10.25344/S4QS3J>
- 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>
- 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>