Journal of Machine Learning **ISSN**: 2790-2048(e), 2790-203X(p)

The Cost-Accuracy Trade-Off in Operator Learning with Neural Networks

Maarten V. de Hoop* [1], Daniel Zhengyu Huang† [2], Elizabeth Qian‡ [2], and Andrew M. Stuart§ [2]

[1]Rice University, Houston, TX, USA.
[2]California Institute of Technology, Pasadena, CA, USA.

Abstract. The term 'surrogate modeling' in computational science and engineering refers to the development of computationally efficient approximations for expensive simulations, such as those arising from numerical solution of partial differential equations (PDEs). Surrogate modeling is an enabling methodology for many-query computations in science and engineering, which include iterative methods in optimization and sampling methods in uncertainty quantification. Over the last few years, several approaches to surrogate modeling for PDEs using neural networks have emerged, motivated by successes in using neural networks to approximate nonlinear maps in other areas. In principle, the relative merits of these different approaches can be evaluated by understanding, for each one, the cost required to achieve a given level of accuracy. However, the absence of a complete theory of approximation error for these approaches makes it difficult to assess this cost-accuracy trade-off. The purpose of the paper is to provide a careful numerical study of this issue, comparing a variety of different neural network architectures for operator approximation across a range of problems arising from PDE models in continuum mechanics.

Keywords:
Computational partial differential equations,
Surrogate modeling,
Operator approximation,
Neural networks,
Computational complexity.

Article Info.:
Volume: 1
Number: 3
Pages: 299- 341
Date: September/2022
doi.org/10.4208/jml.220509

Article History:
Received: 9/5/2022
Accepted: 6/9/2022

Communicated by:
Weinan E

1 Introduction

In many problems in computational partial differential equations (PDEs) the fundamental driver in deciding which approximation methodology to employ is the shape of the cost-accuracy curve: this determines what computational resources are required to achieve a desired level of accuracy, a measure of computational complexity. On this basis some methods may be shown to clearly outperform others, guiding computational practice. In the numerical analysis of PDEs there is a deep literature addressing this issue. This literature comprises two main components: (i) an analysis of the error as a function of the resolution of the finite dimensional approximation [1–5]; and (ii) analysis of the cost of running the model, at a given level of finite-dimensional resolution, often dominated by matrix inversion and/or matrix-vector multiplies [6, 7] and/or by time-stepping and

*mdehoop@rice.edu.
†dzhuang@caltech.edu.
‡eqian@caltech.edu.
§astuart@caltech.edu.

iteration count for nonlinear solvers. Theoretical results in (i) and (ii) may be combined to determine the cost-accuracy curve for different methods and thereby inform the choice of method for a given problem. For certain classes of equations, multi-resolution methods have emerged which are near optimal in terms of minimizing cost for a given error [8, 9].

Data-driven approximation of mappings/operators between function spaces provides a way to learn cheap-to-evaluate surrogates which can bypass the need for employing PDE solvers, after an initial training phase in which data are generated. These surrogates then enable efficient many-query analyses of PDE-based problems in computational science and engineering. However, the theory for data-driven approximations is in its infancy and cost-accuracy curves are not analytically understood. The goal of this work is to provide a numerical study of the cost accuracy trade-off, for a range of operator neural network architectures, including PCA-based neural networks (PCA-Net) [10, 11], DeepONet [12, 13], pointwise evaluation (PARA-Net, defined in this paper), and the Fourier neural operator (FNO) [14, 15]. The numerical studies are conducted on four test problems: (1) the two-dimensional incompressible Navier-Stokes equation, (2) the Helmholtz equation, (3) a structural mechanics test problem, and (4) the linear advection equation.

