Journal of Machine Learning Research 27 (2026) 1-6 Submitted 2/26; Published 6/26

A Library for Learning Neural Operators

Jean Kossaifi[1*], **Nikola Kovachki**[1*], **Zongyi Li**[2*], **David Pitt**[2*], **Miguel Liu-Schiaffini**[2], **Robert J. George**[2], **Boris Bonev**[1], **Kamyar Azizzadenesheli**[1], **Julius Berner**[1,2], **Valentin Duruisseaux**[2] and **Anima Anandkumar**[2]
[1] *NVIDIA* [2] *Caltech*

Editor: Zeyi Wen

Abstract

We present NeuralOperator, an open-source Python library for operator learning. Neural operators generalize neural networks to maps between function spaces instead of finite-dimensional Euclidean spaces. They can be trained and inferenced on input and output functions given at various discretizations, satisfying a discretization convergence properties. Part of the official PyTorch Ecosystem, NeuralOperator provides all the tools for training and deploying neural operator models, as well as developing new ones, in a high-quality, tested, open-source package. It combines cutting-edge models and customizability with a gentle learning curve and simple user interface for newcomers and researchers.

Keywords: Neural operators, deep learning, python, PyTorch, open-source, OSS

1. Introduction

Most scientific problems involve mappings between functions, not finite-dimensional data: notably, partial differential equations (PDEs) are naturally described on function spaces. A practical use case would be, e.g. learning a solution operator mapping between initial conditions and solution functions. Traditional numerical methods operate on discretizations of functions based on meshes of the computational domains, with their accuracy heavily depending on the meshes' resolutions. In concrete applications, such as weather or climate simulations, the requirement of a fine mesh renders such methods computationally intensive, making it intractable to simulate solutions across large sets of parameters (e.g. initial conditions or coefficients) in a reasonable amount of time (Schneider et al., 2017). Deep neural networks have been considered to accelerate the solution of PDEs However, they can only learn mappings between finite-dimensional spaces, not function spaces. In other words, the solutions learned are tied to a fixed discretization, and they often perform poorly when interpolated to higher resolutions (Azizzadenesheli et al., 2024).

To address these limitations, *neural operators* were proposed (Li et al., 2021; Kovachki et al., 2023; Berner et al., 2025). Unlike standard neural networks, neural operators can directly learn mappings between functions, i.e., infinite-dimensional spaces (Kovachki et al., 2024; Berner et al., 2025). They are built from first principles to ensure a *discretization convergence* property: a neural operator, with a fixed set of parameters, can be applied

[*] These authors contributed equally to this work.

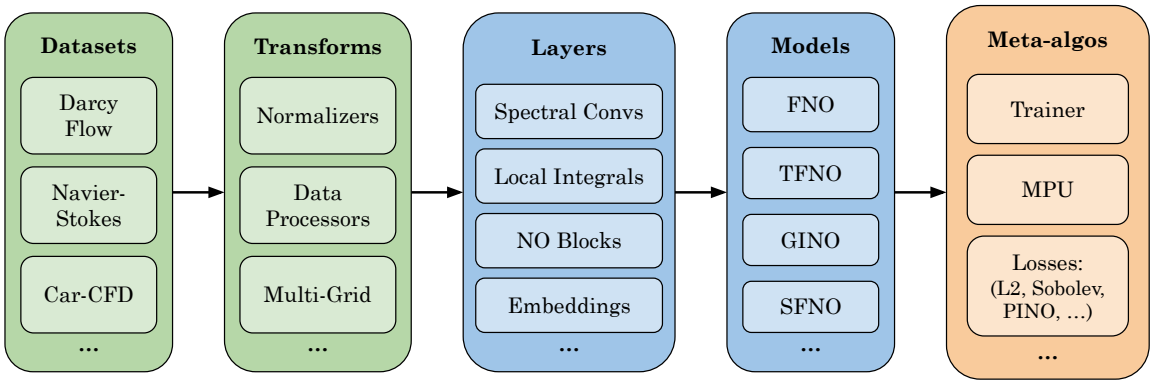

Figure 1: **Overview of functionalities.** The NeuralOperator library provides the full stack required to train and deploy neural operators, including **datasets** and **data loaders**, **core layers and building blocks**, **neural operator models**, **losses** and **training and efficiency methods**.

to input functions given at any discretization. Specifically, for an input function given at various discretizations, the outputs differ only by a discretization error that converges to zero as the discretization is refined.

