Chapter 3

A mathematical guide to operator learning

Nicolas Boullé[a,*] and Alex Townsend[b]
[a]*Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, United Kingdom,* [b]*Department of Mathematics, Cornell University, Ithaca, NY, United States*
[*]*Corresponding author: e-mail address: nb690@cam.ac.uk*

Contents

Abstract
Operator learning aims to discover properties of an underlying dynamical system or partial differential equation (PDE) from data. Here, we present a step-by-step guide to operator learning. We explain the types of problems and PDEs amenable to operator learning, discuss various neural network architectures, and explain how to employ numerical PDE solvers effectively. We also give advice on how to create and manage training data and conduct optimization. We offer intuition behind the various neural network architectures employed in operator learning by motivating them from the point-of-view of numerical linear algebra.

Handbook of Numerical Analysis, Volume 25, ISSN 1570-8659, https://doi.org/10.1016/bs.hna.2024.05.003

Keywords
Scientific machine learning, Deep learning, Operator learning, Partial differential equations

MSC Codes
47-02, 47A58, 47F99, 65-02, 65F55, 65J10

1 Introduction

The recent successes of deep learning (LeCun et al., 2015) in computer vision (Krizhevsky et al., 2012), language model (Brown et al., 2020), and biology (Jumper et al., 2021) have caused a surge of interest in applying these techniques to scientific problems. The field of scientific machine learning (SciML) (Karniadakis et al., 2021), which combines the approximation power of machine learning (ML) methodologies and observational data with traditional modeling techniques based on partial differential equations (PDEs), sets out to use ML tools for accelerating scientific discovery.

SciML techniques can roughly be categorized into three main areas: (1) PDE solvers, (2) PDE discovery, and (3) operator learning (see Fig. 1). First, PDE solvers, such as physics-informed neural networks (PINNs) (Raissi et al., 2019; Lu et al., 2021b; Cuomo et al., 2022; Wang et al., 2023), the deep Galerkin method (Sirignano and Spiliopoulos, 2018), and the deep Ritz method (E and Yu, 2018), consist of approximating the solution a known PDE by a neural network by minimizing the solution's residual. At the same time, PDE discovery aims to identify the coefficients of a PDE from data, such as the SINDy approach (Brunton et al., 2016; Champion et al., 2019), which relies on sparsity-promoting algorithms to determine coefficients of dynamical systems. There are also symbolic regression techniques, such as AI Feynman introduced by Udrescu and Tegmark (2020); Udrescu et al. (2020) and genetic algorithms (Schmidt and Lipson, 2009; Searson et al., 2010), that discover physics equations from experimental data.

Here, we focus on the third main area of SciML, called operator learning (Lu et al., 2021a; Kovachki et al., 2023). Operator learning aims to discover or approximate an unknown operator $\mathcal{A}$, which often takes the form of the solution operator associated with a differential equation. In mathematical terms, the problem can be defined as follows. Given pairs of data (f, u), where $f \in \mathcal{U}$ and $u \in \mathcal{V}$ are from function spaces on a d-dimensional spatial domain $\Omega \subset \mathbb{R}^d$, and a (potentially nonlinear) operator $\mathcal{A} : \mathcal{U} \to \mathcal{V}$ such that $\mathcal{A}(f) = u$, the objective is to find an approximation of $\mathcal{A}$, denoted as $\hat{\mathcal{A}}$, such that for any new data $f' \in \mathcal{U}$, we have $\hat{\mathcal{A}}(f') \approx \mathcal{A}(f')$. In other words, the approximation should be accurate for both the training and unseen data, thus demonstrating good generalization.

This problem is typically approached by representing $\hat{\mathcal{A}}$ as a neural operator, which is a generalization of neural networks as the inputs and outputs

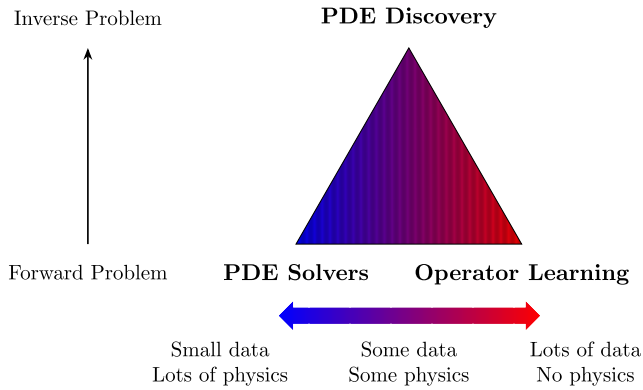

FIGURE 1 Illustrating the role of operator learning in SciML. Operator learning aims to discover or approximate an unknown operator $\mathcal{A}$, which often corresponds to the solution operator of an unknown PDE. In contrast, PDE discovery aims to discover coefficients of the PDE itself, while PDE solvers aim to solve a known PDE using ML techniques.

are functions, not vectors. After discretizing the functions at sensor points $x_1, \ldots, x_m \in \Omega$, one then parametrizes the neural operator with a set of parameters $\theta \in \mathbb{R}^N$, which could represent the weights and biases of the underlying neural network. Then, one typically formulates an optimization problem to find the best parameters:

$$\min_{\theta \in \mathbb{R}^N} \sum_{(f,u) \in \text{data}} L(\hat{\mathcal{A}}(f;\theta), u), \tag{1}$$

where L is a loss function that measures the discrepancy between $\hat{A}(f;\theta)$ and u, and the sum is over all available training data pairs (f, u). The challenges of operator learning often arise from selecting an appropriate neural operator architecture for $\hat{A}$, the computational complexities of solving the optimization problem, and the ability to generalize to new data.

A typical application of operator learning arises when learning the solution operator associated with a PDE, which maps a forcing function f to a solution u. One can informally think of it as the (right) inverse of a differential operator. One of the simplest examples is the solution operator associated with Poisson's equation with zero Dirichlet conditions:

$$-\nabla^2 u = f, \quad x \in \Omega \subset \mathbb{R}^d, \quad u|_{\partial\Omega} = 0, \tag{2}$$

where $\partial\Omega$ means the boundary of Ω. In this case, the solution operator, $\mathcal{A}$, can be expressed as an integral operator:

$$\mathcal{A}(f) = \int_\Omega G(\cdot, y) f(y)\, \mathrm{d}y = u,$$

where G is the Green's function associated with Eq. (2) (Evans, 2010, Chapt. 2). A neural operator is then trained to approximate the action of $\mathcal{A}$ using training data pairs $(f_1, u_1), \ldots, (f_M, u_M)$.

In general, recovering the solution operator is challenging, as it is often nonlinear and high-dimensional, and the available data may be scarce or noisy. Nevertheless, unlike inverse problems, which aim to recover source terms from solutions, the forward problem is usually well-posed. As we shall see, learning solution operators lead to new insights or applications that can complement inverse problem techniques, as described in two surveys (Stuart, 2010) and (Arridge et al., 2019).

1.1 What is a neural operator?

Neural operators (Kovachki et al., 2023; Lu et al., 2021a) are analogues of neural networks with infinite-dimensional inputs. Neural operators were introduced to generalize standard deep learning techniques to learn mappings between function spaces instead of between discrete vector spaces $\mathbb{R}^{d_1}$ to $\mathbb{R}^{d_L}$, where d_1 is the input dimension of a neural network and d_L is the output dimension. In its most traditional formulation, a fully connected neural network can be written as a succession of affine transformations and nonlinear activation functions as

$$\mathcal{N}(x) = \sigma(A_L(\cdots\sigma(A_1 x + b_1)\cdots) + b_L),$$

where $L \geq 1$ is the number of layers, A_i are the weight matrices, b_i are the bias vectors, and $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function, often chosen to be the ReLU function $\sigma(x) = \max\{x, 0\}$. Neural operators generalize this architecture, where the input and output of the neural network are functions instead of vectors. Hence, the input of a neural operator is a function $f : \Omega \to \mathbb{R}^{d_1}$, where $\Omega \subset \mathbb{R}^d$ is the domain of the function, and the output is a function $u : \Omega \to \mathbb{R}^{d_L}$. The neural operator is then defined as a composition of integral operators and nonlinear functions, which results in the following recursive definition at layer i:

$$u_{i+1}(x) = \sigma\left(\int_{\Omega_i} K^{(i)}(x, y)u_i(y)\,\mathrm{d}y + b_i(x)\right), \quad x \in \Omega_{i+1}, \tag{3}$$

where $\Omega_i \subset \mathbb{R}^{d_i}$ is a compact domain, b_i is a bias function, and $K^{(i)}$ is the kernel. The kernels and biases are then parameterized and trained similarly to standard neural networks. However, approximating the kernels or evaluating the integral operators could be computationally expensive. Hence, several neural operator architectures have been proposed to overcome these challenges, such as DeepONets (Lu et al., 2021a) and Fourier neural operators (Li et al., 2021a).

1.2 Where is operator learning relevant?

Operator learning has been successfully applied to many PDEs from different fields, including fluid dynamics with simulations of fluid flow turbulence in the

Navier–Stokes equations at high Reynolds number (Li et al., 2023b; Peng et al., 2022), continuum mechanics (You et al., 2022), astrophysics (Mao et al., 2023), quantum mechanics with the Schrödinger equation (Li et al., 2021a), and weather forecasting (Kurth et al., 2023; Lam et al., 2023). The following four types of applications might directly benefit from operator learning.

Speeding up numerical PDE solvers

First, one can use operator learning to build reduced-order models of complex systems that are computationally challenging to simulate with traditional numerical PDE solvers. For example, this situation arises in fluid dynamics applications such as modeling turbulent flows, which require a very fine discretization or the simulation of high dimensional PDEs. Moreover, specific problems in engineering require the evaluation of the solution operator many times, such as in the design of aircraft or wind turbines. In these cases, a fast but less accurate solver provided by operator learning may be used for forecasting or optimization. This is one of the main motivations behind Fourier neural operators in Li et al. (2021a). There are also applications of operator learning (Zheng et al., 2023) to speed up the sampling process in diffusion models or score-based generative models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021), which require solving complex differential equations. However, one must be careful when comparing performance against classical numerical PDE solvers, mainly due to the significant training time required by operator learning.

Parameter optimization

In our experience, the computational efficiency of operator learning is mainly seen in downstream applications such as parameter optimization. Once the solution operator has been approximated, it can be exploited in an inverse problem framework to recover unknown parameters of the PDE, which may be computationally challenging to perform with existing numerical PDE solvers. Additionally, neural operators do not rely on a fixed discretization as they are mesh-free and parameterized by a neural network that can be evaluated at any point. This property makes them suitable for solving PDEs on irregular domains or transferring the model to other spatial resolutions (Kovachki et al., 2023).

Benchmarking new techniques

Operator learning may also be used to benchmark and develop new deep learning models. As an example, one can design specific neural network architectures to preserve quantities of interest in PDEs, such as symmetries (Olver, 1993a), conservation laws (Evans, 2010, Sec. 3.4), and discretization independence. This could lead to efficient architectures that are more interpretable and generalize better to unseen data, and exploit geometric priors within datasets (Bronstein et al., 2021). Moreover, the vast literature on PDEs and numerical solvers can be leveraged to create datasets and assess the performance of these models in

various settings without requiring significant computational resources for training.

Discovering unknown physics

Last but not least, operator learning is helpful for the discovery of new physics (Lu et al., 2021a). Indeed, the solution operator of a PDE is often unknown, or one may only have access to a few data points without any prior knowledge of the underlying PDE. In this case, operator learning can be used to discover the PDE or a mathematical model to perform predictions. This can lead to new insights into the system's behavior, such as finding conservation laws, symmetries, shock locations, and singularities (Boullé et al., 2022a). However, the complex nature of neural networks makes them challenging to interpret or explain, and there are many future directions for making SciML more interpretative.

1.3 Organization of the paper

This paper is organized as follows. We begin in Section 2 by exploring the connections between numerical linear algebra and operator learning. Then, we review the main neural network architectures used to approximate operators in Section 3. In Section 4, we focus on the data acquisition process, a crucial step in operator learning. We discuss the choice of the distribution of source terms used to probe the system, the numerical PDE solver, and the number of training data. Along the way, we analyze the optimization pipeline, including the possible choices of loss functions, optimization algorithms, and assessment of the results. Finally, in Section 5, we conclude with a discussion of the remaining challenges in the field that include the development of open-source software and datasets, the theoretical understanding of the optimization procedure, and the discovery of physical properties in operator learning, such as symmetries and conservation laws.

2 From numerical linear algebra to operator learning

There is a strong connection between operator learning and the recovery of structured matrices from matrix-vector products. Suppose one aims to approximate the solution operator associated with a linear PDE using a single layer neural operator in the form of Eq. (3) without the nonlinear activation function. Then, after discretizing the integral operator using a quadrature rule, it can be written as a matrix-vector product, where the integral kernel $K : \Omega \times \Omega \to \mathbb{R}$ is approximated by a matrix $A \in \mathbb{R}^{N \times N}$. Moreover, the structure of the matrix is inherited from the properties of the Green's function (see Table 1). The matrix's underlying structure—whether it is low-rank, circulant, banded, or hierarchical low-rank—plays a crucial role in determining the efficiency and approach of the recovery process. This section describes the matrix recovery problem as a

helpful way to gain intuition about operator learning and the design of neural operator architectures (see Section 3). Another motivation for recovering structured solution operators is to ensure that the neural operators are fast to evaluate, which is essential in applications involving parameter optimization and benchmarking (Kovachki et al., 2023).

TABLE 1 Solution operators associated with linear PDEs can often be represented as integral operators with a kernel called a Green's function. The properties of a linear PDE induce different structures on the Green's function, such as translation-invariant or off-diagonal low-rank. When these integral operators are discretized, one forms a matrix-vector product, and hence the matrix recovery problem can be viewed as a discrete analogue of operator learning. The dash in the first column and row means no PDE has a solution operator with a globally smooth kernel.

Property of the PDE	Solution operator's kernel	Matrix structure
—	Globally smooth	Low rank
Periodic BCs & const. coeffs	Convolution kernel	Circulant
Localized behavior	Off-diagonal decay	Banded
Elliptic / Parabolic	Off-diagonal low rank	Hierarchical

Consider an unknown matrix $A \in \mathbb{R}^{N\times N}$ with a known structure such as rank-k or banded. We assume that the matrix A is a black box and cannot be seen, but that one can probe A for information via matrix-vector products, i.e., the maps $x \mapsto Ax$ and $x \mapsto A^\top x$, with $A^\top$ representing the transpose of A. The matrix recovery problem is the task of approximating the matrix A using as few queries to $x \mapsto Ax$ and $x \mapsto A^\top x$ as possible. Every matrix with N columns can be deduced in a maximum of N matrix-vector product queries, as Ae_j for $1 \leq j \leq N$ returns the jth column, where e_j denotes the jth standard basis vector. However, if the matrix A has a specific structure such as low-rank, circulant, banded, or hierarchical low-rank, it is often possible to recover A using far fewer queries. This section describes how to recover structured matrices efficiently using matrix-vector products. We prefer doing matrix recovery with Gaussian random vectors because the infinite-dimensional analogue of these vectors are random functions drawn from a Gaussian process, which is a widespread choice of training input data in operator learning.