There are four sources of error in these operator learning problems: a) discretization of the input and output spaces; b) parameterization of the operator approximators; c) finite data volume; d) the optimizer used in training. In this paper we concentrate on b) and c) and study the cost-accuracy trade-off in relation to data volume and number of parameters in the neural network. The reason for not studying a) in this work is that, if properly designed, operator approximators have the property of discretization invariance, meaning that they are defined to act between function spaces and training of parameters for one discretization can therefore be used for other discretizations [11,14,15]; in this setting a single set of parameters will provide good approximations for all resolutions for which the discretization error is small enough. As for the role of the optimizer d), while there exists numerical evidence that stochastic gradient descent methods can be effective in driving the loss function (close) to its global minimum [16–18], this work is far from being theoretically well-understood and, furthermore, is not in the context of operator learning and partial differential equations. There are also other optimization approaches, for example using second-order information [19, 20], or ensemble methods [21, 22], that may produce different results. However, in order to limit the scope of our numerical studies, we employ stochastic gradient descent using fixed standard choices of the optimization hyperparameters for all test cases. When there is evidence that the optimization itself limits the accuracy achieved, we will highlight this in our discussion. With this caveat, we primarily focus our study on b) and c) and extract clear signals from numerical experiments, laying foundation for future studies which delve into the interactions with a) and d). Our numerical experiments will disentangle the roles of errors caused by b) and by c). The seminal work of Giles [23, 24] on multilevel Monte Carlo methods demonstrates that a theoretical understanding of errors incurred through the interaction of finite sampling and finite dimensional approximation leads to highly efficient methods which use different sample sizes for different finite dimensional approximations of expectations. Future analysis studying the interaction between the sources of error arising from b) and c) would be very valuable in the field of operator approximation and could lead to improved complexity

results, in the spirit of this work of Giles. Furthermore, although we downplay the effect of discretization error a) in this paper, future analysis studying the interaction between the sources of error arising from a), b) and c) could be also very valuable in minimizing the cost of the objective evaluation during training.

1.1 Literature review

Many-Query Motivation. Many computational tasks arising in science and engineering require repeatedly evaluating the output of an expensive computational model, which generally involve solving parametric partial differential equations. Examples include design optimization [25–28], Bayesian calibration and inference [29–35], and multiscale computation [36–42]. In some settings, this computational model may be viewed as a black box mapping functions to functions, making the development of efficient data-driven surrogate models highly desirable. Furthermore, there are increasingly complicated application domains which are data-rich, but for which complexity limits first principles modeling; thus model discovery from data is necessitated. Cyber-physical systems present a wide-class of such problems [43]. Data-driven modeling has great potential to impact such domains.

Data-Driven Surrogate Models. A variety of specific techniques for data-driven surrogate modeling have been developed and described in the literature, including the Koopman operator-based models [44–49], Gaussian process (GP) based model emulators [50–52], and data-driven projection-based reduced models [53–61]. Some of these methods are non-intrusive, or can be extended to non-intrusive instantiations, which can be constructed without prior knowledge of an underlying mechanistic model. The motivation for such methods is to find fast but accurate replacements for expensive computational tasks which need to be repeatedly executed.

Neural Network Based Surrogate Models. Most of the aforementioned surrogate models have at their core a linear model represented as combinations of appropriate basis functions. Composing such models with pointwise nonlinearities in layers leads to neural network based surrogate models. The introduced nonlinearity in the surrogate models does not lead to significant increase in evaluation cost, since it is pointwise, but can significantly improve expressivity and prediction accuracy. As a consequence, neural network based surrogate models are being explored in different fields in science and engineering. They are used for data-driven modeling and scientific discovery, including turbulence modeling [62–65], material modeling [39, 66–69], quantum mechanical modeling of materials [67, 70] and the design and prediction of protein structures [71]. In addition, neural networks have been used to augment conventional data-driven surrogate modeling frameworks. For example, proper orthogonal decomposition is coupled with neural networks to speed up the online stage of reduced models in [10, 72], nonlinear manifolds are introduced and learned by convolutional autoencoders to extend projection-based model reduction beyond approximation in linear subspaces in [73], the approximation of the Koopman operator is enhanced by neural networks for the control of unsteady fluid flows

[74, 75], relationships between GPs and deep neural networks are explored in [76], and Bayesian neural networks for uncertainty quantification are investigated in [77, 78].

