











FTorch: a library for coupling PyTorch models to Fortran

Jack Atkinson ¹✉, Athena Elafrou², Elliott Kasoar ^{1,3}, Joseph G. Wallwork ¹, Thomas Meltzer ¹, Simon Clifford ¹, Dominic Orchard ^{1,4}, and Chris Edsall ¹

¹ Institute of Computing for Climate Science, University of Cambridge, UK ² NVIDIA, UK ³ Scientific Computing Department, Science and Technology Facilities Council, UK ⁴ University of Kent, UK ✉ Corresponding author

DOI: [10.21105/joss.07602](https://doi.org/10.21105/joss.07602)

Software

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

Editor: [Matthew Feickert](#)  

Reviewers:

- [@parikshitbajpai](#)
- [@timothyas](#)
- [@anand-me](#)

Submitted: 06 December 2024

Published: 05 March 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

In the last decade, machine learning (ML) and deep learning (DL) techniques have revolutionised many fields within science, industry, and beyond. Researchers across domains are increasingly seeking to combine ML with numerical modelling to advance research. This typically brings about the challenge of *programming language interoperation*. PyTorch ([Paszke et al., 2019](#)) is a popular framework for designing and training ML/DL models whilst Fortran remains a language of choice for many high-performance computing (HPC) scientific models. The FTorch library provides an easy-to-use, performant, cross-platform method for coupling the two, allowing users to call PyTorch models from Fortran.

FTorch is open-source, open-development, and well-documented with minimal dependencies. A central tenet of its design, in contrast to other approaches, is that FTorch removes dependence on the Python runtime (and virtual environments). By building on the LibTorch backend (written in C++ and accessible via an API), it allows users to run ML models on both CPU and GPU architectures without needing to port code to device-specific languages.

Statement of need

The explosion of ML/DL has brought several promising opportunities to deploy these techniques in scientific research. There are notable applications in the physical sciences ([Carleo et al., 2019](#)), climate science ([Kashinath et al., 2021](#)), and materials science ([Bishara et al., 2023](#)). Common applications include the emulation of computationally intensive processes and the development of data-driven components. Such deployments of ML can achieve improved computational and/or predictive performance, compared to traditional numerical techniques. A common example from the geosciences is ML parameterisation of subgrid processes—a major source of uncertainty in many models (e.g., Bony et al. (2015), Rasp et al. (2018)).

Fortran is widely used for scientific codes due to its performance, stability, array-oriented design, and native support for shared and distributed memory, amongst other features ([Kedward et al., 2022](#)). Many ML frameworks, on the other hand, are accessed using Python. The commonly-used PyTorch framework allows users to design and deploy ML models with many advanced features.

Ideally, users would develop and validate ML models in the PyTorch environment before deploying them into a scientific model. This deployment should require minimal additional code, and guarantee identical results as obtained with the PyTorch interface—something not

guaranteed if re-implementing by hand in Fortran. Ideally one would call out, from Fortran, to an ML model saved from PyTorch, with the results returned directly to the scientific code.

FTorch bridges this gap, reducing the burden on researchers seeking to incorporate ML into their numerical models. It provides an intuitive and user-friendly interface from Fortran to ML models developed using PyTorch. It removes the need for detailed knowledge about language interoperation and the need to develop bespoke coupling code, instead providing a Fortran interface designed to be familiar to both PyTorch and Fortran users.

Having no dependence on the Python runtime, FTorch avoids the incurred overhead of object representations in Python's runtime and is appealing for HPC systems where the management of Python environments can be challenging.

Software description

FTorch is a Fortran wrapper to the LibTorch C++ framework using the `iso_c_binding` module, intrinsic to Fortran since the 2003 standard. This enables shared memory use (where possible) to maximise efficiency by reducing data-transfer during coupling¹ and avoids any use of Python at runtime. PyTorch types are represented through derived types in FTorch, with Tensors supported across a range of data types and ranks using the `fypp` preprocessor (Aradi, 2024).

We utilise the existing support in LibTorch for GPU acceleration without additional device-specific code. `torch_tensors` are targeted to a device through a `device_type` enum, currently supporting CPU, CUDA, XPU, and MPS. Multiple GPUs may be targeted through the optional `device_index` argument.