2.1 Low rank matrix recovery

Let $A \in \mathbb{R}^{N\times N}$ be a rank-k matrix, then it can be expressed for some $C \in \mathbb{R}^{N\times k}$ and $R \in \mathbb{R}^{k\times N}$ as

$$A = CR.$$

Halikias and Townsend (2023) showed that at least $2k$ queries are required to capture the k-dimensional row and column spaces and deduce A. Halko et al. (2011); Martinsson and Tropp (2020) introduced the randomized singular value

decomposition (SVD) as a method to recover a rank-k matrix with probability one in $2k$ matrix-vector products with random Gaussian vectors. The randomized SVD can be expressed as a recovery algorithm in Algorithm 1.

Algorithm 1 Randomized singular value decomposition.

1: Draw a random matrix $X \in \mathbb{R}^{N\times k}$ with i.i.d. standard Gaussian entries.
2: Perform k queries with A: $Y = AX$.
3: Compute the QR factorization of $Y = QR$.
4: Perform k queries with $A^\top$: $Z = A^\top Q$.
5: Return $A = QZ^\top$.

A randomized algorithm is crucial for low-rank matrix recovery to prevent input vectors from lying within the $N - k$ dimensional nullspace of A. Hence, recovering any low-rank matrix with a deterministic algorithm using fixed input vectors is impossible. Then, for the rank-k matrix recovery problem, Algorithm 1 recovers A with probability one. A small oversampling parameter $p \geq 1$ is used for numerical stability, such as $p = 5$. This means that $X \in \mathbb{R}^{N\times(k+p)}$, preventing the chance that a random Gaussian vector might be highly aligned with the nullspace of A.

A convenient feature of Algorithm 1 is that it also works for matrices with numerical rank[1] k, provided that one uses a random matrix X with $k + p$ columns. In particular, a simplified statement of Halko et al. (2011, Thm. 10.7) shows that the randomized SVD recovers a near-best low-rank matrix in the sense that

$$\mathbb{P}\left[\|A - QZ^\top\|_{\mathrm{F}} \leq \left(1 + 15\sqrt{k+5}\right) \min_{\mathrm{rank}(A_k)\leq k} \|A - A_k\|_{\mathrm{F}}\right] \geq 0.999,$$

where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm of A. While other random embeddings can be used to probe A, Gaussian random vectors give the cleanest probability bounds (Martinsson and Tropp, 2020). Moreover, ensuring that the entries in each column of X have some correlation and come from a multivariable Gaussian distribution allows for the infinite-dimensional extension of the randomized SVD and its application to recover Hilbert–Schmidt operators (Boullé and Townsend, 2022, 2023). This analysis allows one to adapt Algorithm 1 to recover solution operators with low-rank kernels.

Low-rank matrix recovery is one of the most straightforward settings to motivate DeepONet (see Section 3.1). One of the core features of DeepONet is to use the trunk net to represent the action of a solution operator on a set of basis functions generated by the so-called branch net. Whereas in low-rank matrix recovery, we often randomly draw the columns of X as input vectors, DeepONet is trained with these functions. However, like DeepONet, low-rank matrix

[1] For a fixed $0 < \epsilon < 1$, we say that a matrix A has numerical rank k if $\sigma_{k+1}(A) < \epsilon\sigma_1(A)$ and $\sigma_k(A) \geq \epsilon\sigma_1(A)$, where $\sigma_1(A) \geq \sigma_2(A) \geq \ldots \geq \sigma_N(A) \geq 0$ are the singular values of A.

recovery is constructing an accurate approximant whose action is on these vectors. Many operators between function spaces can often be represented to high accuracy with DeepONet in the same way that the kernels of solution operators associated with linear PDEs often have algebraically fast decaying singular values.

2.2 Circulant matrix recovery

The FNO (Li et al., 2021a) structure is closely related to circulant matrix recovery. Consider an $N \times N$ circulant matrix C_c, which is parameterized a vector $c \in \mathbb{R}^N$ as follows:

$$C_c = \begin{bmatrix} c_0 & c_{N-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{N-1} & \ddots & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{N-2} & \ddots & \ddots & \ddots & c_{N-1} \\ c_{N-1} & c_{N-2} & \cdots & c_1 & c_0 \end{bmatrix}.$$

To recover C_c with a random Gaussian vector g, we recall that C_c can be interpreted as a multiplication operator in the Fourier basis. By associativity of multiplication, we have

$$C_c g = C_g c.$$

If we perform the matrix-vector product query $y = C_c g$, we can find the vector c by solving the linear system $C_g c = y$. Since c completely defines C_c, we have recovered the circulant matrix. Moreover, the linear system $C_g c = y$ can be solved efficiently using the fast Fourier transform (FFT) in $\mathcal{O}(N \log N)$ operations. A convenient feature of circulant matrices is that given a new vector $x \in \mathbb{R}^N$, one can compute $C_c x$ in $\mathcal{O}(N \log N)$ operations using the FFT.

Circulant matrix recovery motivates Fourier neural operators. Hence, FNOs leverage the fast Fourier transform to efficiently parameterize the kernel of a solution operator, essentially capturing the operator in a spectral sense. Similarly, circulant matrices are diagonalized by the discrete Fourier transform matrix. The infinite-dimensional analogue of a circulant matrix is a solution operator with a periodic and translation invariant kernel, and this is the class of solution operators for which the FNO assumptions are fully justified. FNOs are extremely fast to evaluate because of their structure, making them popular for parameter optimization and favorable for benchmarking against reduced-order models.

2.3 Banded matrix recovery

We now consider a banded matrix $A \in \mathbb{R}^{N\times N}$ with a fixed bandwidth w, i.e.,

$$A_{ij} = 0, \quad \text{if } |i-j| > w.$$

The matrix A can be recovered with $w+2$ matrix-vector products, but not fewer, using the $w+2$ columns of the following matrix as input vectors:

$$\begin{bmatrix} I_{w+2} & \cdots & I_{w+2} \end{bmatrix}^\top,$$

where I_{w+2} is the $(w+2)\times(w+2)$ identity matrix. Since every wth column has disjoint support, these input vectors recover the columns of A. Of course, this also means that a A can be recovered with $w+2$ Gaussian random vectors.

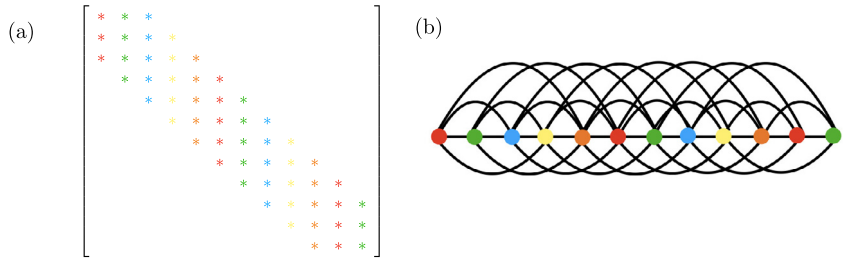

FIGURE 2 (a) A generic 12×12 banded matrix with bandwidth 2, with a maximum of 5 diagonals, and the corresponding graph (b). Here, each vertex is a column of the banded matrix, and two vertices are connected if their corresponding columns do not have disjoint support. The coloring number of 5 determines the minimum number of matrix-vector products needed to recover the structure. Generally, an $N \times N$ banded matrix with bandwidth w can be recovered in $2w+1$ matrix-vector products.

There is a way to understand how many queries one needs as a graph-coloring problem. Consider the graph of size N, corresponding to an $N \times N$ banded matrix with bandwidth w, where two vertices are connected if their corresponding columns do not have disjoint support (see Fig. 2). Then, the minimum number of matrix-vector product queries needed to recover A is the graph coloring number of this graph.[2] One can see why this is the case because all the columns with the same color can be deduced simultaneously with a single matrix-vector product as they must have disjoint support.

Banded matrix recovery motivates Graph neural operators (GNOs), which we will describe later in Section 3.4, as both techniques exploit localized structures within data. GNOs use the idea that relationships in nature are local and can be represented as graphs with no faraway connections. By only allowing local connections, GNOs can efficiently represent solution operators corresponding to local solution operators, mirroring the way banded matrices

[2] Recall that the coloring number of a graph is the minimum number of colors required to color the vertices so that no two vertices connected by an edge are identically colored.

are concentrated on the diagonal. Likewise, with a strong locality, GNOs are relatively fast to evaluate, making them useful for parameter optimization and benchmarking. However, they may underperform if the bandwidth increases or the solution operator is not local.

2.4 Hierarchical low rank matrix recovery

An $N \times N$ rank-k hierarchical off-diagonal low rank (HODLR) matrix, denoted as $H_{N,k}$, is a structure that frequently appears in the context of discretized solution operators associated with elliptic and parabolic PDEs (Hackbusch et al., 2004). To understand its recursive structure, we assume N to be a power of 2 and illustrate the structure in Fig. 3(a).

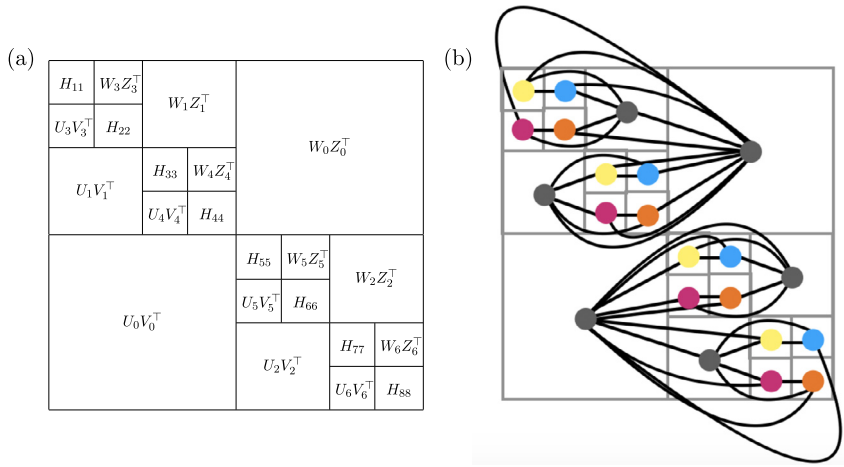

FIGURE 3 (a) A HODLR matrix $H_{N,k}$ after three levels of partitioning. Since $H_{N,k}$ is a rank-k HODLR matrix, U_i, V_i, W_i, and Z_i have at most k columns. The matrices A_{ii} are themselves rank-k HODLR matrices of size $N/8 \times N/8$ and can be further partitioned. (b) Graph corresponding to a hierarchical low-rank matrix with three levels. Here, each vertex is a low-rank block of the matrix, where two vertices are connected if their low-rank blocks occupy the same row. At each level, the number of required matrix-vector input probes to recover that level is proportional to the coloring number of the graph when restricted to submatrices of the same size. In this case, the submatrices that are identically colored can be recovered simultaneously.

Since a rank-k matrix requires $2k$ matrix-vector product queries to be recovered (see Section 2.1), a naive approach to deducing $H_{N,k}$ is to use $2k$ independent queries on each submatrix. However, one can show that some of the submatrices of $H_{N,k}$ can be recovered concurrently using the same queries. We use a graph coloring approach (Levitt and Martinsson, 2022b) to determine which submatrices can be recovered concurrently. This time, we consider the graph where each vertex is a low-rank submatrix of $H_{N,k}$ and connect two vertices if their corresponding low-rank submatrices occupy the same column as in Fig. 3(b). The low-rank submatrices that are identically colored at each level

can be recovered concurrently in only $2k$ queries. Hence, it can be shown that an $N \times N$ hierarchical rank-k matrix can be recovered in fewer than $10k\lceil \log_2(N) \rceil$ matrix-vector products (Halikias and Townsend, 2023). The precise coloring of the graph in Fig. 3(b) can also be used to derive a particular algorithm for hierarchical matrix recovery known as peeling (Levitt and Martinsson, 2022a,b; Lin et al., 2011; Martinsson, 2011). These peeling algorithms have been recently generalized to the infinite-dimensional setting by Boullé et al. (2023).

HODLR recovery can be seen as the simplest version of a multipole graph neural operator (MGNO) (see Section 3.5) as both emphasize the importance of capturing operators at multiple scales. MGNOs are based on a hierarchical graph with interactions at different scales or levels (see Section 3.4). By incorporating local (near-field) and global (far-field) interactions, MGNOs can effectively learn complex patterns. MGNOs are often great at representing solution operators due to their multiscale nature. The price to pay is that the final neural operator can be computationally expensive to evaluate, and it is a complicated structure to implement.

3 Neural operator architectures

In this section, we review the main neural operator architectures used in the literature, namely DeepONets (Lu et al., 2021a), Fourier neural operators (Li et al., 2021a), and Deep Green networks (Gin et al., 2021; Boullé et al., 2022a). We also refer to the recent survey by Goswami et al. (2023) for a review of the different neural operator architectures and their applications. Each of these architectures employs different discretization and approximation techniques to make the neural operator more efficient and scalable by enforcing certain structures on the kernel such as low-rank, periodicity, translation invariance, or hierarchical low-rank structure (see Table 2).

TABLE 2 Summary table of neural operator architectures, describing the property assumption on the operator along with the discretization of the integral kernels.

Neural operators	Property of the operator	Kernel parameterization
DeepONet	Low-rank	Branch and trunk networks
FNO	Translation-invariant	Fourier coefficients
GreenLearning	Linear	Rational neural network
DeepGreen	Semilinear	Kernel matrix
Graph neural operator	Diagonally dominant	Message passing network
Multipole GNO	Off-diagonal low rank	Neural network

Most neural operator architectures also come with theoretical guarantees on their approximation power. These theoretical results essentially consist of universal approximation properties for neural operators (Chen and Chen, 1995;

Kovachki et al., 2023; Lu et al., 2021a), in a similar manner as neural networks (DeVore, 1998), and quantitative error bounds based on approximation theory to estimate the size, i.e., the number of trainable parameters, of a neural operator needed to approximate a given operator between Banach spaces to within a prescribed accuracy (Lanthaler et al., 2022; Yarotsky, 2017).

3.1 Deep operator networks

Deep Operator Networks (DeepONets) are a promising model for learning nonlinear operators and capturing the inherent relationships between input and output functions (Lu et al., 2021a). They extend the capabilities of traditional deep learning techniques by leveraging the expressive power of neural networks to approximate operators in differential equations, integral equations, or more broadly, any functional maps from one function space to another. A key theoretical motivation for DeepONet is the universal operator approximation theorem (Chen and Chen, 1995; Lu et al., 2021a). This result can be seen as an infinite dimensional analogue of the universal approximation operator for neural networks (Cybenko, 1989; Hornik, 1991), which guarantee that a sufficiently wide neural network can approximate any continuous function to any accuracy. Since the introduction of DeepONets by Lu et al. (2021a), several research works focused on deriving error bounds for the approximation of nonlinear operators by DeepONets in various settings, such as learning the solution operator associated with Burger's equation or the advection-diffusion equation (Deng et al., 2022), and the approximation of nonlinear parabolic PDEs (De Ryck and Mishra, 2022; Lanthaler et al., 2022).