Neural Networks for Solving Partial Differential Equations. Solving partial differential equations with neural network-based approximations provides an alternative to traditional approximation approaches (such as finite difference, finite element or spectral methods) by introducing new forms of finite dimensional approximation spaces. Potential advantages include the fact that neural network-based approximations do not require meshes, and also offer the possibility of exploiting nonlinear approximation spaces, both of which may be of great benefit in the numerical solution of high-dimensional PDEs [79–81], and in solving PDEs on complex geometries [82–84]. Moreover, well-developed neural network libraries (i.e., TensorFlow [85] and PyTorch [86]), coupled with GPU architecture, make the solution process both easy-to-use and efficient. The basic methodology in PINNs [83] introduces a loss function defined by the equation, and may be extended to inverse problems through the additional incorporation of data-related penalty terms [87–89]. The related gradient can generally be computed automatically by these well-developed neural network libraries, and hence the optimization or inverse solving procedure are also both easy-to-use and efficient. Theoretical underpinnings of this framework are presented in [90–93] and approaches to improving the performance with locally adaptive activation functions are studied in [94–97].

Neural Networks for Operator Learning. In many of the applications cited in the preceding paragraphs, in both surrogate modeling and in model discovery, the core task is the mapping of one function into another. Much of the preceding work on neural networks for PDEs can be naively extended to neural network methods which would need to be retrained for each different input instance; some of the reduced order modeling work proceeds by learning mappings on finite dimensional spaces without explicitly conceptualizing or recognizing the function space mapping at its core. The focus of the present work is on infinite-dimensional formulations for learning mappings between function spaces, and we refer to this as operator learning. There has been considerable activity in this area in the last five years: starting with the paper [77], focused on surrogate modeling for uncertainty quantification in subsurface flow using deep autoencoder networks; [98] who use a convolutional neural network to minimize parameterized variational problems; DeepONet [12] with architecture comprising two sub-networks used to represent the operator by enforcing separation of input and output functions into the branch and trunk networks; PCA-Net [10, 11] which uses neural networks to map between PCA coefficients representing input and output functions, neural networks based on kernel integral operators [15], and the FNO [14, 99, 100] which parameterizes the integral kernel directly in Fourier space. Finally, we mention [101, 102] which study the learning of Greens functions for nonlinear boundary value problems, [98, 103–105] which propose novel neural network architectures for solving wave and elliptic equations, and [106] which employs the attention mechanism for operator learning. A key aspect of the work in [11, 15] is that the methodology has been designed to be robust to the resolution of the finite-dimensionalizations of the input and output functions; this means that the computational investment in learning a model at

one resolution, or for one particular discretization, may be transferred to other resolutions or discretizations. There is an existing body of published work which compares resolution invariant methods such as PCA-NET, DeepONet and FNO, with methods where the choice and training of parameters is linked to the grid resolution, such as U-NET and other grid-adapted approaches [77]; see [11, 14] for such comparisons. These comparative studies demonstrate the value of using resolution invariant methods.

Universal approximation theorems, stating that neural networks can accurately approximate wide classes of nonlinear continuous operators are starting to emerge to underpin these methodologies [12, 15, 107]. The paper [13] undertakes a careful comparison of the relative merits of DeepONet-based and FNO-based methodologies for operator learning. The paper introduces variants on the basic methods and the results show the impressive flexibility of DeepONet and its ability to solve problems from a variety of physical applications and a variety of complex geometric domains, and suggest directions in which the FNO will benefit from further innovations. The specific numerical results presented, in which a fixed number of parameters is chosen for each method, and the error compared, show smaller errors for DeepONet than for FNO in many examples; however similarly designed experiments on different problems and with different choices of the fixed parameters reach the opposite conclusion about the ordering of the errors incurred by DeepONet and FNO [15, 106]. Thus, in terms of evaluating the relative merits of different surrogates, the papers [13, 15, 106] come up short, because they do not study dependence of evaluation cost on the approximation error achieved. This question is the primary focus of our paper.