Typically, users train a model in PyTorch and save it as TorchScript, a strongly-typed subset of Python. Once saved, the Torchscript model can be loaded from Fortran using FTorch and run via the LibTorch backend. The library comes with a utility script (`pt2ts.py`) to assist with the process of saving models as well as a comprehensive set of examples guiding users through complete Python to Fortran workflows. A focus on user experience underpins the development and is a key aspect behind the adoption of FTorch by various scientific communities.

Full details, including user guide, API documentation, slides and videos, and links to projects is available at <https://cambridge-iccs.github.io/FTorch>.

Comparison to other approaches

- **Replicating a net in Fortran**

That is, taking a model developed in PyTorch and reimplementing it using only Fortran, loading saved weights from file. This is likely to require considerable development effort, re-writing code that already exists and missing opportunities to use the diverse and highly-optimised features of Torch. Re-implementation can be a source of error, requiring additional testing to ensure correctness.

If the overall goal is to incorporate ML into Fortran, rather than using PyTorch specifically, then another approach is to leverage a Fortran-based ML framework such as `neural-fortran` (Curcic, 2019). Whilst it does not allow interaction with PyTorch, `neural-fortran` provides many neural network components for building nets directly in Fortran. However, the set of features is not as rich as PyTorch and GPU offloading is not currently supported.

The `Fiats` ('Functional Inference And Training for Surrogates') library (Rouson & Rasmussen, 2024) is another approach for developing, training, and deploying ML models directly in Fortran, with experimental GPU support at present.

¹i.e. the same data in memory is used by both LibTorch and Fortran without creating a copy.

- **Forpy** ([Rabel, 2020](#))
Forpy is a Fortran module that provides access to Python data structures (including numpy arrays) for interoperability. Whilst this provides wider access to general Python features it has a challenging interface with more boilerplate. It also requires access to the Python runtime from Fortran.
- **TorchFort** ([NVIDIA, 2024](#))
Since we began FTorch, NVIDIA has released TorchFort. This has a similar approach to FTorch, avoiding Python to link against the LibTorch backend. It has a focus on enabling GPU deployment on NVIDIA hardware.
- **fortran-tf-lib** ([Cambridge-ICCS, 2023](#))
Whilst FTorch allows coupling of PyTorch models to Fortran, some use TensorFlow ([Abadi et al., 2015](#)) to develop and train models. These can be coupled to Fortran in a similar manner to FTorch by using `fortran-tf-lib`. Whilst it provides an alternative solution, the library is less rich in features and software sustainability than FTorch.
- **SmartSim** ([Partee et al., 2022](#))
SmartSim is a workflow library developed by HPE and built upon Redis API. It provides a framework for launching ML and HPC workloads, transferring data between the two via a database. This is a versatile approach that can work with a variety of languages and ML frameworks. However, it has a significant learning curve, incurs data-transfer overheads, and requires managing tasks from Python.

Examples of Use

FTorch is actively used in scientific research:

- in the [DataWave project](#) exploring gravity wave drag in atmospheric models to:
 - couple an emulator into the MiMA model ([DataWave, 2023](#)) demonstrating variability of models trained offline when coupled back to a host ([Mansfield & Sheshadri, 2024](#)).
 - couple emulators and new data-driven parameterisations to the Community Atmosphere Model (CAM).
- to couple a U-Net based model of multi-scale convection into [ICON](#) ([DKRZ, 2025](#)) and demonstrate via Shapley values that non-causal learnt relations are more stable when running online ([Heuer et al., 2024](#)).
- As part of [CESM](#) (the Community Earth System Model) ([NCAR, 2025](#)) working to provide a general approach for researchers to couple ML models to the various components of the model suite.

Future development

Recent work in scientific domains suggests that online training is likely important for long-term stability of hybrid models ([Brenowitz et al., 2020](#)). We therefore plan to extend FTorch to expose PyTorch's autograd functionality to support this.

We welcome feature requests and are open to discussion and collaboration.

Acknowledgments

This project is supported by Schmidt Sciences, LLC. We also thank the Institute of Computing for Climate Science for their support.