Neural operators are ideally suited for solving scientific problems, such as PDEs, where the solution is naturally formulated as a map between infinite-dimensional function spaces (Kovachki et al., 2023). However, they present a learning curve to newcomers, and it is non-trivial to correctly implement neural operators that preserve their ability to operate on any input/output resolutions while maintaining their discretization convergence. This library offers state-of-the-art, batteries-included implementations of neural operator models and building blocks, as well as all the tools necessary for training and inference, allowing practitioners to seamlessly use it in their applications and combine it with existing PyTorch codebases.

2. The NeuralOperator Library

NeuralOperator is an official PyTorch Ecosystem[1] project, open-sourced under MIT license[2]. The NeuralOperator library builds on the following **guiding principles**:

- **Resolution-agnostic:** While neural operators come with theoretical and empirical properties on resolution convergence, cross-resolution generalization and physical consistency are not automatic. These properties still depend on the choice of data, well-posedness of the problem itself, data processing, model choice, loss design, etc. This library aims at making these easier by providing modules such as data loaders, architectures, and loss functions, that are applicable to functions at various discretizations. As an example, our implementation of the Fourier Neural Operator (FNO) (Li et al., 2021; Duruisseaux et al., 2025), just like the theoretical model, can take inputs on regular grids of any resolution. Where applicable, we also support input and outputs of different resolutions, e.g. super-resolution.

[1] https://landscape.pytorch.org/
[2] NeuralOperator is hosted at https://github.com/neuraloperator/neuraloperator.

- **Easy-to-use and beginner-friendly:** Out of the box, NeuralOperator provides all necessary functionality to apply neural operators to real-world scientific machine learning problems. It contains simple interfaces to prebuilt neural operator architectures from the literature, subcomponent layers, a `Trainer` module that automates the common training procedure, and additional submodules, including data modules and common loss functions for training neural operators. The library provides example training scripts and a detailed documentation, including an API reference, a user guide, and tutorials on layers and scientific datasets, from basics to advanced customization. A comprehensive practical guide to FNOs with NeuralOperator is also available (Duruisseaux et al., 2025).
- **Flexible for advanced users:** The library is designed to be highly modular. It enables a fast learning curve, rapid experimentation, and configurability, while providing a simple interface to neural operators for newcomers. This is achieved by providing various layers of modularity, from end-to-end models to more involved blocks such as spherical convolutions or tensor factorizations and algorithms such as multi-grid domain decomposition and incremental learning. We aim for the package to grow along with the field and continue providing state-of-the-art architectures, and welcome contributions from the community.
- **Reliable:** The library is designed with reliability in mind. All core functionalities are covered by unit tests, and new additions are validated with an automated CI/CD pipeline that includes tests from the function level up to end-to-end model testing. Moreover, it is documented both in-line in the source code and on an accompanying documentation website and API reference that are automatically built in CI/CD.

Neural Operator Building Blocks. The library offers a variety of prebuilt operator `torch.nn.Module` layers, that can be combined to build any existing or new neural operator model. These layers can be broken down into integral transforms (Li et al., 2020, 2021; Kovachki et al., 2023; Bonev et al., 2023), pointwise operators, positional embeddings, multi-layer blocks, and extra functionality (padding, normalization, interpolation, etc.).