1.2 Our contributions

Neural network surrogates for operators require a significant volume of training data to ensure reasonable predictive power. The number of parameters needs to increase with data volume, and when the parameterization of the neural network architecture becomes more complicated, the evaluation cost increases. Quantifying the resulting cost-accuracy trade-off involves interaction between statistical error from data sampling and approximation error from parameterization; in particular the issue of how to choose the number of parameters, given the amount of available data, is fundamental to success and efficiency of the methodology. Understanding the trade-off between cost and accuracy, for different neural operators, and in different parameter/data regimes, is thus of central importance in guiding how this field develops; it forms the focus of this work. Because theory in this arena is currently limited, our study is purely computational. Our belief is that careful computational studies will provide impetus for the development of theoretical understanding of the issues they explore. Our main contributions are as follows:

- We give a unified presentation of a variety of different operator approximators: three have appeared in the recent literature, PCA-Net, DeepONet and FNO, whilst a fourth, a lifting of finite dimensional neural networks via pointwise evaluation, which we label PARA-Net, is formalized in this paper.
- We numerically study these operator approximators in the context of four model

problems: i) mapping forcing to solution in a 2D incompressible Newtonian fluid; ii) mapping wavespeed to disturbance field in the Helmholtz equation; iii) mapping applied tension load to stress field in an elastic solid; and iv) mapping initial condition to solution at time one in the advection equation.

- We study error as a function of data volume (for fixed number of parameters), error as a function of number of parameters (for fixed data volume), and evaluation cost as a function of error, for all four test problems. These experiments quantify anticipated differences in the cost-accuracy trade-off between problems with smooth outputs (i,ii) and those with discontinuous outputs (iii,iv). The experiments also demonstrate data limiting effects and the potential for over-fitting in some methods.
- We show that the PARA-Net approach, whilst being a natural generalization of neural networks between Euclidean spaces, is not competitive with the other three methods considered, for the problems we consider, as measured by evaluation cost per unit accuracy. PCA-Net, DeepONet and FNO all exhibit desirable behavior, and are all clearly viable methodologies for certain problems in certain regimes; in contrast PARA-Net is both consistently more expensive whilst also being particularly prone to over-fitting, except for problem (iv); as a result it does not appear to be a useful general approach. We include PARA-Net because doing so highlights the drawbacks of not conceptualizing operator approximation as learning a function to function mapping.
- For a number of test problems and operator approximations, we provide explicit instances of the test cases which lead to the median test error and to the largest test error, yielding insight into failure modes of the learning procedures, especially for the non-smooth problems (iii,iv).
- We numerically study the range space of the DeepONet approximator, contrasting it with PCA-Net, demonstrating the pros and cons of its basic implementation, and motivating the PCA modification of the basic DeepONet method that is introduced in [13].

Our experiments do not provide a definitive answer as to which operator approximator is best for any given class of problems. Rather we focus our attention on the following *complexity* question: how does evaluation cost scale with accuracy for these methods? Our experiments demonstrate that the answer to this question depends on the problem. In a similar vein, the results in [13, 15, 106] also indicate that preferred method with respect to error at a fixed parametrization level is problem dependent. However, since [13, 15, 106] consider only fixed parameterizations, and hence fixed cost, they do not shed light on complexity. Our experiments demonstrate the merits of considering the complexity question, and suggest the need for theory to underpin empirical findings. We believe conclusions as to preferred methodology for any given problem, even with respect to the complexity measure studied here, are premature; our numerical results simply focus on the need for theory which addresses the complexity question.

The code and dataset are accessible online at `https://github.com/Zhengyu-Huang/Operator-Learning`.

2 Problem formulation

Consider the following Banach spaces of functions $\mathcal{U}, \mathcal{V}$:

$$\mathcal{U} = \{u : D_u \to \mathbb{R}^{d_i}\}, \text{ where } D_u \subseteq \mathbb{R}^{d_x},$$
$$\mathcal{V} = \{v : D_v \to \mathbb{R}^{d_o}\}, \text{ where } D_v \subseteq \mathbb{R}^{d_y}.$$

Our goal is to determine operators $\Psi^\dagger : \mathcal{U} \to \mathcal{V}$ from (samples from) the probability measure $\left(\mathrm{Id}, \Psi^\dagger\right)^\# \mu$, where μ is supported on $\mathcal{U}$ and equipped with the Borel σ-algebra, and the push-forward $\left(\mathrm{Id}, \Psi^\dagger\right)^\# \mu$ is supported on $\mathcal{U} \times \mathcal{V}$, also equipped with the Borel σ-algebra. For simplicity we will assume that D_u and D_v are closed and bounded.