A DeepONet is a two-part deep learning network consisting of a branch network and a trunk network. The branch net encodes the operator's input functions f into compact, fixed-size latent vectors $b_1(f(x_1), \dots, f(x_m)), \dots, b_p(f(x_1), \dots, f(x_m))$, where $\{x_i\}_{i=1}^m$ are the sensor points at which the input functions are evaluated. The trunk net decodes these latent vectors to produce the final output function at the location $y \in \Omega$ as

$$\mathcal{N}(f)(y) = \sum_{k=1}^{p} b_k(f(x_1), \dots, f(x_m)) t_k(y).$$

A schematic of a deep operator network is given in Fig. 4. The defining feature of DeepONets is their ability to handle functional input and output, thus enabling them to learn a wide array of mathematical operators effectively. It's worth mentioning that the branch network and the trunk network can have distinct neural network architectures tailored for different purposes, such as performing a feature expansion on the input of the trunk network as $y \to \begin{pmatrix} y & \cos(\pi y) & \sin(\pi y) & \dots \end{pmatrix}$ to take into account any potential oscillatory patterns in the data (Di Leoni et al., 2023). Moreover, while the interplay of the branch and trunk networks is crucial, the output of a DeepONet does not

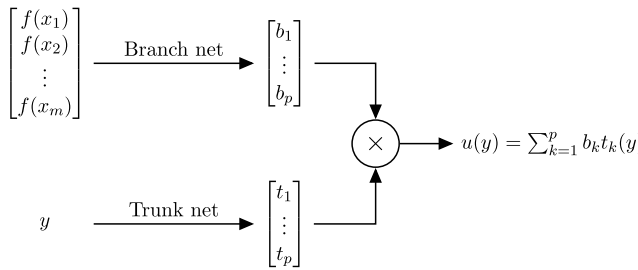

FIGURE 4 Schematic diagram of a deep operator network (DeepONet). A DeepONet parametrizes a neural operator using a branch network and a truncation (trunk) network. The branch network encodes the input function f as a vector of p features, which is then multiplied by the trunk network to yield a rank-p representation of the solution u.

necessarily depend on the specific input points but rather on the global property of the entire input function, which makes it suitable for learning operator maps.

One reason behind the performance of DeepONet might be its connection with the low-rank approximation of operators and the SVD (see Section 2.1). Hence, one can view the trunk network as learning a basis of functions $\{t_k\}_{k=1}^{p}$ that are used to approximate the operator, while the branch network expresses the output function in this basis by learning the coefficients $\{b_k\}_{k=1}^{p}$. Moreover, the branch network can be seen as a feature extractor, which encodes the input function into a compact representation, thus reducing the problem's dimensionality to p, where p is the number of branch networks. Additionally, several architectures, namely the POD-DeepONet (Lu et al., 2022) and SVD-DeepONet (Venturi and Casey, 2023), have been proposed to strengthen the connections between DeepONet and the SVD of the operator and increase its interpretability.

A desirable property for a neural operator architecture is to be discretization invariant in the sense that the model can act on any discretization of the source term and be evaluated at any point of the domain (Kovachki et al., 2023). This property is crucial for the generalization of the model to unseen data and the transferability of the model to other spatial resolutions. While DeepONets can be evaluated at any location of the output domain, DeepONets are not discretization invariant in their original formulation by Lu et al. (2021a) as the branch network is evaluated at specific points of the input domain (see Fig. 4). However, this can be resolved using a low-rank neural operator (Kovachki et al., 2023), sampling the input functions at local spatial averages (Lanthaler et al., 2022), or employing a principal component analysis (PCA) alternative of the branch network (de Hoop et al., 2022).

The training of DeepONets is performed using a supervised learning process. It involves minimizing the mean-squared error between the predicted output $\mathcal{N}(f)(y)$ and the actual output of u the operator on the training functions at

random locations $\{y_j\}_{j=1}^n$, i.e.,

$$\min_{\theta\in\mathbb{R}^N} \frac{1}{|\text{data}|} \sum_{(f,u)\in\text{data}} \frac{1}{n}\sum_{j=1}^{n} |\mathcal{N}(f)(y_j) - u(y_j)|^2. \tag{4}$$

The term inside the first sum approximates the integral of the mean-squared error, $|\mathcal{N}(f)-u|^2$, over the domain Ω using Monte-Carlo integration. The optimization is typically done via backpropagation and gradient descent algorithms, which are the same as in traditional neural networks. Importantly, DeepONets allow for different choices of loss functions, depending on the problem. For example, mean squared error is commonly used for regression tasks, but other loss functions might be defined to act as a regularizer and incorporate prior physical knowledge of the problem (Goswami et al., 2022; Wang et al., 2021b). The selection of an appropriate loss function is a crucial step in defining the learning process of these networks and has a substantial impact on their performance (see Section 4.2.1).

DeepONet has been successfully applied and adapted to a wide range of problems, including predicting cracks in fracture mechanics using a variational formulation of the governing equations (Goswami et al., 2022), simulating the New York-New England power grid behavior with a probabilistic and Bayesian framework to quantify the uncertainty of the trajectories (Moya et al., 2023), as well as predicting linear instabilities in high-speed compressible flows with boundary layers (Di Leoni et al., 2023).

3.2 Fourier neural operators

Fourier neural operators (FNOs) (Li et al., 2021a; Kovachki et al., 2023) are a class of neural operators motivated by Fourier spectral methods. FNOs have found their niche in dealing with high-dimensional PDEs, which are notoriously difficult to solve using traditional numerical methods due to the curse of dimensionality. They've demonstrated significant success in learning and predicting solutions to various PDEs, particularly those with periodic boundary conditions or those that can be transformed into the spectral domain via Fourier transform. This capability renders FNOs an invaluable tool in areas where PDEs play a central role, such as fluid dynamics, quantum mechanics, and electromagnetism.

The main idea behind FNOs is to choose the kernels $K^{(i)}$ in Eq. (3) as translation-invariant kernels satisfying $K^{(i)}(x,y) = k^{(i)}(x-y)$ (provided the input and output domains are torus) such that the integration of the kernel can be performed efficiently as a convolution using the Fast Fourier Transform (FFT) (Cooley and Tukey, 1965), i.e., multiplication in the feature space of Fourier coefficients. Hence, the integral operation in Eq. (3) can be performed as

$$\int_{\Omega_i} k^{(i)}(x-y)u_i\,\mathrm{d}y = \mathcal{F}^{-1}(\mathcal{F}(k^{(i)})\mathcal{F}(u_i))(x) = \mathcal{F}^{-1}(\mathcal{R}\cdot\mathcal{F}(u_i))(x), \quad x\in\Omega_i,$$

where $\mathcal{F}$ denotes the Fourier transform and $\mathcal{F}^{-1}$ its inverse. The kernel $K^{(i)}$ is parametrized by a periodic function $k^{(i)}$, which is discretized by a (trainable) weight vector of Fourier coefficients $\mathcal{R}$, and truncated to a finite number of Fourier modes. Then, if the input domain is discretized uniformly with m sensor points, and the vector $\mathcal{R}$ contains at most $k_{\max} \leq m$ modes, the convolution can be performed in quasilinear complexity in $\mathcal{O}(m \log m)$ operations via the FFT. This is a significant improvement over the $\mathcal{O}(m^2)$ operations required to evaluate the integral in Eq. (3) using a quadrature rule. In practice, one can restrict the number of Fourier modes to $k_{\max} \ll m$ without significantly affecting the accuracy of the approximation whenever the input and output functions are smooth so that their representation in the Fourier basis enjoy rapid decay of the coefficients, thus further reducing the computational and training complexity of the neural operator.

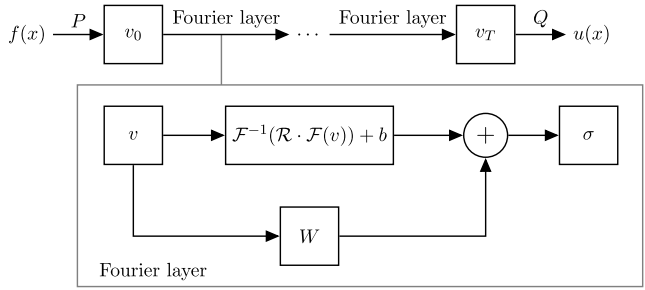

FIGURE 5 Schematic diagram of a Fourier neural operator (FNO). The networks P and Q, respectively, lift the input function f to a higher dimensional space and project the output of the last Fourier layer to the output dimension. An FNO mainly consists of a succession of Fourier layers, which perform the integral operations in neural operators as a convolution in the Fourier domain and component-wise composition with an activation function σ.

We display a diagram of the architecture of an FNO in Fig. 5. The input function f is first lifted to a higher dimensional space by a neural network P. Then, a succession of Fourier layers is applied to the lifted function, which is parametrized by a vector of Fourier coefficients $\mathcal{R}_i$, a bias vector b_i, and a weight matrix W. Then, the output of the FNO at the ith layer is given by

$$v_i = \sigma(W_i v_{i-1} + \mathcal{F}^{-1}(\mathcal{R}_i \cdot \mathcal{F}(u_{i-1})) + b_i),$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function whose action is defined component-wise, often chosen to be the ReLU function. The weight matrix W_i and bias vector b_i perform a linear transformation of the input function v_i. After the last Fourier layer, the output of the FNO is obtained by applying a final neural network Q on the output of the last Fourier layer to project it to the output dimension.

The training of FNOs, like DeepONets, is carried out via a supervised learning process. It typically involves minimizing a loss function that measures the

discrepancy between the predicted and the true output of the operator on the input functions with respect to the trainable parameters of the neural network as in Eq. (4). Here, one needs to perform backpropagation through the Fourier layers, which is enabled by the implementation of fast GPU differentiable FFTs (Mathieu et al., 2014) in deep learning frameworks such as PyTorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2016).

While FNOs have been proposed initially to alleviate the computational expense of performing integral operations in neural operators by leveraging the FFT, they have a distinctive advantage in learning operators where computations in the spectral domain are more efficient or desirable. This arises naturally when the target operator, along with the input and output functions, are smooth so that their representation as Fourier coefficients decay exponentially fast, yielding an efficient truncation. Hence, by selecting the architecture of the FNO appropriately, such as the number of Fourier modes $k_{\max}$, or the initialization of the Fourier coefficients, one can obtain a neural operator that preserves specific smoothness properties. However, when the input or output training data is not smooth, FNO might suffer from Runge's phenomenon near discontinuities (de Hoop et al., 2022).

One main limitation of the FNO architectures is that the FFT should be performed on a uniform grid and rectangular domains, which is not always the case in practice. This can be overcome by applying embedding techniques to transform the input functions to a uniform grid and extend them to simple geometry, using a Fourier analytic continuation technique (Bruno et al., 2007). Recently, several works have been proposed to extend the FNO architecture to more general domains, such as using a zero padding, linear interpolation (Lu et al., 2022), or encoding the geometry to a regular latent space with a neural network (Li et al., 2022, 2023a). However, this might lead to a loss of accuracy and additional computational cost. Moreover, the FFT is only efficient for approximating translation invariant kernels, which do not occur when learning solution operators of PDEs with nonconstant coefficients.

Other related architectures aim to approximate neural operators directly in the feature space, such as spectral neural operators (SNO) (Fanaskov and Oseledets, 2022), which are based on spectral methods and employ a simple feedforward neural network to map the input function, represented as a vector of Fourier or Chebyshev coefficients to an output vector of coefficients. Finally, Raonic et al. (2023) introduce convolutional neural operators (CNOs) to alleviate the aliasing phenomenon of convolutional neural networks (CNNs) by learning the mapping between bandlimited functions. Contrary to FNOs, CNOs parameterize the integral kernel on a $k \times k$ grid and perform the convolution in the physical space as

$$\int_{\Omega} k(x-y) f(y)\,\mathrm{d}y = \sum_{i,j=1}^{k} k_{ij} f(x - z_i j), \quad x \in \Omega,$$

where z_{ij} are the grid points.

Similarly to DeepONets, Fourier neural operators are universal approximators, in the sense that they are dense in the space of continuous operators (Bhattacharya et al., 2021; Kovachki et al., 2023, 2021). However, even while being universal approximators, FNOs could, in theory, require a huge number of parameters to approximate a given operator to a prescribed accuracy $\epsilon > 0$. As an example, Kovachki et al. (2021) showed that the size of the FNO must grow exponentially fast as ϵ decreases to approximate any operator between rough functions whose Fourier coefficients decay only at a logarithmic rate. Fortunately, these pessimistic lower bounds are not observed in practice when learning solution operators associated with PDEs. Indeed, in this context, one can exploit PDE regularity theory and Sobolev embeddings to derive quantitative bounds on the size of FNOs for approximating solution operators that only grow sublinearly with the error. Here, we refer to the analysis of Darcy flow and the two-dimensional Navier–Stokes equations by Kovachki et al. (2021).

3.3 Deep Green networks

Deep Green networks (DGN) employ a different approach compared to DeepONets and FNOs to approximate solution operators of PDEs (Gin et al., 2021; Boullé et al., 2022a). Instead of enforcing certain properties on the integral kernel in Eq. (3), such as being low-rank (DeepONet) or translation-invariant (FNO), DGN learns the kernel directly in the physical space. Hence, assume that the underlying differential operator is a linear boundary value problem of the form

$$\mathcal{L}u = f \text{ in } \Omega, \quad \text{and} \quad u = 0 \text{ on } \partial\Omega. \tag{5}$$

Under suitable regularity assumptions on operator $\mathcal{L}$ (e.g., uniform ellipticity or parabolicity), the solution operator $\mathcal{A}$ can be expressed as an integral operator with a Green kernel $G : \Omega \times \Omega \to \mathbb{R} \cup \{\infty\}$ as (Evans, 2010; Boullé et al., 2022b; Boullé and Townsend, 2023)

$$\mathcal{A}(f)(x) = u(x) = \int_\Omega G(x, y) f(y) \, \mathrm{d}y, \quad x \in \Omega.$$

Boullé et al. (2022a) introduced GreenLearning networks (GL) to learn the Green kernel G directly from data. The main idea behind GL is to parameterize the kernel G as a neural network $\mathcal{N}$ and minimize the following relative mean-squared loss function to recover an approximant to G:

$$\min_{\theta \in \mathbb{R}^N} \frac{1}{|\text{data}|} \sum_{(f,u) \in \text{data}} \frac{1}{\|u\|^2_{L^2(\Omega)}} \int_\Omega \left(u(x) - \int_\Omega \mathcal{N}(x, y) f(y) \, \mathrm{d}y \right)^2 \mathrm{d}x. \tag{6}$$

Once trained, the network $\mathcal{N}$ can be evaluated at any point in the domain, similarly to FNO and DON. A key advantage of this method is that it provides a

more interpretable model, as the kernel can be visualized and analyzed to recover properties of the underlying differential operators (Boullé et al., 2022a). However, this comes at the cost of higher computational complexity, as the integral operation in Eq. (6) must be computed accurately using a quadrature rule and typically requires $\mathcal{O}(m^2)$ operations, as opposed to the $\mathcal{O}(m \log m)$ operations required by FNOs, where m is the spatial discretization of the domain Ω.

