The Journal of Open Source Software

Φ_{ML}: Intuitive Scientific Computing with Dimension Types for Jax, PyTorch, TensorFlow & NumPy

Philipp Holl [1] **and Nils Thuerey** [1]

1 School of Computation, Information and Technology, Technical University of Munich, Germany

DOI: 10.21105/joss.06171

Software

- Review
- Repository
- Archive

Editor: Marcel Stimberg

Reviewers:

- @wandeln
- @chaoming0625
- @gauravbokil8

Submitted: 11 August 2023
Published: 29 February 2024

Summary

Φ_{ML} is a math and neural network library designed for science applications. It enables users to quickly evaluate many network architectures on their data sets, perform (sparse) linear and non-linear optimization, and write differentiable simulations that scale to *n* dimensions. Φ_{ML} is compatible with Jax, PyTorch, TensorFlow and NumPy, and user code can be executed on all of these backends. The project is hosted at https://github.com/tum-pbs/PhiML under the MIT license.

Statement of need

Machine learning (ML) has become an essential tool for scientific research. In recent years, ML has been used to make significant advances in a wide range of scientific fields, including chemistry (Butler et al., 2018), materials science (Wei et al., 2019), weather and climate prediction (Bochenek & Ustrnul, 2022; Rolnick et al., 2022), computational fluid dynamics (Brunton et al., 2020), drug discovery (Jumper et al., 2021; Vamathevan et al., 2019), astrophysics (De La Calleja & Fuentes, 2004; Ntampaka et al., 2015; Petroff et al., 2020), geology (Rodriguez-Galiano et al., 2015), and many more. The use of ML for scientific applications is still in its early stages, but it has the potential to revolutionize the way that science is done. ML can help researchers to make new discoveries and insights that were previously impossible.

The availability of domain knowledge sets science applications apart from other ML fields like computer vision or language modelling. Domain knowledge often allows for explicit modelling of known dynamics by simulating them with handwritten algorithms, which has been shown to improve results when training ML models (Raissi et al., 2019; Um et al., 2020). Implementing differentiable simulations into ML frameworks requires different functions and concepts than classical ML tasks. The major differences are:

- Data typically represent objects or signals that exist in space and time. Data dimensions are interpretable, e.g. vector components, time series, *n*-dimensional lattices.
- Information transfer is usually local, resulting in sparsity in the dependency matrix between objects (particles, elements or cells).
- A high numerical accuracy is desirable for some operations, often requiring 64-bit and 32-bit floating point calculations.

However, current machine learning frameworks have been designed for the core ML tasks which reflects in their priorities and design choices. This can result in overly verbose code when implementing scientific applications and may require implementing custom operators, since many common functions like sparse-sparse matrix multiplication, periodic padding or sparse linear solvers are not available in all libraries.

Φ_{ML} is a scientific computing library based on Python 3 (Van Rossum & Drake, 2009) targeting scientific applications that use machine learning methods. Its main goals are:

- **Reusability.** Code based on Φ_{ML} should be able to run in many settings without modification. It should be agnostic towards the dimensionality of simulated systems and the employed discretization. All code should be trivially vectorizable.
- **Compatibility.** Users should be free to choose whatever ML or third-party library they desire without modifying their simulation code. Φ_{ML} should support Linux, Windows and Mac.
- **Usability.** Φ_{ML} should be easy to learn and use, matching existing APIs where possible. It should encourage users to write concise and expressive code.
- **Maintainability.** All high-level source code of Φ_{ML} should be easy to understand. Continuous testing should be used to ensure that future updates do not break existing code.
- **Performance.** Φ_{ML} should be able to make use of hardware accelerators, such as GPUs and TPUs, where possible. During development, we prioritize rapid code iterations over execution speed but the completed code should run as fast as if written directly against the chosen ML library.

In the following, we explain the architecture and major features that help Φ_{Flow} reach these goals. Φ_{ML} consists of a high-level NumPy-like API geared towards writing easy-to-read and scalable simulation code, as well as a neural network API designed to allow users to quickly iterate over many network architectures and hyperparameter settings. Similar to eagerpy (Rauber et al., 2020), Φ_{ML} integrates with Jax (Bradbury et al., 2018), PyTorch (Paszke et al., 2019), TensorFlow (Abadi et al., 2016) and NumPy (Harris et al., 2020) and provides a custom `Tensor` class. However, Φ_{ML} adds additional functionality.