We approximate $\Psi^\dagger$ by different classes of parametric operators, each of which characterizes a different neural network architecture. To this end, let $\Theta \subseteq \mathbb{R}^p$ denote the space of parameters of the neural network and consider a family of functions from $\mathcal{U}$ into $\mathcal{V}$ defined by $\Psi : \mathcal{U} \times \Theta \mapsto \mathcal{V}$. In the idealized case of infinite data, the parameters $\theta \in \Theta$ are chosen to be $\theta^\star$, the minimizer of the risk defined by:

$$\text{Risk:} \quad \mathcal{R}_\infty(\theta) := \mathbb{E}^{u\sim\mu}\|\Psi^\dagger(u) - \Psi(u;\theta)\|_{\mathcal{V}}^2 = \|\Psi^\dagger - \Psi\|_{L^2_\mu(\mathcal{U},\mathcal{V})}^2,$$

where the last expression is defined with integration over $\mathcal{U}$ in the Bochner sense. In practice we only have access to samples $\{u_n\}_{n=1}^N$ from μ, assumed to be i.i.d. and defining the resulting empirical measure μ^N, together with $\{\Psi^\dagger(u_n)\}_{n=1}^N$. This enables us to define the empirical risk:

$$\begin{aligned}\text{Empirical Risk:} \quad \mathcal{R}_N(\infty)(\theta) :=& \mathbb{E}^{u\sim\mu^N}\|\Psi^\dagger(u) - \Psi(u;\theta)\|_{\mathcal{V}}^2 \\ =& \frac{1}{N}\sum_{n=1}^N \|\Psi^\dagger(u_n) - \Psi(u_n;\theta)\|_{\mathcal{V}}^2.\end{aligned}$$

Parameter $\theta^{\star,N}$ is the value of θ which minimizes $\mathcal{R}_N(\infty)(\theta)$. In practice, in view of the non-convex nature of the optimization over θ, we may only have access to an approximation of $\theta^{\star,N}$.

Remark 2.1. Within both the risk and the empirical risk various modifications may be relevant; for example $\|\cdot\|_{\mathcal{V}}$ may be replaced by the norm in any space into which $\mathcal{V}$ is continuously embedded. When studying error as a function of data volume, parameter dimension, and evaluation cost, we will also use the following relative error measure (approximated empirically):

$$\mathbb{E}^\mu\left(\frac{\|\Psi^\dagger(u) - \Psi(u;\theta^{\star,N})\|_{\mathcal{V}}}{\|\Psi^\dagger(u)\|_{\mathcal{V}}}\right). \tag{2.1}$$

Remark 2.2. Many of our examples concern the case where $D_u = D_v = D$, $d_x = d_y = d$. Furthermore, many of our examples also concern the case where the vector-valued functions in $\mathcal{U}$ and $\mathcal{V}$ are in fact scalar-valued so that the input and output dimensions are $1 : d_i = d_o = 1$. However, since the methodology applies outside these simpler settings, we expose basic ideas at this greater level of generality.

3 Neural networks

In this section we describe the various neural networks that we will compare in this study. We establish preliminary notation in Section 3.1. In Sections 3.2 to 3.5, we introduce four classes of neural network, highlighting similarities and differences.

3.1 Preliminaries

Let $\mathcal{H}$ denote a Hilbert space with inner product $\langle \cdot, \cdot \rangle$ and induced norm $\| \cdot \|$. We then consider the Gelfand triple $\mathcal{U} \subseteq \mathcal{H} \subseteq \mathcal{U}^*$ where the embedding of $\mathcal{U}$ in $\mathcal{H}$ is continuous and $\mathcal{U}^*$ denotes the dual space of $\mathcal{U}$ – the space of linear functionals on $\mathcal{U}$. Given $L_k \in \mathcal{U}^*$, $k = 1, \cdots d_u$, the function u in $\mathcal{U}$ is partially characterized by the finite dimensional vector $Lu := \{L_k u\}_{k=1}^{d_u}$. This will be used to characterize inputs to Ψ in the formulations of the three neural network architectures in Sections 3.2 to 3.4.