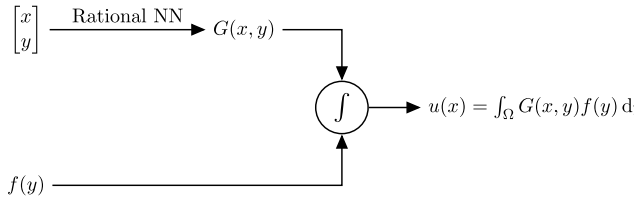

FIGURE 6 Schematic diagram of a GreenLearning network (GL), which approximates the integral kernel (Green's function) associated with linear PDEs using rational neural networks.

As Green's functions may be unbounded or singular, Boullé et al. (2022a) propose to use a rational neural network (Boullé et al., 2020) to approximate the Green kernel. A rational neural network is a neural network whose activation functions are rational functions, defined as the ratio of two polynomials whose coefficients are learned during the training phase of the network. This choice of architecture is motivated by the fact that rational networks have higher approximation power than the standard ReLU networks (Boullé et al., 2020), in the sense that they require exponentially fewer layers to approximate continuous functions within a given accuracy and may take arbitrary large values, which is a desirable property for approximating Green kernels. A schematic diagram of a GL architecture is available in Fig. 6.

When the underlying differential operator is nonlinear, the solution operator $\mathcal{A}$ cannot be written as an integral operator with a Green's function. In this case, Gin et al. (2021) propose to learn the solution operator $\mathcal{A}$ using a dual autoencoder architecture Deep Green network (DGN), which is a neural operator architecture that learns an invertible coordinate transform map that linearizes the nonlinear boundary value problem. The resulting linear operator is approximated by a matrix, which represents a discretized Green's function but could also be represented by a neural network if combined with the GL technique. This approach has been successfully applied to learn the solution operator of the nonlinear cubic Helmholtz equation non-Sturm–Liouville equation and discover an underlying Green's function (Gin et al., 2021).

Other deep learning-based approaches (Lin et al., 2023; Peng et al., 2023; Sun et al., 2023) have since been introduced to recover Green's functions using deep learning, but they rely on a PINN technique in the sense that they require the knowledge of the underlying PDE operator. Finally, Stepaniants (2023) proposes to learn the Green kernel associated with linear partial differential op-

erators using a reproducible kernel Hilbert space (RKHS) framework, which leads to a convex loss function.

3.4 Graph neural operators

As described in Section 3.3, solution operators associated with linear, elliptic, or parabolic PDEs of the form $\mathcal{L}u = f$ can be written as an integral operator with a Green kernel G (Evans, 2010, Sec. 2.2.4). For simplicity, we consider Green kernels associated with uniformly elliptic operators in divergence form defined on a bounded domain Ω in spatial dimension $d \geq 3$:

$$\mathcal{L}u = -\text{div}(A(x)\nabla u) = f, \quad \text{on } \Omega \subset \mathbb{R}^d, \tag{7}$$

where $A(x)$ is a bounded coefficient matrix satisfying the uniform ellipticity condition $A(x)\xi \cdot \xi \geq \lambda|\xi|^2$ for all $x \in \Omega$ and $\xi \in \mathbb{R}^d$, for some $\lambda > 0$. In this section, we present a neural operator architecture that takes advantage of the local structure of the Green kernel associated with Eq. (7), inferred by PDE regularity theory.

This architecture is called graph neural operator (GNO) (Li et al., 2020a) and is inspired by graph neural network (GNN) models (Scarselli et al., 2008; Wu et al., 2020; Zhou et al., 2020). It focuses on capturing the Green kernel's short-range interactions to reduce the integral operation's computational complexity in Eq. (3). The main idea behind GNO is to perform the integral operation in Eq. (5) locally on a small ball of radius r, $B(x, r)$, around x for each $x \in \Omega$ as follows:

$$\mathcal{A}(f)(x) = u(x) \approx \int_{B(x,r)} G(x, y) f(y)\, \mathrm{d}y, \quad x \in \Omega. \tag{8}$$

Here, Li et al. (2020a) propose to discretize the domain Ω using a graph, whose nodes represent discretized spatial locations, and use a message passing network architecture (Gilmer et al., 2017) to perform an average aggregation of the nodes as in Eq. (8). The approach introduced by Li et al. (2020a) aims to approximate the restriction G_r to the Green's function G on a band of radius r along the diagonal of the domain $\Omega \times \Omega$, defined as

$$G_r(x, y) = \begin{cases} G(x, y), & \text{if } |x - y| \leq r, \\ 0, & \text{otherwise,} \end{cases}$$

where $|\cdot|$ is the Euclidean distance in $\mathbb{R}^d$.

This neural architecture is justified by the following pointwise bound satisfied by the Green's function and proven by Grüter and Widman (1982, Thm. 1.1):

$$|G(x, y)| \leq C(d, A)|x - y|^{2-d}, \quad x, y \in \Omega, \tag{9}$$

where Ω is a compact domain in $\mathbb{R}^d$ for $d \geq 3$, and C is a constant depending only on d and the coefficient matrix $A(x)$. Similar bounds have been derived in spatial dimension $d = 2$ by Cho et al. (2012); Dong and Kim (2009) and for Green's functions associated with time-dependent, parabolic, PDEs (Hofmann and Kim, 2004; Cho et al., 2008). Then, integrating Eq. (9) over the domain $\Gamma_r := \{(x, y) \in \Omega \times \Omega : |x - y| > r\}$ yields a bound on the approximation error between G and G_r that decays algebraically fast as r increases:

$$\begin{aligned} \|G - G_r\|_{L^2(\Omega\times\Omega)} &= \left(\int_{\Gamma_r} |G(x, y)|^2 \, \mathrm{d}x \, \mathrm{d}y\right)^{1/2} \\ &\leq C(d, A) \left(\int_{\Gamma_r} r^{4-2d} \, \mathrm{d}x \, \mathrm{d}y\right)^{1/2} \\ &\leq |\Omega| C(d, A) r^{2-d}. \end{aligned}$$

This implies that the Green's function can be well approximated by a bandlimited kernel G_r and that the approximation error bound improves in high dimensions. To illustrate this, we plot in Fig. 7 the Green's function associated with the one-dimensional Poisson equation on $\Omega = [0, 1]$ with homogeneous Dirichlet boundary conditions, along with the error between the Green's function G and its truncation G_r along a bandwidth of radius r along the diagonal of the domain.

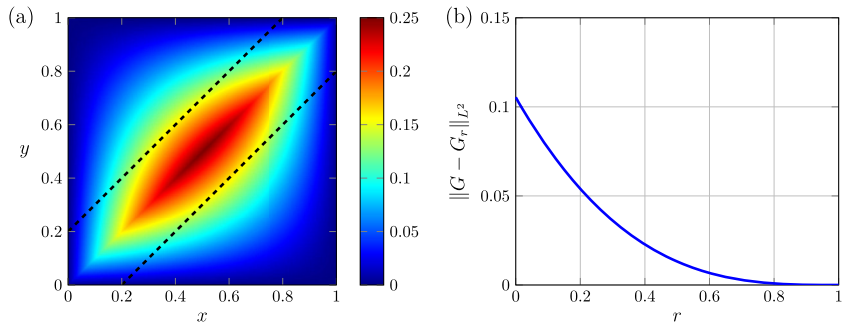

FIGURE 7 (a) Green's function associated with the one-dimensional Poisson equation on $\Omega = [0, 1]$ with homogeneous Dirichlet boundary conditions. The dashed lines highlight a band of radius $r = \sqrt{2}/10$ around the diagonal. (b) L^2-norm of the error between the Green's function G and its truncation G_r along a bandwidth of radius r along the diagonal of the domain.

3.5 Multipole graph neural operators

Multipole graph neural operator (MGNO) has been introduced by Li et al. (2020b) and is a class of multiscale networks that extends the graph neural operator architecture described in Section 3.4 to capture the long-range interactions of the Green kernel. The main idea behind MGNO is to decompose the Green

kernel G into a sum of low-rank kernels as $G = K_1 + \cdots + K_L$, which approximates the short and wide-range interactions in the PDEs. This architecture is motivated by the same reasons that led to the development of hierarchical low-rank matrices (see Section 2.4), such as the fast multipole method (Greengard and Rokhlin, 1997; Ying et al., 2004). It allows for the evaluation of the integral operation in Eq. (3) in linear complexity.

MGNO is based on low-rank approximations of kernels, similar to DeepONets or low-rank neural operators (see Section 3.1 and Li et al., 2020a), but is more flexible than vanilla DeepONets since it does not require the underlying kernels to be low-rank. Hence, if we consider a Green's function G associated with a uniformly elliptic PDE in the form of Eq. (7), then Weyl's law (Weyl, 1911; Canzani, 2013; Minakshisundaram and Pleijel, 1949) states that the eigenvalues of the solution operator associated with Eq. (7) decay at an algebraic rate of $\lambda_n \sim cn^{-2/d}$ for a constant $c > 0$. This implies that the approximation error between the solution operator and its best rank-k approximant decays only algebraically with k. Moreover, the decay rate deteriorates in high dimensions. In particular, the length p of the feature vector in DeepONets must be significantly large to approximate the solution operator to a prescribed accuracy.

However, Bebendorf and Hackbusch (2003, Thm. 2.8) showed that the Green's function G associated with Eq. (7) can be well approximated by a low-rank kernel when restricted to separated subdomains $D_X \times D_Y$ of $\Omega \times \Omega$, satisfying the strong admissibility condition: $\operatorname{dist}(D_X, D_Y) < \operatorname{diam}(D_Y)$. Here, the distance and diameter in $\mathbb{R}^d$ are defined as

$$\operatorname{dist}(D_X, D_Y) = \inf_{x \in D_X, y \in D_Y} |x - y|, \qquad \operatorname{diam}(D_Y) = \sup_{y_1, y_2 \in D_Y} |y_1 - y_2|.$$

Then, for any $\epsilon \in (0, 1)$, there exists a separable approximation of the form $G_k(x, y) = \sum_{i=1}^{k} u_i(x) v_i(y)$, with $k = \mathcal{O}(\log(1/\epsilon)^{d+1})$, such that

$$\|G - G_k\|_{L^2(D_X \times D_Y)} \leq \epsilon \|G\|_{L^2(D_X \times \hat{D}_Y)},$$

where $\hat{D}_Y$ is a domain slightly larger than D_Y (Bebendorf and Hackbusch, 2003, Thm. 2.8). This property has been exploited by Boullé and Townsend (2023); Boullé et al. (2023, 2022b) to derive sample complexity bounds for learning Green's functions associated with elliptic and parabolic PDEs. It motivates the decomposition of the Green kernel into a sum of low-rank kernels $G = K_1 + \ldots + K_L$ in MGNO architectures. Indeed, one can exploit the low-rank structure of Green's functions on well-separated domains to perform a hierarchical decomposition of the domain $\Omega \times \Omega$ into a tree of subdomains satisfying the admissibility condition. In Fig. 8, we illustrate the decomposition of the Green's function associated with the 1D Poisson equation on $\Omega = [0, 1]$ with homogeneous Dirichlet boundary conditions into a hierarchy of $L = 3$ levels of different range of interactions. The first level captures the long-range interactions, while the last level captures the short-range interactions. Then, the integral

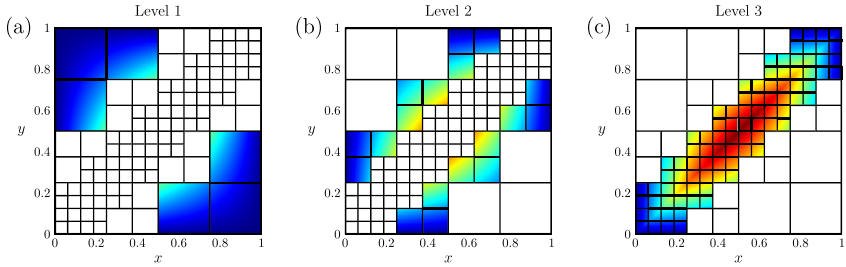

FIGURE 8 Decomposition of the Green's function associated with the 1D Poisson equation, displayed in Fig. 7(a), into a hierarchy of kernels capturing different ranges of interactions: from long-range (a) to short-range (c) interactions.

operation in Eq. (3) can be performed by aggregating the contributions of the subdomains in the tree, starting from the leaves and moving up to the root. This allows for the evaluation of the integral operation in Eq. (3) in linear complexity in the number of subdomains. The key advantage is that the approximation error on each subdomain decays exponentially fast as the rank of the approximating kernel increases.

One alternative approach to MGNO is to encode the different scales of the solution operators using a wavelet basis. This class of operator learning techniques (Feliu-Faba et al., 2020; Gupta et al., 2021; Tripura and Chakraborty, 2022) is based on the wavelet transform and aims to learn the solution operator kernel at multiple scale resolutions. One advantage over MGNO is that it does not require building a hierarchy of meshes, which could be computationally challenging in high dimensions or for complex domain geometries.

Finally, motivated by the success of the self-attention mechanism in transformers architectures for natural language processing (Vaswani et al., 2017) and image recognition (Dosovitskiy et al., 2020), several architectures have been proposed to learn global correlations in solution operators of PDEs. In particular, Cao (2021) introduced an architecture based on the self-attention mechanism for operator learning and observed higher performance on benchmark problems when compared against the Fourier Neural Operator. More recently, Kissas et al. (2022) propose a Kernel-Coupled Attention mechanism to learn correlations between the entries of a vector feature representation of the output functions. In contrast, Hao et al. (2023a) designed a general neural operator transformer (GNOT) that allows for multiple input functions and complex meshes.

4 Learning neural operators

In this section, we discuss various technical aspects involved in training neural operators, such as the data acquisition of forcing terms and solutions, the amount of training data required in practice, and the optimization algorithms and loss functions used to train neural operators.

4.1 Data acquisition

This section focuses on the data acquisition process for learning neural operators. In real-world applications, one may not have control over the distribution of source terms and solutions or locations of the sensors to measure the solutions at specific points in the domain. Therefore, we consider an idealized setting where one is interested in generating synthetic data using numerical PDE solvers to develop neural operator architectures. In this case, one has complete control over the distribution of source terms and solutions, as well as the locations of the sensors.