- **Dimension names.** Tensor dimensions are always referenced by their user-defined name, not their index. We support the syntax `tensor.dim` for operations like indexing or unstacking to make using dimension names as simple as possible.
- **Automatic reshaping.** Φ_{ML} automatically transposes tensors and inserts singleton dimensions to match arguments. Consequently, user code is agnostic to the dimension order by default.
- **Element names.** Slices or *items* along dimensions can be named as well, e.g. allowing users to specify that a dimension lists the values `(x,y,z)` or `(r,g,b)`. These names can be used in slicing, gathering and scattering operations.
- **Dimension types.** Tensor dimensions are grouped into five different types: *batch*, *spatial*, *instance*, *channel*, and *dual*. This allows tensor-related functions to automatically select dimensions to operate on, without requiring the user to specify individual dimensions.
- **Non-uniform tensors.** Stacking tensors with different dimension sizes yields non-uniform tensors. Φ_{ML} keeps track of the resulting shape, allowing users to operate on non-uniform tensors the same way as uniform ones.
- **Floating-point precision by context.** All tensor operations determine the desired floating point precision from the operation context, not the data types of its inputs. This is much simpler and more predictable than the systems used by other libraries.
- **Lazy stacking.** New memory is only allocated once stacked data is required as a block. Consequently, functions can unstack the components, operate on them individually, and restack them, without worrying about unnecessary memory allocations.
- **Sparse matrices from linear functions.** Φ_{ML} can transform linear functions into their corresponding sparse matrix representation. This makes solving linear systems of equations more performant and enables computation of preconditioners.
- **Compute device from Inputs.** Tensor operations execute on the device on which the tensors reside. This prevents unintentional copies and transfers, as users have to explicitly declare them.
- **Custom CUDA Operatorions.** Φ_{ML} provides custom CUDA kernels for specific operations that could bottleneck simulations, such as grid sampling for TensorFlow or linear solves.

Research Projects

Φ_{ML} has been in development since 2019 as part of the Φ_{Flow} project where it originated as a unified API for TensorFlow and NumPy, used to run differentiable fluid simulations. Φ_{Flow} includes geometry, physics, and visualization modules, all of which use the `math` API of Φ_{ML} to benefit from its reusability, compatibility, and performance.

It was first used to show that differentiable PDE simulations can be used to train neural networks that steer the dynamics towards desired outcomes (Holl et al., 2019). Differentiable PDEs, implemented against Φ_{ML}'s API, were later shown to benefit learning corrections for low-resolution or incomplete physics models (Um et al., 2020). These findings were summarized and formalized in Thuerey et al. (2022), along with many additional examples.

The library was also used in network optimization publications, such as showing that inverted simulations can be used to train networks (Holl et al., 2022) and that gradient inversion benefits learning the solutions to inverse problems (Schnell et al., 2021).

Simulations powered by Φ_{ML} have since been used in open data sets (Gupta & Brandstetter, 2022; Takamoto et al., 2022) and in publications from various research groups (Brandstetter et al., 2021, 2023; Li et al., 2023; Parekh et al., 1993; Ramos et al., 2022; Sengar et al., 2021; Wandel et al., 2020, 2021; P. Wang, 2023; R. Wang et al., 2022a, 2022b; Wu et al., 2022).

Acknowledgements

We would like to thank Robin Greif, Kartik Bali, Elias Djossou and Brener Ramos for their contributions, as well as everyone who contributed to the project on GitHub.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., & others. (2016). Tensorflow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265–283.

Bochenek, B., & Ustrnul, Z. (2022). Machine learning in weather prediction and climate analyses—applications and perspectives. *Atmosphere*, *13*(2), 180. https://doi.org/10.3390/atmos13020180

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.2.5). http://github.com/google/jax

Brandstetter, J., Berg, R. van den, Welling, M., & Gupta, J. K. (2023). *Clifford neural layers for PDE modeling*. arXiv. https://arxiv.org/abs/2209.04934

Brandstetter, J., Worrall, D. E., & Welling, M. (2021). Message passing neural PDE solvers. *International Conference on Learning Representations*.

Brunton, S. L., Noack, B. R., & Koumoutsakos, P. (2020). Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, *52*, 477–508. https://doi.org/10.1146/annurev-fluid-010719-060214

Butler, K. T., Davies, D. W., Cartwright, H., Isayev, O., & Walsh, A. (2018). Machine learning for molecular and materials science. *Nature*, *559*(7715), 547–555. https://doi.org/10.1038/s41586-018-0337-2

De La Calleja, J., & Fuentes, O. (2004). Machine learning and image analysis for morphological galaxy classification. *Monthly Notices of the Royal Astronomical Society*, *349*(1), 87–93. https://doi.org/10.1111/j.1365-2966.2004.07442.x

Gupta, J. K., & Brandstetter, J. (2022). *Towards multi-spatiotemporal-scale generalized PDE modeling*. arXiv. https://arxiv.org/abs/2209.15616

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Holl, P., Koltun, V., & Thuerey, N. (2022). Scale-invariant learning by physics inversion. *Advances in Neural Information Processing Systems*, *35*, 5390–5403.

Holl, P., Thuerey, N., & Koltun, V. (2019). Learning to control PDEs with differentiable physics. *International Conference on Learning Representations*.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., & others. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583–589. https://doi.org/10.1038/s41586-021-03819-2

Li, Z., Patil, S., Shu, D., & Farimani, A. B. (2023). Latent neural PDE solver for time-dependent systems. *NeurIPS 2023 AI for Science Workshop*.