When $\mathcal{U}, \mathcal{V}$ are themselves Hilbert spaces we may compute the mean and covariance operator with respect to μ and $(\Psi^\dagger)^\# \mu$, respectively, assuming first and second moments exist. The eigen-decomposition of the covariance operators (PCA) gives rise to orthonormal bases $\{\phi_j\}_{j \in \mathbb{N}}$ in $\mathcal{U}$ and $\{\psi_j\}_{j \in \mathbb{N}}$ in $\mathcal{V}$. Note that

$$\phi_j : D_u \to \mathbb{R}^{d_i}, \quad j \in \mathbb{N}; \qquad \psi_j : D_v \to \mathbb{R}^{d_o}, \quad j \in \mathbb{N}.$$

3.2 PCA-Net

Here it is simplest to think of $\mathcal{H} = \mathcal{U} = L^2(D_u; \mathbb{R}^{d_i})$. Truncate the PCA basis of μ to the first d_u modes, $\{\phi_j\}_{j=1}^{j=d_u}$, and truncate the PCA basis of $\Psi^\# \mu$ to the first d_v modes $\{\psi_j\}_{j=1}^{j=d_v}$. Then define $L_k \in \mathcal{U}^*$ by $L_k u = \langle \phi_k, u \rangle$, $k = 1, \cdots d_u$. Introduce function $\alpha : \mathbb{R}^{d_u} \times \Theta \to \mathbb{R}^{d_v}$ defined componentwise by

$$\alpha_j : \mathbb{R}^{d_u} \times \Theta \to \mathbb{R}, \quad j = 1, 2, \cdots, d_v,$$

which maps from the PCA coefficients $\{L_k u\}_{k=1}^{d_u}$ of μ to the PCA coefficients $\{\alpha_j\}_{j=1}^{d_v}$ of $\Psi^\# \mu$, and is parameterized by $\theta \in \Theta$. Here function α denotes a finite dimensional neural network.

The approximating operator between $\mathcal{U}$ and $\mathcal{V}$ is then defined by

$$\Psi_{PCA}(u; \theta)(y) = \sum_{j=1}^{d_v} \alpha_j(Lu; \theta) \psi_j(y) = \alpha(Lu; \theta)^T \psi(y), \quad \forall u \in U \quad y \in D_v,$$

where $Lu = \{L_k u\}_{k=1}^{d_u}$ and $\psi(y) = \{\psi_j(y)\}_{k=1}^{d_v}$. We note that each α_j is $\mathbb{R}$-valued, so α may be viewed as a column vector in $\mathbb{R}^{d_v}$, whilst each ψ_j is $\mathbb{R}^{d_o}$-valued, so ψ may be viewed as a matrix in $\mathbb{R}^{d_v} \times \mathbb{R}^{d_o}$. Thus $\Psi_{PCA}(u; \theta, \mu)(y)$ is $\mathbb{R}^{d_o}$-valued.

Remark 3.1. Note the following facts concerning this architecture which place it slightly outside the standard training framework, using empirical risk, outlined above.

- PCA is defined on a Hilbert space, given measure μ; in practice it is implemented on finite-dimensional approximations of a Hilbert space, using samples from measure μ.
- The linear functionals $\{L_k\}$ in $\mathcal{U}^*$ and the functions $\{\psi_j\}$ in $\mathcal{V}$, needed to define the architecture of Ψ_{PCA}, are pre-computed using PCA and decoupled from the training of the neural network. Thus preliminary calculations, using PCA on the dataset, are conducted to define the risk or empirical risk.

3.3 DeepONet

Several works in recent years use the terms 'DeepONet' or 'deep operator network' to describe related, but different neural network architectures used to approximate infinite-dimensional operators [12, 84, 108–110]. These architectures have in common the use of two separate networks, called the *trunk* and *branch* networks, to map input functions to output functions, generalizing the 'shallowONet' architecture first proposed in [107]. The trunk network is used learn the output space representation; the branch network learns the input to output solution mapping in the output space spanned by the trunk network.