4.1.1 Distribution of source terms

The source terms $\{f_j\}_{j=1}^N$ used to generate pairs of training data to train neural operators are usually chosen to be random functions, sampled from a Gaussian random field (Lu et al., 2021a). Let $\Omega \subset \mathbb{R}^d$ be a domain, then a stochastic process $\{X_x,\, x \in \Omega\}$ indexed by Ω, is Gaussian if, for every finite set of indices $x_1, \ldots, x_n \in \Omega$, the vector of random variables $(X_{x_1}, \ldots, X_{x_n})$ follows a multivariate Gaussian distribution. The Gaussian process (GP) distribution is completely determined by the following mean and covariance functions (Adler, 2010, Sec. 1.6):

$$\mu(x) = \mathbb{E}\{X_x\}, \quad K(x, y) = \mathbb{E}\{[X_x - \mu(x)]^\top [X_y - \mu(y)]\}, \quad x \in \Omega.$$

In the rest of the paper, we will denote a Gaussian process with mean μ and covariance kernel K by $\mathcal{GP}(\mu, K)$. The mean function μ is usually chosen to be zero, while K is symmetric and positive-definite.

When K is continuous Mercer's theorem (Mercer, 1909) states that there exists an orthonormal basis of eigenfunctions $\{\psi_j\}_{j=1}^\infty$ of $L^2(\Omega)$, and nonnegative eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots > 0$ such that

$$K(x, y) = \sum_{j=1}^{\infty} \lambda_j \psi_j(x) \psi_j(y), \quad x, y \in \Omega,$$

where the sum is absolutely and uniformly convergent (Hsing and Eubank, 2015, Thm. 4.6.5). Here, the eigenvalues and eigenfunctions of the kernel are defined as solutions to the associated Fredholm integral equation:

$$\int_\Omega K(x, y) \psi_j(y)\, \mathrm{d}y = \lambda_j \psi_j(x), \quad x \in \Omega.$$

Then, the Karhunen–Loève theorem (Karhunen, 1946; Loève, 1946) ensures that a zero mean square-integrable Gaussian process X_x with continuous covariance function K admits the following representation:

$$X_x = \sum_{j=1}^{\infty} \sqrt{\lambda_j} c_j \psi_j(x), \quad c_j \sim \mathcal{N}(0, 1), \quad x \in \Omega, \tag{10}$$

where c_j are independent and identically distributed (i.i.d.) random variables, and the convergence is uniform in $x \in \Omega$. Suppose the eigenvalue decomposition of the covariance function is known. In that case, one can sample a random function from the associated GP, $\mathcal{GP}(0, K)$, by sampling the coefficients c_j in Eq. (10) from a standard Gaussian distribution and truncated the series up to the desired resolution. Under suitable conditions, one can relate the covariance function K's smoothness to the random functions sampled from the associated GP (Adler, 2010, Sec. 3). Moreover, the decay rate of the eigenvalues provides information about the smoothness of the underlying kernel (Ritter et al., 1995; Zhu et al., 1998). In practice, the number of eigenvalues greater than machine precision dictates the dimension of the finite-dimensional vector space spanned by the random functions sampled from $\mathcal{GP}(0, K)$.

One of the most common choices of covariance functions for neural operator learning include the squared-exponential kernel (Lu et al., 2021a; Boullé et al., 2022a), which is defined as

$$K(x, y) = \exp(-|x - y|^2/(2\ell^2)), \quad x, y \in \Omega,$$

where $\ell > 0$ is the length-scale parameter, which roughly characterizes the distance at which two point values of a sampled random function become uncorrelated (Rasmussen and Williams, 2006, Chapt. 5). Moreover, eigenvalues of the squared-exponential kernel decay exponentially fast at a rate that depends on the choice of ℓ (Zhu et al., 1998; Boullé and Townsend, 2022). After a random function f has been sampled from the GP, one typically discretizes it by performing a piecewise linear interpolation at sensor points $x_1, \ldots, x_m \in \Omega$ by evaluating f as these points. The interpolant can then solve the underlying PDE or train a neural operator. The number of sensors is chosen to resolve the underlying random functions and depends on their smoothness. Following the analysis by Lu et al. (2021a, Suppl. Inf. S4), in one dimension, the error between f and its piecewise linear interpolant is of order $\mathcal{O}(1/(m^2\ell^2))$, and one should choose $m \geq 1/\ell$. A typical value of ℓ lies in the range $\ell \in [0.01, 0.1]$ with $m = 100$ sensors (Lu et al., 2021a; Boullé et al., 2022a). We illustrate the eigenvalues of the squared-exponential kernel on $\Omega = [0, 1]$ with length-scale parameters $\ell \in \{0.1, 0.05, 0.01\}$, along with the corresponding random functions sampled from the associated GP in Fig. 9. As the length-scale parameter ℓ decreases, the eigenvalues decay faster, and the sampled random functions become smoother.

Another possible choice of covariance functions for neural operator learning (Benitez et al., 2023; Zhu et al., 2023) comes from the Matérn class of covariance functions (Rasmussen and Williams, 2006; Stein, 1999):

$$K(x, y) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}|x - y|}{\ell} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}|x - y|}{\ell} \right), \quad x, y \in \Omega,$$

where Γ is the Gamma function, K_ν is a modified Bessel function, and ν, ℓ are positive parameters that enable the control of the smoothness of the sampled

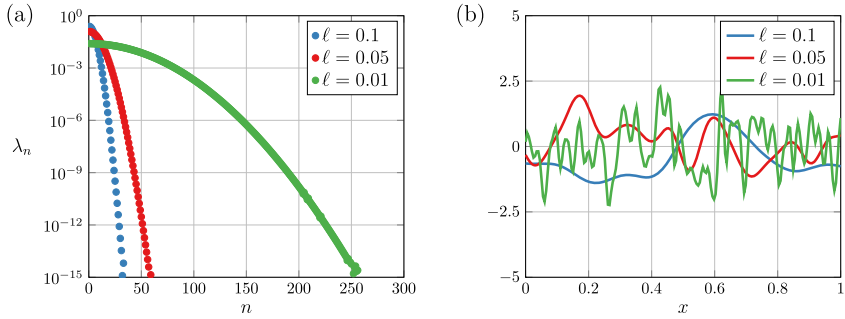

FIGURE 9 (a) Eigenvalues of the squared-exponential kernel on $\Omega = [0, 1]$ with length-scale parameters $\ell \in \{0.1, 0.05, 0.01\}$. (b) Random functions sampled from the associated Gaussian process $\mathcal{GP}(0, K)$, where K is the squared-exponential kernel with length-scale parameters $\ell \in \{0.1, 0.05, 0.01\}$.

random functions. Hence, the resulting Gaussian process is $\lceil \nu \rceil - 1$ times mean-squared differentiable (Rasmussen and Williams, 2006, Sec. 4.2). Moreover, the Matérn kernel converges to the squared-exponential covariance function as $\nu \to \infty$. We refer to the book by Rasmussen and Williams (2006) for a detailed analysis of other standard covariance functions in Gaussian processes.

Finally, Li et al. (2021a); Kovachki et al. (2023) propose to use a Green kernel associated with a differential operator, which is a power of the Helmholtz equation, as a covariance function:

$$K = A(-\nabla^2 + cI)^{-\nu}.$$

Here, A, $\nu > 0$, and $c \geq 0$ are parameters that respectively govern the scaling of the Gaussian process, the algebraic decay rate of the spectrum, and the frequency of the first eigenfunctions of the covariance function. One motivation for this choice of distribution is that it allows the enforcement of prior information about the underlying model, such as the order of the differential operator, directly into the source terms. A similar behavior has been observed in a randomized linear algebra context when selecting the distribution of random vectors for approximating matrices from matrix-vector products using the randomized SVD (Boullé and Townsend, 2022). For example, the eigenvalues associated with this covariance kernel decay at an algebraic rate, implying that random functions sampled from this GP would be more oscillatory. This could lead to higher performance of the neural operator on high-frequency source terms. However, one downside of this approach is that a poor choice of the parameters can affect the training and approximation error of the neural operator. Hence, the studies (Li et al., 2021a; Kovachki et al., 2023) employ different parameter choices in each of the reported numerical experiments, suggesting that the covariance hyperparameters have been heavily optimized.

4.1.2 Numerical PDE solvers

After choosing the covariance function and generating the random source terms from the Gaussian process, one must solve the underlying PDE to generate the corresponding solutions. In general, the PDE is unknown, and one only has access to an oracle (such as physical experiments or black-box numerical solver) that outputs solutions u to the PDE from input source terms f as $\mathcal{L}u = f$. However, generating synthetic data from known mathematical models to design or evaluate neural operator architectures is often convenient. One can use numerical PDE solvers to generate the corresponding solutions in this case. This section briefly describes the different numerical methods that can be used to solve the underlying PDEs, along with their key attributes summarized in Table 3. We want to emphasize that these methods have many variations, and we refer to the most standard ones.

TABLE 3 Summary of the different properties of standard finite difference, finite element, and spectral methods for solving PDEs.

Property	Finite differences	Finite elements	Spectral methods
Domain geometry	Simple	Complex	Simple
Approximation	Local	Local	Global
Linear system	Large sparse	Large sparse	Small dense
Convergence rate	Algebraic	Algebraic	Spectral

When the PDE does not depend on time, the most common techniques for discretizing and solving it are finite difference methods (FDM), finite element methods (FEM), and spectral methods. The finite difference method consists of discretizing the computational domain Ω into a grid and approximating spatial derivatives of the solution u from linear combinations of the values of u at the grid points using finite difference operators (Iserles, 2009, Chap. 8). This approach is based on a local Taylor expansion of the solution and is very easy to implement on rectangular geometries. However, it usually requires a uniform grid approximation of the domain, which might not be appropriate for complex geometries and boundary conditions. The finite element method (Süli and Mayers, 2003, Chap. 14) employs a different approach than FDM and considers the approximation of the solution u on a finite-dimensional vector space spanned by basis functions with local support on Ω. The spatial discretization of the domain Ω is performed via a mesh representation. The basis functions are often chosen as piecewise polynomials supported on a set of elements, which are adjacent cells in the mesh. This approach is more flexible than FDM and can be used to solve PDEs on complex geometries and boundary conditions. However, it is more challenging to implement and requires the construction of a mesh of the domain Ω, which can be computationally expensive. We highlight that the finite difference and finite element methods lead to large, sparse, and highly structured linear algebra systems, which can be solved efficiently using iterative methods.

Two commonly used finite element software for generating training data for neural operators are FEniCS (Alnæs et al., 2015) and Firedrake (Rathgeber et al., 2016; Ham et al., 2023). These open-source software exploit the Unified Form Language (UFL) developed by Alnæs et al. (2014) to define the weak form of the PDE in a similar manner as in mathematics and automatically generate the corresponding finite element assembly code before exploiting fast linear and nonlinear solvers through a Python interface with the high-performance PETSc library (Balay et al., 2023).

Finally, spectral methods (Iserles, 2009; Gottlieb and Orszag, 1977; Trefethen, 2000) are based on the approximation of the solution u on a finite-dimensional vector space spanned by basis functions with global support on Ω, usually Chebyshev polynomials or trigonometric functions. Spectral methods are motivated by the fact that the solution u of a PDE defined on a 1D interval is often smooth if the source term is smooth and so can be well-approximated by a Fourier series if it is periodic or a Chebyshev series if it is not periodic. Hence, spectral methods lead to exponential convergence, also called spectral accuracy, to analytic solutions with respect to the number of basis functions, unlike FDM and FEM, which only converge at an algebraic rate. The fast convergence rate of spectral methods implies that a small number of basis functions is usually required to achieve a given accuracy. Therefore, the matrices associated with the resulting linear algebra systems are much smaller than for FEM but are dense. In summary, spectral methods have a competitive advantage over FDM and FEM on simple geometries, such as tensor-product domains, and when the solution is smooth. At the same time, FEM might be difficult to implement but is more flexible. A convenient software for solving simple PDEs using spectral methods is the Chebfun software system (Driscoll et al., 2014), an open-source package written in MATLAB®.

For time-dependent PDEs, one typically starts by performing a time-discretization using a time-stepping scheme, such as backward differentiation schemes (e.g. backward Euler) and Runge–Kutta methods (Iserles, 2009; Süli and Mayers, 2003), and then employ a spatial discretization method, such as the techniques described before in this section, to solve the resulting stationary PDE at each time-step.

4.1.3 Amount of training data

Current neural operator approaches typically require a relatively small amount of training data, in the order of a thousand input-output pairs, to approximate solution operators associated with PDEs (Lu et al., 2021a; Goswami et al., 2023; Kovachki et al., 2023; Boullé et al., 2023). This contrasts with the vast amount of data used to train neural networks for standard supervised learning tasks, such as image classification, which could require hundreds of millions of labeled samples (LeCun et al., 2015). This difference can be explained by the fact that solution operators are often highly structured, which can be exploited to design data-efficient neural operator architectures (see Section 3).

Recent numerical experiments have shown that the rate of convergence of neural operators with respect to the number of training samples evolves in two regimes (Lu et al., 2021a, Fig. S10). In the first regime, we observe a fast decay of the testing error at an exponential rate (Boullé et al., 2023). Then, the testing error decays at a slower algebraic rate in the second regime for a larger amount of samples and saturates due to discretization error and optimization issues, such as convergence to a suboptimal local minimum.

On the theoretical side, several works derived sample complexity bounds that characterize the amount of training data required to learn solution operators associated with certain classes of linear PDEs to within a prescribed accuracy $0 < \epsilon < 1$. In particular, Boullé et al. (2023); Schäfer and Owhadi (2021) focus on approximating Green's functions associated with uniformly elliptic PDEs in divergence form:

$$-\operatorname{div}(A(x)\nabla u) = f, \quad x \in \Omega \subset \mathbb{R}^d, \tag{11}$$

where $A(x)$ is a symmetric bounded coefficient matrix (see Eq. (7)). These studies construct data-efficient algorithms that converge exponentially fast with respect to the number of training pairs. Hence, they can recover the Green's function associated with Eq. (11) to within ϵ using only $\mathcal{O}(\text{polylog}(1/\epsilon))$ sample pairs. The method employed by Boullé et al. (2023) consists of recovering the hierarchical low-rank structure satisfied by Green's function on well-separated subdomains (Bebendorf and Hackbusch, 2003; Lin et al., 2011; Levitt and Martinsson, 2022b) using a generalization of the rSVD to Hilbert–Schmidt operators (Boullé and Townsend, 2022, 2023; Halko et al., 2011; Martinsson and Tropp, 2020). Interestingly, the approach by Schäfer and Owhadi (2021) is not based on low-rank techniques but relies on the sparse Cholesky factorization of elliptic solution operators (Schäfer et al., 2021).

