A Library for Learning Neural Operators

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

[1]*NVIDIA*   [2]*Caltech*

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. Built on top of PyTorch, 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.

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, for instance, 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 (such as 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 by mapping from parameters directly to the solution on a given discretization (Guo et al., 2016; Bhatnagar et al., 2019; Wang et al., 2020; Gupta and Brandstetter, 2023). 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. In particular, there is no guarantee that neural networks generalize to other discretizations, and they often perform poorly when interpolated to higher resolutions (Azizzadenesheli et al., 2024).

To address these limitations, a new class of machine learning models, known as *neural operators*, was proposed (Li et al., 2020, 2021; Lu et al., 2021; Hao et al., 2023; Raonic et al., 2024). While standard neural networks learn mappings between fixed-size discretizations, i.e., finite-dimensional spaces, neural operators can directly learn mappings between functions, i.e., infinite-dimensional spaces (Bhattacharya et al., 2021; Lu et al., 2021; Kovachki et al., 2024; Azizzadenesheli et al., 2024). They are built from first principles to ensure a *discretization*

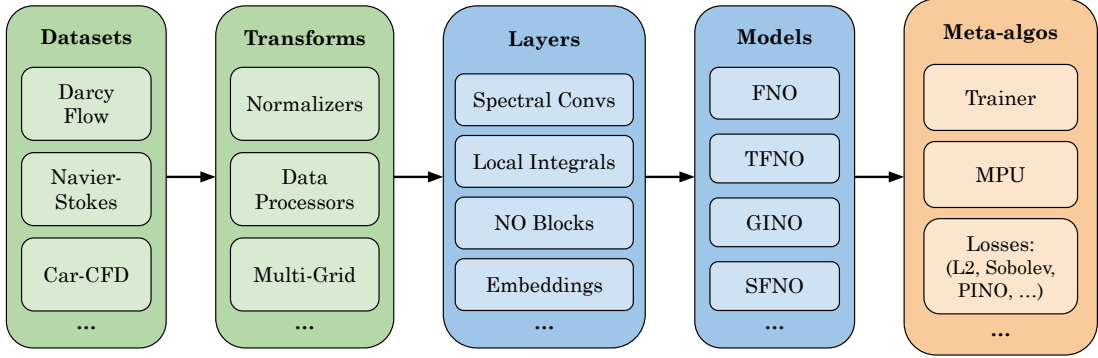*These authors contributed equally to this work.

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**.

convergence property: a neural operator, with a fixed set of parameters, can be applied 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.

As such, 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 (Li et al., 2021; 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 property of 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 to train and inference them on practical problems. Practitioners can seamlessly use it in their applications and combine it with existing PyTorch codebases.

2 The NeuralOperator Library

NEURALOPERATOR is open-sourced under MIT license[1]. In the following, we provide details on its design principles and functionalities; see Figure 1 for an overview.

The NEURALOPERATOR library builds on the following **guiding principles**:

- **Resolution-agnostic:** As the crucial difference to existing frameworks, modules in NEURALOPERATOR, such as data loaders, architectures, and loss functions, should be applicable to functions at various discretization. As an example, our implementation of

---

[1]Neural Operator is hosted at https://github.com/neuraloperator/neuraloperator.

the Fourier Neural Operator (FNO) (Li et al., 2021), just like the theoretical model, can take inputs on regular grids of any resolution. Where applicable, we also aim to support super-resolution, in which model input and output resolutions differ.

- **Easy-to-use and beginner-friendly:** Out of the box, NEURALOPERATOR provides all necessary functionality to apply neural operator models to scientific machine learning problems in the real world. NEURALOPERATOR contains simple interfaces to prebuilt neural operator architectures from the literature, subcomponent layers, a `Trainer` module that automates the common procedure of training a neural operator, and additional submodules, including data modules and common loss functions for training neural operators. NEURALOPERATOR is built on top of PyTorch (Paszke et al., 2019), so we follow the `torch.nn.Module` convention for implementing models and their forward passes based on submodule layers.

- **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 in the field. 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.

- **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 layers that can be combined to build any existing or new neural operator model. These layers can be broken down by category into integral transforms (Li et al., 2021, 2020; 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 end-to-end training examples to apply these architectures to various problems. The core building block of neural operators is the integral transform, a learnable map between two functions supplied at any two meshes. We provide several implementations for different 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 (FFT), resulting in the spectral convolution layer used in *Fourier Neural Operator* (FNOs) (Li et al., 2021). We also implement *Tensorized Fourier Neural Operators* (TFNOs) (Kossaifi et al., 2023), which use tensor decompositions for the parameters of the spectral convolution and architectural improvements of FNOs. When working on a sphere, we can leverage the Spherical Fourier transform, which is implemented via the spherical harmonic transform as proposed in *Spherical Fourier Neural*

Operators (SFNOs) (Bonev et al., 2023). The advantages of GNOs and FNOs can be combined using *Geometry-informed Neural Operators* (GINOs).

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 one-dimensional viscous Burger's equation (Li et al., 2024), two-dimensional Navier-Stokes equations (Kossaifi et al., 2023), as well as three-dimensional Navier-Stokes equations over the surface of car models (Umetani and Bickel, 2018).

Trainers and meta-algorithms. NEURALOPERATOR also provides a suite of functionalities to simplify the process of training and deploying neural operator models. We provide a flexible `DataProcessor` module to pipeline all normalization and additional transformations into the form expected for a model's forward pass and transform raw outputs into the form expected for computing losses. These also ensure that discretization convergence is respected in operations such as domain padding. We also provide a minimal `Trainer` module that automates the standard logic of a machine-learning training loop, applying the data processor, optimizing model parameters, and tracking validation metrics throughout the course of 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., 2023, 2019), 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.

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

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.

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64:525–545, 2019.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric PDEs. *The SMAI journal of computational mathematics*, 7, 2021.

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.

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.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490, 2016.

Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized PDE modeling. *Transactions on Machine Learning Research*, 2023.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. GNOT: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.

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.

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, 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.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Bogdan Raonic, Roberto Molinaro, Tim De Ryck, Tobias Rohner, Francesca Bartolucci, Rima Alaifari, Siddhartha Mishra, and Emmanuel de Bézenac. Convolutional neural operators for robust and accurate learning of PDEs. *Advances in Neural Information Processing Systems*, 36, 2024.

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.

Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1457–1466, 2020.