Ntampaka, M., Trac, H., Sutherland, D. J., Battaglia, N., Póczos, B., & Schneider, J. (2015). A machine learning approach for dynamical mass measurements of galaxy clusters. *The Astrophysical Journal*, *803*(2), 50. https://doi.org/10.1088/0004-637X/803/2/50

Parekh, N., Zou, A., Jungling, I., Endlich, K., Sadowski, J., & Steinhausen, M. (1993). Sex differences in control of renal outer medullary circulation in rats: Role of prostaglandins. *American Journal of Physiology-Renal Physiology*, *264*(4), F629–F636. https://doi.org/10.1152/ajprenal.1993.264.4.F629

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*.

Petroff, M. A., Addison, G. E., Bennett, C. L., & Weiland, J. L. (2020). Full-sky cosmic microwave background foreground cleaning using machine learning. *The Astrophysical Journal*, *903*(2), 104. https://doi.org/10.3847/1538-4357/abb9a7

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

Ramos, B., Trost, F., & Thuerey, N. (2022). Control of two-way coupled fluid systems with differentiable solvers. *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*. https://doi.org/10.48550/arXiv.2206.00342

Rauber, J., Bethge, M., & Brendel, W. (2020). *EagerPy: Writing code that works natively with PyTorch, TensorFlow, JAX, and NumPy*. arXiv. https://arxiv.org/abs/2008.04175

Rodriguez-Galiano, V., Sanchez-Castillo, M., Chica-Olmo, M., & Chica-Rivas, M. (2015). Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines. *Ore Geology Reviews*, *71*, 804–818. https://doi.org/10.1016/j.oregeorev.2015.01.001

Rolnick, D., Donti, P. L., Kaack, L. H., Kochanski, K., Lacoste, A., Sankaran, K., Ross, A. S., Milojevic-Dupont, N., Jaques, N., Waldman-Brown, A., & others. (2022). Tackling climate change with machine learning. *ACM Computing Surveys (CSUR)*, *55*(2), 1–96.

Schnell, P., Holl, P., & Thuerey, N. (2021). Half-inverse gradients for physical deep learning. *International Conference on Learning Representations*.

Sengar, V., Seemakurthy, K., Gubbi, J., & P, B. (2021). Multi-task learning based approach for surgical video desmoking. *Proceedings of the Twelfth Indian Conference on Computer Vision, Graphics and Image Processing*, 1–9. https://doi.org/10.1145/3490035.3490283

Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., & Niepert, M. (2022). PDEBench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, *35*, 1596–1611.

Thuerey, N., Holl, P., Mueller, M., Schnell, P., Trost, F., & Um, K. (2022). *Physics-based deep learning*. arXiv. https://arxiv.org/abs/2109.05237

Um, K., Brand, R., Fei, Y. R., Holl, P., & Thuerey, N. (2020). Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. *Advances in Neural Information Processing Systems*, *33*, 6111–6122.

Vamathevan, J., Clark, D., Czodrowski, P., Dunham, I., Ferran, E., Lee, G., Li, B., Madabhushi, A., Shah, P., Spitzer, M., & others. (2019). Applications of machine learning in drug discovery and development. *Nature Reviews Drug Discovery*, *18*(6), 463–477. https://doi.org/10.1038/s41573-019-0024-5

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. CreateSpace. ISBN: 1441412697

Wandel, N., Weinmann, M., & Klein, R. (2020). Learning incompressible fluid dynamics from scratch-towards fast, differentiable fluid models that generalize. *International Conference on Learning Representations*.

Wandel, N., Weinmann, M., & Klein, R. (2021). Teaching the incompressible Navier–Stokes equations to fast neural surrogate models in three dimensions. *Physics of Fluids*, *33*(4). https://doi.org/10.1063/5.0047428

Wang, P. (2023). The applications of generative adversarial network in surgical videos. *Third International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI 2022)*, *12509*, 300–305. https://doi.org/10.1117/12.2656026

Wang, R., Walters, R., & Yu, R. (2022a). Approximately equivariant networks for imperfectly symmetric dynamics. *International Conference on Machine Learning*, 23078–23091.

Wang, R., Walters, R., & Yu, R. (2022b). Meta-learning dynamics forecasting using task inference. *Advances in Neural Information Processing Systems*, *35*, 21640–21653.

Wei, J., Chu, X., Sun, X.-Y., Xu, K., Deng, H.-X., Chen, J., Wei, Z., & Lei, M. (2019). Machine learning in materials science. *InfoMat*, *1*(3), 338–358. https://doi.org/10.1002/inf2.12028

Wu, T., Maruyama, T., & Leskovec, J. (2022). Learning to accelerate partial differential equations via latent global evolution. *Advances in Neural Information Processing Systems*, *35*, 2240–2253.