Some of the low-rank recovery techniques employed by Boullé et al. (2023) extend naturally to time-dependent parabolic PDEs (Boullé et al., 2022b) in spatial dimension $d \geq 1$ of the form:

$$\frac{\partial u}{\partial t} - \operatorname{div}(A(x,t)\nabla u) = f, \quad x \in \Omega \subset \mathbb{R}^d, \quad t \in (0,T], \tag{12}$$

where the coefficient matrix $A(x,t) \in \mathbb{R}^{d\times d}$ is symmetric positive definite with bounded coefficient functions in $L^\infty(\Omega \times [0,T])$, for some $0 < T < \infty$, and satisfies the uniform parabolicity condition (Evans, 2010, Sec. 7.1.1). Parabolic systems in the form of Eq. (12) model various time-dependent phenomena, including heat conduction and particle diffusion. Boullé et al. (2022b, Thm. 9) showed that the Green's function associated with Eq. (12) admits a hierarchical low-rank structure on well-separated subdomains, similarly to the elliptic case (Bebendorf and Hackbusch, 2003). Combining this with the pointwise bounds satisfied by the Green's function (Cho et al., 2012), one can construct an algorithm that recovers the Green's function to within ϵ using $\mathcal{O}(\text{poly}(1/\epsilon))$ sample pairs (Boullé et al., 2022b, Thm. 10).

Finally, other approaches (de Hoop et al., 2023; Jin et al., 2023; Stepaniants, 2023) derived convergence rates for a broader class of operators between infinite-dimensional Hilbert spaces, which are not necessarily associated with solution operators of PDEs. In particular, de Hoop et al. (2023) consider the problem of estimating the eigenvalues of an unknown, and possibly unbounded, self-adjoint operator assuming that the operator is diagonalizable in a known basis of eigenfunctions, and highlight the impact of varying the smoothness of training and test data on the convergence rates. Then, Stepaniants (2023); Jin et al. (2023) derive upper and lower bounds on the sample complexity of Hilbert–Schmidt operators between two reproducing kernel Hilbert spaces (RKHS) that depend on the smoothness of the input and output functions.

4.2 Optimization

Once the neural operator architecture has been selected and the training dataset is constituted, the next task is to train the neural operator by solving an optimization problem in the form of Eq. (1). The aim is to identify the best parameters of the underlying neural network so that the output $\hat{A}(f;\theta)$ of the neural operator evaluated at a forcing term f in the training dataset fits the corresponding ground truth solution u. This section describes the most common choices of loss functions and optimization algorithms employed in current operator learning approaches. Later in Section 4.2.3, we briefly discuss how to measure the convergence and performance of a trained neural operator.

4.2.1 Loss functions

The choice of loss function in operator learning is a critical step, as it directs the optimization process and ultimately affects the model's performance. Different types of loss functions can be utilized depending on the task's nature, the operator's structure, and the function space's properties. A common choice of loss function in ML is the mean squared error (MSE), which is defined as

$$L_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{m}\sum_{j=1}^{m}|\hat{A}(f_i)(x_j) - u_i(x_j)|^2 \approx \frac{1}{N}\sum_{i=1}^{N}\|\hat{A}(f_i) - u_i\|^2_{L^2(\Omega)}, \tag{13}$$

and is employed in the original DeepONet study (Lu et al., 2021a). Here, N is the number of training samples, m is the number of sensors, f_i is the i-th forcing term, u_i is the corresponding ground truth solution, $\hat{A}(f_i)$ is the output of the neural operator evaluated at f_i, and x_j is the j-th sensor location. This loss function discretizes the squared L^2 error between the output of the neural operator and the ground truth solution at the sensor locations. When the sensor grid is regular, one can employ a higher-order quadrature rule to discretize the L^2 norm. Moreover, in most cases, it may be beneficial to use a relative error, especially when the magnitudes of the output function can vary widely. Then,

Boullé et al. (2022a) use the following relative squared L^2 loss function

$$L = \frac{1}{N}\sum_{i=1}^{N} \frac{\|\hat{A}(f_i) - u_i\|^2_{L^2(\Omega)}}{\|u_i\|^2_{L^2(\Omega)}}, \tag{14}$$

which is discretized using a trapezoidal rule. The most common loss function in operator learning is the relative L^2 error employed in Fourier neural operator techniques (Li et al., 2021a):

$$L_2 = \frac{1}{N}\sum_{i=1}^{N} \frac{\|\hat{A}(f_i) - u_i\|_{L^2(\Omega)}}{\|u_i\|_{L^2(\Omega)}}. \tag{15}$$

Kovachki et al. (2023) observed a better normalization of the model when using a relative loss function, and that the choice of Eq. (15) decreases the testing error by a factor of two compared to Eq. (14).

For tasks that require robustness to outliers or when it is important to measure the absolute deviation, the L^1 loss can be employed. It is defined as

$$L_1 = \frac{1}{N}\sum_{i=1}^{N} \frac{\|\hat{A}(f_i) - u_i\|_{L^1(\Omega)}}{\|u_i\|_{L^1(\Omega)}}. \tag{16}$$

This loss function tends to be less sensitive to large deviations than the L^2 loss (Alpak et al., 2023; Lyu et al., 2023; Zhao et al., 2024). Furthermore, Sobolev norms can also be used as a loss function when the unknown operator $\mathcal{A}$ involves functions in Sobolev spaces (Evans, 2010, Chapt. 5), particularly when the derivatives of the input and output functions play a role (Son et al., 2021; Yu et al., 2023; O'Leary-Roseberry et al., 2024). For example, one could perform training with a relative H^1 loss to enforce the smoothness of the neural operator output. Finally, when the underlying PDE is known, one can enforce it as a weak constraint when training the neural operator by adding a PDE residual term to the loss function (Li et al., 2021b; Wang et al., 2021b), similarly to physics-informed neural networks (Raissi et al., 2019).

4.2.2 Optimization algorithms and implementation

The training procedure of neural operators is typically performed using Adam optimization algorithm (Kingma and Ba, 2015; Kovachki et al., 2023; Lu et al., 2021a; Li et al., 2021a; Goswami et al., 2023) or one of its variants such as AdamW (Loshchilov and Hutter, 2019; Hao et al., 2023a). Hence, the work introducing DeepONets by Lu et al. (2021a) employed Adam algorithm to train the neural network architecture with a default learning rate of 0.001. In contrast, Kovachki et al. (2023) incorporate learning rate decay throughout the optimization of Fourier neural operators. One can also employ a two-step training approach by minimizing the loss function using Adam algorithm for a fixed

number of iterations and then fine-tuning the neural operator using the L-BFGS algorithm (Byrd et al., 1995; Cuomo et al., 2022; He et al., 2020; Mao et al., 2020; Boullé et al., 2022a). This approach has been shown to improve the convergence rate of the optimization when little data is available in PINNs applications (He et al., 2020). Popular libraries for implementing and training neural operators include PyTorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2016).

Thus far, there has been limited focus on the theoretical understanding of convergence and optimization of neural operators. Since neural operators are a generalization of neural networks in infinite dimensions, existing convergence results of physics-informed neural networks (Wang et al., 2021a, 2022b) based on the neural tangent kernel (NTK) framework (Jacot et al., 2018; Du et al., 2019; Allen-Zhu et al., 2019) should naturally extend to neural operators. One notable exception is the study by Wang et al. (2022a), which analyzes the training of physics-informed DeepONets (Wang et al., 2021b) and derives a weighting scheme guided by NTK theory to balance the data and the PDE residual terms in the loss function.

4.2.3 *Measuring convergence and superresolution*

After training a neural operator, one typically measures its performance by evaluating the testing error, such as the relative L^2-error, on a set of unseen data generated using the procedure described in Section 4. In general, state-of-the-art neural operator architectures report a relative testing error of $1\% - 10\%$ depending on the problems considered (Kovachki et al., 2023; Lu et al., 2021a; Li et al., 2021a).

However, it is essential to note that the testing error may not be a good measure of the performance of a neural operator, as it does not provide any information about the generalization properties of the model. Hence, the testing forcing terms are usually sampled from the same distribution as the training forcing terms, so they lie on the same finite-dimensional function space, determined by the spectral decay of the GP covariance kernel eigenvalues (see Section 4.1). Moreover, in real applications, the testing source terms could have different distributions than the ones used for training ones, and extrapolation of the neural operator may be required. Zhu et al. (2023) investigate the extrapolation properties of DeepONet with respect to the length-scale parameter of the underlying source term GP. They observe that the testing error increases when the length-scale parameter corresponding to the test data decreases. At the same time, the neural operator can extrapolate to unseen data with a larger length scale than the training dataset, i.e., smoother functions.

One attractive property of neural operators is their resolution invariance to perform predictions at finer spatial resolutions than the training dataset on which they have been trained. This is usually called zero-shot superresolution (Kovachki et al., 2023, Sec. 7.2.3). To investigate this property, we reproduce the numerical examples of Kovachki et al. (2023, Sec. 7.2) and train a Fourier neu-

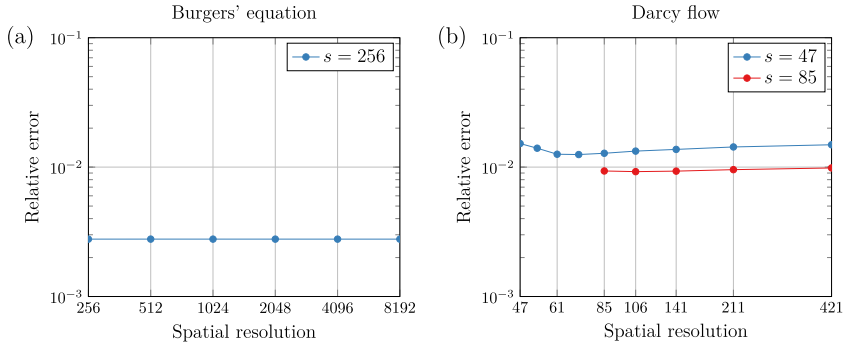

FIGURE 10 (a) Relative test error at different spatial resolutions of the Fourier neural operator trained to approximate the solution operator of the 1D Burgers' equation (17) with trained resolution of $s = 256$. (b) FNO trained on 2D Darcy flow a resolution of $s = 47$ (blue) and $s = 85$ (red) and evaluated at higher spatial resolutions.

ral operator to approximate the solution operator of Burgers' equation and Darcy flow at a low-resolution data and evaluate the operator at higher resolutions. We consider the one-dimensional Burgers' equation:

$$\frac{\partial}{\partial t}u(x,t) + \frac{1}{2}\frac{\partial}{\partial_x}(u(x,t)^2) = \nu\frac{\partial^2}{\partial_{x^2}}u(x,t), \quad x \in (0,2\pi), \quad t \in [0,1], \tag{17}$$

with periodic boundary conditions and viscosity $\nu = 0.1$. We are interested in learning the solution operator $\mathcal{A} : L^2_{\text{per}}((0,2\pi)) \to H^1_{\text{per}}((0,2\pi))$, which maps initial conditions $u_0 \in L^2_{\text{per}}((0,2\pi))$ to corresponding solutions $u(\cdot,1) \in H^1_{\text{per}}((0,2\pi))$ to Eq. (17) at time $t = 1$. We then discretize the source and solution training data on a uniform grid with spatial resolution $s = 256$ and evaluate the trained neural operator at finer spatial resolutions $s \in \{512, 1024, 2048\}$. We observe in Fig. 10(a) that the relative testing error of the neural operator is independent of the spatial resolutions, as reported by Kovachki et al. (2023, Sec. 7.2).

Next, we consider the two-dimensional Darcy flow equation (18) with constant source term $f = 1$ and homogeneous Dirichlet boundary conditions on a unit square domain $\Omega = [0,1]^2$:

$$-\operatorname{div}(a(x)\nabla u) = 1, \quad x \in [0,1]^2. \tag{18}$$

We train a Fourier neural operator to approximate the solution operator, mapping the coefficient function a to the associated solution u to Eq. (18). We reproduce the numerical experiment in Kovachki et al. (2023, Sec. 6.2), where the random coefficient functions a are piecewise constant. The random functions a are generated as $a \sim T \circ f$, where $f \sim \mathcal{GP}(0, C)$, with $C = (-\Delta + 9I)^{-2}$ and

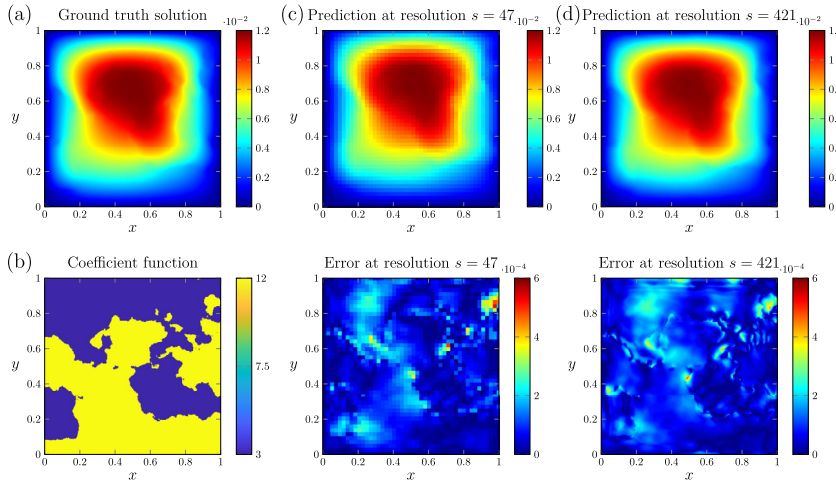

FIGURE 11 (a) Ground truth solution u to the 2D Darcy flow equation (18) corresponding to the coefficients function plotted in (b). (c)-(d) Predicted solution and approximation error at $s = 47$ and $s = 421$ by a Fourier neural operator trained on a Darcy flow dataset with spatial resolution of $s = 47$.

$T : \mathbb{R} \to \mathbb{R}^+$ is defined as

$$T(x) = \begin{cases} 12, & \text{if } x \geq 0, \\ 3, & \text{if } x < 0. \end{cases}$$

We discretize the coefficient and solution training data on a $s \times s$ uniform grid with spatial resolution $s = 47$ and evaluate the trained neural operator at higher spatial resolutions in Fig. 10. The relative testing error does not increase as the spatial resolution increases. Moreover, training the neural operator on a higher spatial resolution dataset can decrease the testing error. We also plot the ground truth solution u to Eq. (18) in Fig. 11(a) corresponding to the coefficient function plotted in panel (b), along with the predicted solutions and approximation errors at $s = 47$ and $s = 421$ by the Fourier neural operator in panels (c) and (d). We want to point the reader interested in the discretization properties of neural operators to the recent perspective on representation equivalent neural operators (ReNO) by Bartolucci et al. (2023).

5 Conclusions and future challenges

In this paper, we provided a comprehensive overview of the recent developments in neural operator learning, a new paradigm at the intersection of scientific computing and ML for learning solution operators of PDEs. Given the recent surge of interest in this field, a key question concerns the choice of neural architectures for different PDEs. Most theoretical studies in the field analyze and compare neural operators through the prism of approximation theory. We proposed a

framework based on numerical linear algebra and matrix recovery problems for interpreting the type of neural operator architectures that can be used to learn solution operators of PDEs. Hence, solution operators associated with linear PDEs can often be written as integral operators with a Green's function and recover by a one-layer neural operator, which after discretization is equivalent to a matrix recovery problem.