Neural Operator Architectures. NeuralOperator provides state-of-the-art neural operator architectures and training recipes. The core building block of neural operators is the integral transform, a learnable map between two functions supplied at any two meshes. We provide implementations for several neural operator architectures. Using a general learnable kernel integral, *Graph Neural Operators* (GNOs) (Li et al., 2020) allow learning on arbitrary geometries. When inputs are defined over a regular grid, the kernel integral can be efficiently realized through a Fast Fourier Transform, resulting in the spectral convolution layer used in FNOs (Li et al., 2021). We also implement *Tensor FNOs* (TFNOs) (Kossaifi et al., 2023), which use low-rank tensor decompositions. When working on a sphere, we can leverage the Spherical Fourier transform, implemented via spherical harmonic transforms *Spherical FNOs* (SFNOs) (Bonev et al., 2023). The efficiency of FNOs and SFNOs on structured grids can be combined with GNOs or optimal transport, yielding the *Geometry-informed Neural Operators* (GINOs) (Li et al., 2023; Lin et al., 2025) and Optimal Transport Neural Operators (OTNOs) (Li et al., 2025). Locally supported kernels to learn differential and integral operators (LocalNOs) (Liu-Schiaffini et al., 2024) can also supplement spectral convolutions. Recurrent Neural Operators (RNOs) (Liu-Schiaffini et al., 2023) extend FNOs

to systems with long-term temporal dependencies by introducing recurrence and memory directly on function spaces. *Physics-Informed Neural Operators* (PINOs) (Li et al., 2024; Ganeshram et al., 2025) can also be trained and finetuned leveraging derivative computation routines and other functionalities available in NeuralOperator.

Datasets. We also provide convenient interfaces to common benchmark datasets for training operators on PDE data. The data is hosted on a public Zenodo archive (at `https://zenodo.org/communities/neuraloperator`) and available for automatic download through the library's modules. This includes input-output pairs governed by Darcy's law, a 1d viscous Burger's equation (Li et al., 2024), 2d Navier-Stokes equations (Kossaifi et al., 2023), as well as 3d Navier-Stokes equations over the surface of car models (Umetani and Bickel, 2018).

Trainers and meta-algorithms. NeuralOperator provides a suite of functionalities to simplify the training and deployment of neural operators. We provide a flexible `DataProcessor` to pipeline all normalization and transformations needed to convert inputs and outputs into the forms expected by the model and for computing losses. These also ensure that discretization convergence is respected in operations such as domain padding. We also provide a minimal `Trainer` that automates the standard logic of a machine-learning training loop, applying the data processor, optimizing model parameters, and tracking validation metrics throughout training. The `Trainer` is modular and highly configurable; users can provide their own models, data, and objectives, and the interface provides users with the opportunity to add custom training logic for domain-specific applications, including meta-algorithms such as incremental learning (George et al., 2024). Newcomers can directly use it on their own applications, while advanced users can directly import the neural architectures and core components and modules they need into their own workflow.

Efficiency The library also packages a variety of tools for memory-efficient training of neural operator models. To compress the learnable parameters of an FNO model by performing tensor decomposition on spectral weights (Kossaifi et al., 2019, 2023), an interface for tensorization is directly exposed in our implementation of the FNO model and spectral convolution layers. Additionally, the `Trainer` includes a native option for quantization via mixed-precision training (Tu et al., 2024). Last, training of neural operators can be done incrementally (George et al., 2024), and distributed using the built-in multi-grid domain decomposition (Kossaifi et al., 2023). These features further expand the library's capabilities and accessibility by enabling the learning of larger-scale operators on GPU.

3. Conclusion

NeuralOperator provides state-of-the-art neural operator architectures and associated functionality in a modular, robust, and well-documented package. Built on top of PyTorch, its simple interfaces and modular components offer a gentle learning curve for new users while remaining highly extensible for conducting real-world experiments with neural operator models. It aims to democratize neural operators for scientific applications, grow alongside the field, and provide the latest architectures and layers as the state-of-the-art progresses.