In this paper we employ a specific variant of the DeepONet branch/trunk architecture, chosen in order to facilitate comparison with PCA-Net; specifically we consider a version of DeepONet in which we use the same linear functionals in $\mathcal{U}$ as in PCA-Net as input to the branch network. However, whilst PCA-Net uses fixed PCA basis functions in the output space, DeepONet *learns* the representation in the output space as the trunk network. We note that the variant on DeepONet appearing in [12, Problem 5] also uses a Karhunen-Loeve expansion, which is equivalent to PCA, as input, but uses the information in a different way; in particular the coefficients of the Karhunen-Loeve expansion, which in our notation are analogous to Lu, are used in [12] as input to the trunk network, rather than the branch network.

In summary the key difference between our implementations of DeepONet and PCA-Net is that the functions $\{\psi_j\}$ in the output space are given in DeepONet by a neural network, rather than using PCA on the output space $\mathcal{V}$ as in PCA-Net; for DeepONet they are learned in the training phase, rather than being precomputed from the data as in PCA-Net. However, our implementations of DeepONet and PCA-Net both take linear functionals on the input PCA basis in $\mathcal{U}$ as input to the neural networks labelled α. We emphasize that the original formulation of the DeepONet used pointwise evaluations of the input function in $\mathcal{V}$ as inputs to the neural network, rather than PCA coefficients. A brief discussion of pointwise evaluations, written in our general framework, may be found in Appendix A. It is possible, indeed likely, that for some problems, our implementation of both DeepONet and PCA-Net could show improved performance by working with pointwise evaluation functionals as input; however, our numerical results are not focused on this question, but rather on how the two methods perform in relation to their different representations of the output space.

The operator between $\mathcal{U}$ and $\mathcal{V}$ is defined as follows:

$$\Psi_{DEEP}(u;\theta)(y) = \sum_{j=1}^{d_v} \alpha_j(Lu;\theta_\alpha)\psi_j(y;\theta_\psi) = \alpha(Lu;\theta_\alpha)^T\psi(y;\theta_\psi), \quad \forall u \in U, \quad y \in D_v,$$

where the functions $\alpha_j : \mathbb{R}^{d_u} \times \Theta \to \mathbb{R}$ have the same structure as in PCA-Net, and are collectively referred to as the "branch" of the DeepONet neural network. The functions $\{\psi_j\}$ are collectively referred to as the "trunk" of the DeepONet architecture, and are defined by neural networks of the form

$$\psi_j : D_v \times \Theta \to \mathbb{R}^{d_o}, \quad j = 1, \cdots, d_v.$$

We denote by θ the collection of the hyperparameters θ_α and θ_ψ appearing in α and ψ. These are computed from minimizing the (empirical) risk over $\theta \in \Theta$. No parameters in Θ are shared between α and ψ, but their optimal choice is coupled through the empirical risk minimization.

Remark 3.2. Our choice of notation highlights similarities between PCA-Net and DeepONet. However features that distinguish DeepONet from PCA-Net include:

- For DeepONet, both the trunk, $\{\psi\}$, and the branch, $\{\alpha\}$ are neural networks, whereas for PCA-Net only the branch is. In particular for DeepONet $\psi(\cdot;\theta)$ is selected via optimization during training, in contrast to PCA-Net for which ψ is defined by the data, using PCA, prior to parameter selection, via optimization, for the neural network.
- In DeepONet, L may not depend on μ if pointwise evaluations are used (see Appendix A).
- For both PCA-Net and DeepONet, the norm $\|\cdot\|_{\mathcal{V}}$ appearing in the empirical risk, or weaker norm in a space into which V is continuously embedded, is approximated using pointwise evaluations at a set of points $\{y_l\}$. Indeed, in the original work underpinning DeepONet [107] a set of fixed pointwise evaluations are part of the definition of the architecture and appear in the definition of the empirical risk used in training. We prefer to employ an operator perspective on the method, and in particular to define the empirical risk through a norm on $\mathcal{V}$, enabling comparison with other neural networks formulated as operator approximators.