References

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., ... Zheng, X. (2015). *TensorFlow, Large-scale machine learning on heterogeneous systems*. <https://doi.org/10.5281/zenodo.4724125>
- Aradi, B. (2024). *Fypp*. <https://fypp.readthedocs.io>
- Bishara, D., Xie, Y., Liu, W. K., & Li, S. (2023). A state-of-the-art review on machine learning-based multiscale modeling, simulation, homogenization and design of materials. *Archives of Computational Methods in Engineering*, 30(1), 191–222. <https://doi.org/10.1007/s11831-022-09795-8>
- Bony, S., Stevens, B., Frierson, D. M., Jakob, C., Kageyama, M., Pincus, R., Shepherd, T. G., Sherwood, S. C., Siebesma, A. P., Sobel, A. H., & others. (2015). Clouds, circulation and climate sensitivity. *Nature Geoscience*, 8(4), 261–268. <https://doi.org/10.1038/ngeo2398>
- Brenowitz, N. D., Henn, B., McGibbon, J., Clark, S. K., Kwa, A., Perkins, W. A., Watt-Meyer, O., & Bretherton, C. S. (2020). Machine learning climate model dynamics: Offline versus online performance. *arXiv Preprint arXiv:2011.03081*. <https://doi.org/10.48550/arXiv.2011.03081>
- Cambridge-ICCS. (2023). *Fortran-tf-lib*. <https://github.com/Cambridge-ICCS/fortran-tf-lib>
- Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., & Zdeborová, L. (2019). Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4), 045002. <https://doi.org/10.1103/RevModPhys.91.045002>
- 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>
- DataWave. (2023). *MiMA machine learning*. <https://github.com/DataWaveProject/MiMA-machine-learning>
- DKRZ. (2025). *ICON (icosahedral nonhydrostatic) model*. <https://www.icon-model.org/>
- Heuer, H., Schwabe, M., Gentine, P., Giorgetta, M. A., & Eyring, V. (2024). Interpretable multiscale machine learning-based parameterizations of convection for ICON. *Journal of Advances in Modeling Earth Systems*, 16(8), e2024MS004398. <https://doi.org/10.1029/2024MS004398>
- Kashinath, K., Mustafa, M., Albert, A., Wu, J., Jiang, C., Esmailzadeh, S., Azzadenesheli, K., Wang, R., Chattopadhyay, A., Singh, A., & others. (2021). Physics-informed machine learning: Case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A*, 379(2194), 20200093. <https://doi.org/10.1098/rsta.2020.0093>
- Kedward, L. J., Aradi, B., Čertík, O., Curcic, M., Ehlert, S., Engel, P., Goswami, R., Hirsch, M., Lozada-Blanco, A., Magnin, V., & others. (2022). The state of Fortran. *Computing in Science & Engineering*, 24(2), 63–72. <https://doi.org/10.1109/MCSE.2022.3159862>
- Mansfield, L. A., & Sheshadri, A. (2024). Uncertainty quantification of a machine learning subgrid-scale parameterization for atmospheric gravity waves. *Journal of Advances in Modeling Earth Systems*, 16(7), e2024MS004292. <https://doi.org/10.1029/2024MS004292>
- NCAR. (2025). *CESM, the community earth system model*. <https://www.cesm.ucar.edu/>
- NVIDIA. (2024). *TorchFort*. <https://nvidia.github.io/TorchFort/>
- Partee, S., Ellis, M., Rigazzi, A., Shao, A. E., Bachman, S., Marques, G., & Robbins, B. (2022). Using machine learning at scale in numerical simulations with SmartSim: An application to ocean climate modeling. *Journal of Computational Science*, 62, 101707. <https://doi.org/10.1016/j.jocs.2022.101707>

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., & others. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32.

Rabel, E. (2020). *Forpy*. <https://github.com/ylikx/forpy>

Rasp, S., Pritchard, M. S., & Gentine, P. (2018). Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39), 9684–9689. <https://doi.org/10.1073/pnas.1810286115>

Rouson, D., & Rasmussen, K. (2024). *Fiats: Functional inference and training for surrogates*. <https://github.com/BerkeleyLab/fiats>