Acknowledgement

David Pitt is supported by the Schmidt Scholars in Software Engineering program. Anima Anandkumar is supported in part by the Bren endowed chair, ONR (MURI grant N00014-23-1-2654), and the AI2050 senior fellow program at Schmidt Sciences.

References

Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics*, pages 1–9, 2024.

Julius Berner, Miguel Liu-Schiaffini, Jean Kossaifi, Valentin Duruisseaux, Boris Bonev, Kamyar Azizzadenesheli, and Anima Anandkumar. Principled approaches for extending neural architectures to function spaces for operator learning. 2025. URL https://arxiv.org/abs/2506.10973.

Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere. In *International conference on machine learning*, pages 2806–2823. PMLR, 2023.

Valentin Duruisseaux, Jean Kossaifi, and Anima Anandkumar. Fourier neural operators explained: A practical perspective, 2025. URL https://arxiv.org/abs/2512.01421.

Adarsh Ganeshram, Haydn Maust, Valentin Duruisseaux, Zongyi Li, Yixuan Wang, Daniel Leibovici, Oscar Bruno, Thomas Hou, and Anima Anandkumar. Fc-pino: High precision physics-informed neural operators via fourier continuation, 2025.

Robert Joseph George, Jiawei Zhao, Jean Kossaifi, Zongyi Li, and Anima Anandkumar. Incremental spatial and spectral learning of neural operators for solving large-scale PDEs. *Transactions on Machine Learning Research*, 2024.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research*, 20(26):1–6, 2019.

Jean Kossaifi, Nikola Kovachki, Kamyar Azizzadenesheli, and Anima Anandkumar. Multi-grid tensorized fourier neural operator for high-resolution PDEs, 2023.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24 (89):1–97, 2023.

Nikola B Kovachki, Samuel Lanthaler, and Andrew M Stuart. Operator learning: Algorithms and analysis. *arXiv preprint arXiv:2402.15715*, 2024.

Xinyi Li, Zongyi Li, Nikola Kovachki, and Anima Anandkumar. Geometric operator learning with optimal transport, 2025. URL https://arxiv.org/abs/2507.20065.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

Zongyi Li, Nikola Borislavov Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Prakash Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Anima Anandkumar. Geometry-informed neural operator for large-scale 3D PDEs, 2023. URL https://arxiv.org/abs/2309.00583.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/JMS Journal of Data Science*, 1(3):1–27, 2024.

Ryan Y. Lin, Julius Berner, Valentin Duruisseaux, David Pitt, Daniel Leibovici, Jean Kossaifi, Kamyar Azizzadenesheli, and Anima Anandkumar. Enabling Automatic Differentiation with Mollified Graph Neural Operators, April 2025.

Miguel Liu-Schiaffini, Clare E Singer, Nikola Kovachki, Tapio Schneider, Kamyar Azizzadenesheli, and Anima Anandkumar. Tipping point forecasting in non-stationary dynamics on function spaces. *arXiv preprint arXiv:2308.08794*, 2023.

Miguel Liu-Schiaffini, Julius Berner, Boris Bonev, Thorsten Kurth, Kamyar Azizzadenesheli, and Anima Anandkumar. Neural operators with localized integral and differential kernels. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024. URL https://openreview.net/forum?id=fTOeB5L4PP.

Tapio Schneider, João Teixeira, Christopher S Bretherton, Florent Brient, Kyle G Pressel, Christoph Schär, and A Pier Siebesma. Climate goals and computing the future of clouds. *Nature Climate Change*, 7(1):3–5, 2017.

Renbo Tu, Colin White, Jean Kossaifi, Boris Bonev, Gennady Pekhimenko, Kamyar Azizzadenesheli, and Anima Anandkumar. Guaranteed approximation bounds for mixed-precision neural operators. In *International Conference on Learning Representations*, 2024.

Nobuyuki Umetani and Bernd Bickel. Learning three-dimensional flow for interactive aerodynamic design. *ACM Trans. Graph.*, 37(4), July 2018. doi: 10.1145/3197517.3201325.