Moreover, the choice of architectures, such as FNO or DeepONet, enforces or preserves different properties of the PDE solution operator, such as being translation invariant, low-rank, or off-diagonal low-rank (see Table 2). We then focused on the data acquisition process. We highlighted the importance of the distribution of source terms, usually sampled from a Gaussian process with a tailored covariance kernel, on the resulting performance of the neural operator. Following recent works on elliptic and parabolic PDEs and numerical experiments, we also discussed the relatively small amount of training data needed for operator learning. Finally, we studied the different choices of optimization algorithms and loss functions and highlighted the superresolution properties of neural operators, i.e., their ability to be evaluated at higher resolution than the training dataset with a minor impact on the performance. There are, however, several remaining challenges in the field.

Distribution of probes

Most applications of neural operators employ source terms that are globally supported on the domain, sampled from a Gaussian process, and whose distribution is fixed before training. However, this might now always apply to real-world engineering or biological systems, where source terms could be localized in space and time. A significant problem is to study the impact of the distribution of locally supported source terms on the performance of neural operators, both from a practical and theoretical viewpoint. Hence, recent sample complexity works on elliptic and parabolic PDEs exploit structured source terms (Boullé et al., 2023, 2022b; Schäfer and Owhadi, 2021). Another area of future research is to employ adaptive source terms to fine-tune neural operators for specific applications. This could lead to higher performance by selecting source terms that maximize the training error or allow efficient transfer learning between different applications without retraining a large neural operator.

Software and datasets

An essential step towards democratizing operator learning involves the development of open-source software and datasets for training and comparing neural operators, similar to the role played by the MNIST (LeCun et al., 1998) and ImageNet (Deng et al., 2009) databases in the improvement of computer vision techniques. However, due to the fast emerging methods in operator learning, there have been limited attempts beyond (Lu et al., 2022) to standardize the datasets and software used in the field. Establishing a list of standard PDE problems across different scientific fields, such as fluid dynamics, quantum mechan-

ics, and epidemiology, with other properties (e.g. linear/nonlinear, steady/time-dependent, low/high dimensional, smooth/rough solutions, simple/complex geometry) would allow researchers to compare and identify the neural operator architectures that are the most appropriate for a particular task. A recent benchmark has been proposed to evaluate the performance of physics-informed neural networks for solving PDEs (Hao et al., 2023b).

Real-world applications

Neural operators have been successfully applied to perform weather forecasting and achieve spectacular performance in terms of accuracy and computational time to solutions compared to traditional numerical weather prediction techniques while being trained on historical weather data (Kurth et al., 2023; Lam et al., 2023). An exciting development in the field of operator learning would be to expand the scope of applications to other scientific fields and train the models on real datasets, where the underlying PDE governing the data is unknown to discover new physics.

Theoretical understanding

Following the recent works on the approximation theory of neural operators and sample complexity bounds for different classes of PDEs, there is a growing need for a theoretical understanding of convergence and optimization. In particular, an exciting area of research would be to extend the convergence results of physics-informed neural networks and the neural tangent kernel framework to neural operators. This would enable the derivation of rigorous convergence rates for different types of neural operator architectures and loss functions and new schemes for initializing the weight distributions in the underlying neural networks.

Physical properties

Most neural operator architectures are motivated by obtaining a good approximation of the solution operator of a PDE. However, the resulting neural operator is often highly nonlinear, difficult to interpret mathematically, and might not satisfy the physical properties of the underlying PDE, such as conservation laws or symmetries (Olver, 1993b). There are several promising research directions in operator learning related to symmetries and conservation laws (Otto et al., 2023). One approach would be to enforce known physical properties when training neural operators, either strongly through structure preserving architectures (Richter-Powell et al., 2022), or weakly by adding a residual term in the loss function (Li et al., 2021b; Wang et al., 2021b). Another direction is to discover new physical properties of the underlying PDEs from the trained neural operator. While (Boullé et al., 2022a) showed that symmetries of linear PDEs can be recovered from the learned Green's function, this approach has not been extended to nonlinear PDEs. Finally, one could also consider using

reinforcement learning techniques for enforcing physical constraints after the optimization procedure, similar to recent applications in large language models (Ouyang et al., 2022).

Acknowledgments

The work of both authors was supported by the Office of Naval Research (ONR), under grant N00014-23-1-2729. N.B. was supported by an INI-Simons Postdoctoral Research Fellowship. A.T. was supported by National Science Foundation grants DMS-2045646 and a Weiss Junior Fellowship Award.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al., 2016. Tensorflow: a system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation, pp. 265–283.

Adler, R.J., 2010. The Geometry of Random Fields. SIAM.

Allen-Zhu, Z., Li, Y., Song, Z., 2019. A convergence theory for deep learning via over-parameterization. In: International Conference on Machine Learning, pp. 242–252.

Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N., 2015. The FEniCS project version 1.5. Arch. Numer. Softw. 3 (100).

Alnæs, M.S., Logg, A., Ølgaard, K.B., Rognes, M.E., Wells, G.N., 2014. Unified form language: a domain-specific language for weak formulations of partial differential equations. ACM Trans. Math. Softw. 40 (2), 1–37.

Alpak, F.O., Vamaraju, J., Jennings, J.W., Pawar, S., Devarakota, P., Hohl, D., 2023. Augmenting deep residual surrogates with Fourier neural operators for rapid two-phase flow and transport simulations. SPE J., 1–22.

Arridge, S., Maass, P., Öktem, O., Schönlieb, C.-B., 2019. Solving inverse problems using data-driven models. Acta Numer. 28, 1–174.

Balay, S., Abhyankar, S., Adams, M.F., et al., 2023. PETSc Users Manual. Argonne National Laboratory.

Bartolucci, F., de Bézenac, E., Raonić, B., Molinaro, R., Mishra, S., Alaifari, R., 2023. Are neural operators really neural operators? Frame theory meets operator learning. arXiv preprint. arXiv: 2305.19913.

Bebendorf, M., Hackbusch, W., 2003. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. Numer. Math. 95 (1), 1–28.

Benitez, J.A.L., Furuya, T., Faucher, F., Kratsios, A., Tricoche, X., de Hoop, M.V., 2023. Out-of-distributional risk bounds for neural operators with applications to the Helmholtz equation. arXiv preprint. arXiv:2301.11509.

Bhattacharya, K., Hosseini, B., Kovachki, N.B., Stuart, A.M., 2021. Model reduction and neural networks for parametric PDEs. SMAI J. Comput. Math. 7, 121–157.

Boullé, N., Townsend, A., 2022. A generalization of the randomized singular value decomposition. In: International Conference on Learning Representations.

Boullé, N., Townsend, A., 2023. Learning elliptic partial differential equations with randomized linear algebra. Found. Comput. Math. 23 (2), 709–739.

Boullé, N., Nakatsukasa, Y., Townsend, A., 2020. Rational neural networks. In: Advances in Neural Information Processing Systems, vol. 33, pp. 14243–14253.

Boullé, N., Earls, C.J., Townsend, A., 2022a. Data-driven discovery of Green's functions with human-understandable deep learning. Sci. Rep. 12 (1), 4824.

Boullé, N., Kim, S., Shi, T., Townsend, A., 2022b. Learning Green's functions associated with time-dependent partial differential equations. J. Mach. Learn. Res. 23 (218), 1–34.

Boullé, N., Halikias, D., Townsend, A., 2023. Elliptic PDE learning is provably data-efficient. Proc. Natl. Acad. Sci. USA 120 (39), e2303904120.

Bronstein, M.M., Bruna, J., Cohen, T., Veličković, P., 2021. Geometric deep learning: grids, groups, graphs, geodesics, and gauges. arXiv preprint. arXiv:2104.13478.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al., 2020. Language models are few-shot learners. In: Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901.

Bruno, O.P., Han, Y., Pohlman, M.M., 2007. Accurate, high-order representation of complex three-dimensional surfaces via Fourier continuation analysis. J. Comput. Phys. 227 (2), 1094–1125.

Brunton, S.L., Proctor, J.L., Kutz, J.N., 2016. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc. Natl. Acad. Sci. USA 113 (15), 3932–3937.

Byrd, R.H., Lu, P., Nocedal, J., Zhu, C., 1995. A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. 16 (5), 1190–1208.

Canzani, Y., 2013. Analysis on Manifolds via the Laplacian. Harvard University.

Cao, S., 2021. Choose a transformer: Fourier or Galerkin. In: Advances in Neural Information Processing Systems, vol. 34, pp. 24924–24940.

Champion, K., Lusch, B., Kutz, J.N., Brunton, S.L., 2019. Data-driven discovery of coordinates and governing equations. Proc. Natl. Acad. Sci. USA 116 (45), 22445–22451.

Chen, T., Chen, H., 1995. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Trans. Neural Netw. 6 (4), 911–917.

Cho, S., Dong, H., Kim, S., 2008. On the Green's matrices of strongly parabolic systems of second order. Indiana Univ. Math. J. 57 (4), 1633–1677.

Cho, S., Dong, H., Kim, S., 2012. Global estimates for Green's matrix of second order parabolic systems with application to elliptic systems in two dimensional domains. Potential Anal. 36 (2), 339–372.

Cooley, J.W., Tukey, J.W., 1965. An algorithm for the machine calculation of complex Fourier series. Math. Comput. 19 (90), 297–301.

Cuomo, S., Di Cola, V.S., Giampaolo, F., Rozza, G., Raissi, M., Piccialli, F., 2022. Scientific machine learning through physics–informed neural networks: where we are and what's next. J. Sci. Comput. 92 (3), 88.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. 2 (4), 303–314.

de Hoop, M.V., Huang, D.Z., Qian, E., Stuart, A.M., 2022. The cost-accuracy trade-off in operator learning with neural networks. arXiv preprint. arXiv:2203.13181.

de Hoop, M.V., Kovachki, N.B., Nelsen, N.H., Stuart, A.M., 2023. Convergence rates for learning linear operators from noisy data. SIAM/ASA J. Uncertain. Quantificat. 11 (2), 480–513.

De Ryck, T., Mishra, S., 2022. Generic bounds on the approximation error for physics-informed (and) operator learning. In: Advances in Neural Information Processing Systems, vol. 35, pp. 10945–10958.

Deng, B., Shin, Y., Lu, L., Zhang, Z., Karniadakis, G.E., 2022. Approximation rates of DeepONets for learning operators arising from advection–diffusion equations. Neural Netw. 153, 411–426.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L., 2009. Imagenet: a large-scale hierarchical image database. In: Conference on Computer Vision and Pattern Recognition. IEEE, pp. 248–255.

DeVore, R.A., 1998. Nonlinear approximation. Acta Numer. 7, 51–150.

Di Leoni, P.C., Lu, L., Meneveau, C., Karniadakis, G.E., Zaki, T.A., 2023. Neural operator prediction of linear instability waves in high-speed boundary layers. J. Comput. Phys. 474, 111793.

Dong, H., Kim, S., 2009. Green's matrices of second order elliptic systems with measurable coefficients in two dimensional domains. Trans. Am. Math. Soc. 361 (6), 3303–3323.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al., 2020. An image is worth 16x16 words: transformers for image recognition at scale. arXiv preprint. arXiv:2010.11929.

Driscoll, T.A., Hale, N., Trefethen, L.N., 2014. Chebfun Guide. Pafnuty Publications. http://www.chebfun.org/docs/guide/.
Du, S., Lee, J., Li, H., Wang, L., Zhai, X., 2019. Gradient descent finds global minima of deep neural networks. In: International Conference on Machine Learning, pp. 1675–1685.
E, W., Yu, B., 2018. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. Commun. Math. Stat. 6 (1), 1–12.
Evans, L.C., 2010. Partial Differential Equations, 2nd edition. American Mathematical Society.
Fanaskov, V., Oseledets, I., 2022. Spectral neural operators. arXiv preprint. arXiv:2205.10573.
Feliu-Faba, J., Fan, Y., Ying, L., 2020. Meta-learning pseudo-differential operators with deep neural networks. J. Comput. Phys. 408, 109309.
Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E., 2017. Neural message passing for quantum chemistry. In: International Conference on Machine Learning, pp. 1263–1272.
Gin, C.R., Shea, D.E., Brunton, S.L., Kutz, J.N., 2021. DeepGreen: deep learning of Green's functions for nonlinear boundary value problems. Sci. Rep. 11 (1), 1–14.
Goswami, S., Yin, M., Yu, Y., Karniadakis, G.E., 2022. A physics-informed variational deeponet for predicting crack path in quasi-brittle materials. Comput. Methods Appl. Mech. Eng. 391, 114587.
Goswami, S., Bora, A., Yu, Y., Karniadakis, G.E., 2023. Physics-informed deep neural operator networks. In: Machine Learning in Modeling and Simulation: Methods and Applications. Springer, pp. 219–254.
Gottlieb, D., Orszag, S.A., 1977. Numerical Analysis of Spectral Methods: Theory and Applications. SIAM.
Greengard, L., Rokhlin, V., 1997. A new version of the fast multipole method for the Laplace equation in three dimensions. Acta Numer. 6, 229–269.
Grüter, M., Widman, K.-O., 1982. The Green function for uniformly elliptic equations. Manuscr. Math. 37 (3), 303–342.
Gupta, G., Xiao, X., Bogdan, P., 2021. Multiwavelet-based operator learning for differential equations. In: Advances in Neural Information Processing Systems, vol. 34, pp. 24048–24062.
Hackbusch, W., Khoromskij, B.N., Kriemann, R., 2004. Hierarchical matrices based on a weak admissibility criterion. Computing 73 (3), 207–243.
Halikias, D., Townsend, A., 2023. Structured matrix recovery from matrix-vector products. Numer. Linear Algebra Appl., e2531.
Halko, N., Martinsson, P.-G., Tropp, J.A., 2011. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. 53 (2), 217–288.
Ham, D.A., Kelly, P.H.J., Mitchell, L., Cotter, C.J., Kirby, R.C., Sagiyama, K., Bouziani, N., Vorderwuelbecke, S., Gregory, T.J., Betteridge, J., Shapero, D.R., Nixon-Hill, R.W., Ward, C.J., Farrell, P.E., Brubeck, P.D., Marsden, I., Gibson, T.H., Homolya, M., Sun, T., McRae, A.T.T., Luporini, F., Gregory, A., Lange, M., Funke, S.W., Rathgeber, F., Bercea, G.-T., Markall, G.R., 2023. Firedrake User Manual, 1st edition. Imperial College London and University of Oxford and Baylor University and University of Washington.
Hao, Z., Wang, Z., Su, H., Ying, C., Dong, Y., Liu, S., Cheng, Z., Song, J., Zhu, J., 2023a. GNOT: a general neural operator transformer for operator learning. In: International Conference on Machine Learning, pp. 12556–12569.
Hao, Z., Yao, J., Su, C., Su, H., Wang, Z., Lu, F., Xia, Z., Zhang, Y., Liu, S., Lu, L., et al., 2023b. PINNacle: a comprehensive benchmark of physics-informed neural networks for solving PDEs. arXiv preprint. arXiv:2306.08827.
He, Q., Barajas-Solano, D., Tartakovsky, G., Tartakovsky, A.M., 2020. Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. Adv. Water Resour. 141, 103610.
Ho, J., Jain, A., Abbeel, P., 2020. Denoising diffusion probabilistic models. In: Advances in Neural Information Processing Systems, vol. 33, pp. 6840–6851.
Hofmann, S., Kim, S., 2004. Gaussian estimates for fundamental solutions to certain parabolic systems. Publ. Mat., 481–496.

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural Netw. 4 (2), 251–257.

Hsing, T., Eubank, R., 2015. Theoretical Foundations of Functional Data Analysis, with an Introduction to Linear Operators. John Wiley & Sons.

Iserles, A., 2009. A First Course in the Numerical Analysis of Differential Equations. Cambridge University Press.

Jacot, A., Gabriel, F., Hongler, C., 2018. Neural tangent kernel: convergence and generalization in neural networks. In: Advances in Neural Information Processing Systems, vol. 31.

Jin, J., Lu, Y., Blanchet, J., Ying, L., 2023. Minimax optimal kernel operator learning via multilevel training. In: 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., et al., 2021. Highly accurate protein structure prediction with AlphaFold. Nature 596 (7873), 583–589.

Karhunen, K., 1946. Über lineare methoden in der wahrscheinlichkeitsrechnung. Ann. Acad. Sci. Fenn., Ser. A I 37, 3–79.

Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L., 2021. Physics-informed machine learning. Nat. Rev. Phys. 3 (6), 422–440.

Kingma, D.P., Ba, J., 2015. Adam: a method for stochastic optimization. In: Proc. 3rd International Conference on Learning Representation.

Kissas, G., Seidman, J.H., Guilhoto, L.F., Preciado, V.M., Pappas, G.J., Perdikaris, P., 2022. Learning operators with coupled attention. J. Mach. Learn. Res. 23 (1), 9636–9698.

Kovachki, N., Lanthaler, S., Mishra, S., 2021. On universal approximation and error bounds for Fourier neural operators. J. Mach. Learn. Res. 22, 1–76.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., Anandkumar, A., 2023. Neural operator: learning maps between function spaces with applications to PDEs. J. Mach. Learn. Res. 24 (89), 1–97.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 25.

Kurth, T., Subramanian, S., Harrington, P., Pathak, J., Mardani, M., Hall, D., Miele, A., Kashinath, K., Anandkumar, A., 2023. Fourcastnet: accelerating global high-resolution weather forecasting using adaptive Fourier neural operators. In: Proceedings of the Platform for Advanced Scientific Computing Conference, pp. 1–11.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., et al., 2023. Learning skillful medium-range global weather forecasting. Science, eadi2336.

Lanthaler, S., Mishra, S., Karniadakis, G.E., 2022. Error estimates for DeepONets: a deep learning framework in infinite dimensions. Trans. Math. Appl. 6 (1).

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86 (11), 2278–2324.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521 (7553), 436–444.

Levitt, J., Martinsson, P.-G., 2022a. Linear-complexity black-box randomized compression of hierarchically block separable matrices. arXiv preprint. arXiv:2205.02990.

Levitt, J., Martinsson, P.-G., 2022b. Randomized compression of rank-structured matrices accelerated with graph coloring. arXiv preprint. arXiv:2205.03406.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A., 2020a. Neural operator: graph kernel network for partial differential equations. arXiv preprint. arXiv:2003.03485.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Stuart, A., Bhattacharya, K., Anandkumar, A., 2020b. Multipole graph neural operator for parametric partial differential equations. In: Advances in Neural Information Processing Systems, vol. 33, pp. 6755–6766.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A., 2021a. Fourier neural operator for parametric partial differential equations. In: International Conference on Learning Representations.

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., Anandkumar, A., 2021b. Physics-informed neural operator for learning partial differential equations. arXiv preprint. arXiv:2111.03794.
Li, Z., Huang, D.Z., Liu, B., Anandkumar, A., 2022. Fourier neural operator with learned deformations for PDEs on general geometries. arXiv preprint. arXiv:2207.05209.
Li, Z., Kovachki, N.B., Choy, C., Li, B., Kossaifi, J., Otta, S.P., Nabian, M.A., Stadler, M., Hundt, C., Azizzadenesheli, K., et al., 2023a. Geometry-informed neural operator for large-scale 3D PDEs. arXiv preprint. arXiv:2309.00583.
Li, Z., Peng, W., Yuan, Z., Wang, J., 2023b. Long-term predictions of turbulence by implicit U-Net enhanced Fourier neural operator. Phys. Fluids 35 (7).
Lin, G., Chen, F., Hu, P., Chen, X., Chen, J., Wang, J., Shi, Z., 2023. BI-GreenNet: learning Green's functions by boundary integral network. Commun. Math. Stat. 11 (1), 103–129.
Lin, L., Lu, J., Ying, L., 2011. Fast construction of hierarchical matrix representation from matrix–vector multiplication. J. Comput. Phys. 230 (10), 4071–4087.
Loève, M., 1946. Fonctions aleatoire de second ordre. Rev. Sci. 84, 195–206.
Loshchilov, I., Hutter, F., 2019. Decoupled weight decay regularization. In: International Conference on Learning Representations.
Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.E., 2021a. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nat. Mach. Intell. 3 (3), 218–229.
Lu, L., Meng, X., Mao, Z., Karniadakis, G.E., 2021b. DeepXDE: a deep learning library for solving differential equations. SIAM Rev. 63 (1), 208–228.
Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., Karniadakis, G.E., 2022. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. Comput. Methods Appl. Mech. Eng. 393, 114778.
Lyu, Y., Zhao, X., Gong, Z., Kang, X., Yao, W., 2023. Multi-fidelity prediction of fluid flow based on transfer learning using Fourier neural operator. Phys. Fluids 35 (7).
Mao, S., Dong, R., Lu, L., Yi, K.M., Wang, S., Perdikaris, P., 2023. PPDONet: deep operator networks for fast prediction of steady-state solutions in disk–planet systems. Astrophys. J. Lett. 950 (2), L12.
Mao, Z., Jagtap, A.D., Karniadakis, G.E., 2020. Physics-informed neural networks for high-speed flows. Comput. Methods Appl. Mech. Eng. 360, 112789.
Martinsson, P.-G., 2011. A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix. SIAM J. Matrix Anal. Appl. 32 (4), 1251–1274.
Martinsson, P.-G., Tropp, J.A., 2020. Randomized numerical linear algebra: foundations and algorithms. Acta Numer. 29, 403–572.
Mathieu, M., Henaff, M., LeCun, Y., 2014. Fast training of convolutional networks through FFTs. In: International Conference on Learning Representations.
Mercer, J., 1909. Functions of positive and negative type, and their connection with the theory of integral equations. Philos. Trans. R. Soc. A 209, 415–446.
Minakshisundaram, S., Pleijel, Å., 1949. Some properties of the eigenfunctions of the Laplace-operator on Riemannian manifolds. Can. J. Math. 1 (3), 242–256.
Moya, C., Zhang, S., Lin, G., Yue, M., 2023. Deeponet-grid-uq: a trustworthy deep operator framework for predicting the power grid's post-fault trajectories. Neurocomputing 535, 166–182.
O'Leary-Roseberry, T., Chen, P., Villa, U., Ghattas, O., 2024. Derivative-informed neural operator: an efficient framework for high-dimensional parametric derivative learning. J. Comput. Phys. 496, 112555.
Olver, P.J., 1993a. Applications of Lie Groups to Differential Equations. Springer Science & Business Media.
Olver, P.J., 1993b. Applications of Lie Groups to Differential Equations, 2nd edition. Springer-Verlag.
Otto, S.E., Zolman, N., Kutz, J.N., Brunton, S.L., 2023. A unified framework to enforce, discover, and promote symmetry in machine learning. arXiv preprint. arXiv:2311.00212.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al., 2022. Training language models to follow instructions with human feedback. In: Advances in Neural Information Processing Systems, vol. 35, pp. 27730–27744.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. Pytorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32.

Peng, R., Dong, J., Malof, J., Padilla, W.J., Tarokh, V., 2023. Deep generalized Green's functions. arXiv preprint. arXiv:2306.02925.

Peng, W., Yuan, Z., Wang, J., 2022. Attention-enhanced neural network models for turbulence simulation. Phys. Fluids 34 (2).

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. J. Comput. Phys. 378, 686–707.

Raonic, B., Molinaro, R., Rohner, T., Mishra, S., de Bezenac, E., 2023. Convolutional neural operators. In: ICLR 2023 Workshop on Physics for Machine Learning.

Rasmussen, C.E., Williams, C., 2006. Gaussian Processes for Machine Learning. MIT Press.

Rathgeber, F., Ham, D.A., Mitchell, L., Lange, M., Luporini, F., McRae, A.T., Bercea, G.-T., Markall, G.R., Kelly, P.H., 2016. Firedrake: automating the finite element method by composing abstractions. ACM Trans. Math. Softw. 43 (3), 1–27.

Richter-Powell, J., Lipman, Y., Chen, R.T., 2022. Neural conservation laws: a divergence-free perspective. In: Advances in Neural Information Processing Systems, vol. 35, pp. 38075–38088.

Ritter, K., Wasilkowski, G.W., Woźniakowski, H., 1995. Multivariate integration and approximation for random fields satisfying Sacks-Ylvisaker conditions. Ann. Appl. Probab., 518–540.

Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G., 2008. The graph neural network model. IEEE Trans. Neural Netw. 20 (1), 61–80.

Schäfer, F., Owhadi, H., 2021. Sparse recovery of elliptic solvers from matrix-vector products. arXiv preprint. arXiv:2110.05351.

Schäfer, F., Sullivan, T.J., Owhadi, H., 2021. Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity. Multiscale Model. Simul. 19 (2), 688–730.

Schmidt, M., Lipson, H., 2009. Distilling free-form natural laws from experimental data. Science 324 (5923), 81–85.

Searson, D.P., Leahy, D.E., Willis, M.J., 2010. GPTIPS: an open source genetic programming toolbox for multigene symbolic regression. In: Proceedings of the International Multiconference of Engineers and Computer Scientists, vol. 1. Citeseer, pp. 77–80.

Sirignano, J., Spiliopoulos, K., 2018. DGM: a deep learning algorithm for solving partial differential equations. J. Comput. Phys. 375, 1339–1364.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S., 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In: International Conference on Machine Learning, pp. 2256–2265.

Son, H., Jang, J.W., Han, W.J., Hwang, H.J., 2021. Sobolev training for physics informed neural networks. arXiv preprint. arXiv:2101.08932.

Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B., 2021. Score-based generative modeling through stochastic differential equations. In: International Conference on Learning Representations.

Stein, M.L., 1999. Interpolation of Spatial Data: Some Theory for Kriging. Springer Science & Business Media.

Stepaniants, G., 2023. Learning partial differential equations in reproducing kernel Hilbert spaces. J. Mach. Learn. Res. 24 (86), 1–72.

Stuart, A.M., 2010. Inverse problems: a Bayesian perspective. Acta Numer. 19, 451–559.

Süli, E., Mayers, D.F., 2003. An Introduction to Numerical Analysis. Cambridge University Press.

Sun, J., Liu, Y., Wang, Y., Yao, Z., Zheng, X., 2023. BINN: a deep learning approach for computational mechanics problems based on boundary integral equations. Comput. Methods Appl. Mech. Eng. 410, 116012.

Trefethen, L.N., 2000. Spectral Methods in MATLAB. SIAM.
Tripura, T., Chakraborty, S., 2022. Wavelet neural operator: a neural operator for parametric partial differential equations. arXiv preprint. arXiv:2205.02191.
Udrescu, S.-M., Tegmark, M., 2020. AI Feynman: a physics-inspired method for symbolic regression. Sci. Adv. 6 (16), eaay2631.
Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., Tegmark, M., 2020. AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. In: Advances in Neural Information Processing Systems, vol. 33, pp. 4860–4871.
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. In: Advances in Neural Information Processing Systems, vol. 30.
Venturi, S., Casey, T., 2023. Svd perspectives for augmenting deeponet flexibility and interpretability. Comput. Methods Appl. Mech. Eng. 403, 115718.
Wang, S., Wang, H., Perdikaris, P., 2021a. On the eigenvector bias of Fourier feature networks: from regression to solving multi-scale PDEs with physics-informed neural networks. Comput. Methods Appl. Mech. Eng. 384, 113938.
Wang, S., Wang, H., Perdikaris, P., 2021b. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. Sci. Adv. 7 (40), eabi8605.
Wang, S., Wang, H., Perdikaris, P., 2022a. Improved architectures and training algorithms for deep operator networks. J. Sci. Comput. 92 (2), 35.
Wang, S., Yu, X., Perdikaris, P., 2022b. When and why PINNs fail to train: a neural tangent kernel perspective. J. Comput. Phys. 449, 110768.
Wang, S., Sankaran, S., Wang, H., Perdikaris, P., 2023. An expert's guide to training physics-informed neural networks. arXiv preprint. arXiv:2308.08468.
Weyl, H., 1911. Über die asymptotische verteilung der eigenwerte. Nachr. Ges. Wiss. Gött., Math.-Phys. Kl. 1911, 110–117.
Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y., 2020. A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. 32 (1), 4–24.
Yarotsky, D., 2017. Error bounds for approximations with deep ReLU networks. Neural Netw. 94, 103–114.
Ying, L., Biros, G., Zorin, D., 2004. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. J. Comput. Phys. 196 (2), 591–626.
You, H., Zhang, Q., Ross, C.J., Lee, C.-H., Yu, Y., 2022. Learning deep implicit Fourier neural operators (IFNOs) with applications to heterogeneous material modeling. Comput. Methods Appl. Mech. Eng. 398, 115296.
Yu, A., Yang, Y., Townsend, A., 2023. Tuning frequency bias in neural network training with nonuniform data. In: International Conference on Learning Representations.
Zhao, X., Chen, X., Gong, Z., Zhou, W., Yao, W., Zhang, Y., 2024. RecFNO: a resolution-invariant flow and heat field reconstruction method from sparse observations via Fourier neural operator. Int. J. Therm. Sci. 195, 108619.
Zheng, H., Nie, W., Vahdat, A., Azizzadenesheli, K., Anandkumar, A., 2023. Fast sampling of diffusion models via operator learning. In: International Conference on Machine Learning, pp. 42390–42402.
Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M., 2020. Graph neural networks: a review of methods and applications. AI Open 1, 57–81.
Zhu, H., Williams, C.K., Rohwer, R., Morciniec, M., 1998. Gaussian regression and optimal finite dimensional linear models. In: Neural Networks and Machine Learning. Springer-Verlag.
Zhu, M., Zhang, H., Jiao, A., Karniadakis, G.E., Lu, L., 2023. Reliable extrapolation of deep neural operators informed by physics or sparse observations. Comput. Methods Appl. Mech. Eng. 412, 116064.