3.4 PARA-Net

As we do for PCA-Net, we truncate the PCA basis of μ, $\{\phi_j\}_{j=1}^{d_u}$, using the first d_u modes, and define $L_k \in \mathcal{U}^*$ by $L_k u = \langle \phi_k, u \rangle$, $k = 1, \cdots d_u$. Again it is simplest to think of $\mathcal{H} = \mathcal{U} = L^2(D_u; \mathbb{R}^{d_i})$. We introduce the real-valued neural network

$$\psi : \mathbb{R}^{d_u} \times D_v \times \Theta \to \mathbb{R}^{d_o}$$

and then define $\Psi_{PARA} : \mathcal{U} \times \Theta \to \mathcal{V}$ by

$$\Psi_{PARA}(u;\theta)(y) = \psi(Lu, y;\theta).$$

Remark 3.3. Note that ψ is a standard neural network between Euclidean spaces. By defining a collection of linear functionals L in $\mathcal{U}^*$, evaluating this finite dimensional neural network at input (Lu, y), and varying over u in $\mathcal{U}$ and y in D_v we create an operator. In this sense it is a natural method. However, as we will show, it is not competitive with the other methods presented here in terms of the cost-accuracy trade-off for the problems we consider. We include it because it is a natural generalization but its poor performance serves as a motivation to think beyond the confines of standard finite dimensional Euclidean neural networks and to exploit structure such as PCA bases, branch-trunk decomposition or (next subsection) Fourier representation.

Recall that the graph of Ψ is the set $\{\Psi(u), u \in \mathcal{U}\}$. Both PCA-Net and DeepONet give rise to linear approximation spaces for the graph of Ψ in that, for all $u \in \mathcal{U}$, the approximation of $\Psi(u) \in \mathcal{V}$ lies in the same finite dimensional subspace of $\mathcal{V}$, independently of u, defined by the span of the $\{\psi_j\}$. In contrast PARA-Net gives rise to a an approximation space in $\mathcal{V}$ which depends nonlinearly on u. This is a potential advantage but, as we will show for the problems we consider here, this advantage is outweighed by the computational cost of the pointwise evaluations required to construct full output function reconstructions. This challenge is exacerbated by multiple spatial dimensions, since the number of grid points generally scales exponentially with spatial dimension. However it is conceivable that, for some operator approximation problems not considered here, the trade-off between nonlinear approximation and the cost resulting from repeated pointwise evaluation results in PARA-Net being competitive. The potential benefits of nonlinear approximation spaces are discussed in [111].

3.5 Fourier neural operator (FNO)

For simplicity, we consider the setting where $D_u = D_v = D = [0,1]^d$ so that $d_x = d_y = d$. In this setting it is simplest to think of $\mathcal{H} = L^2(D; \mathbb{R}^{d_i})$ (or a generalization to impose, for example, the divergence-free condition) and $\mathcal{U} = \mathcal{V} = C_{\text{per}}(D; \mathbb{R}^{d_i})$, the set of continuous periodic functions on the unit cube; alternatively $\mathcal{U}$ may be an RKHS, such as a Sobolev space of periodic functions of fractional order greater than $d/2$, which is continuously embedded into $C_{\text{per}}(D; \mathbb{R}^{d_i})$.

Let R, Q denote standard finite dimensional neural networks

$$R : \mathbb{R}^{d_i} \times \Theta \to \mathbb{R}^{d_f},$$
$$Q : \mathbb{R}^{d_f} \times \Theta \to \mathbb{R}^{d_o},$$

where d_f is the number of channels[1] used in FNO-NET, typically larger than d_i or d_o. We use R and Q to define operators $\mathcal{R}$ and $\mathcal{Q}$ by pointwise evaluation:

$$\begin{aligned}(\mathcal{R}u)(x, \theta_R) &= R(u(x), \theta_R),\\ (\mathcal{Q}u)(x, \theta_Q) &= Q(u(x), \theta_Q).\end{aligned} \tag{3.1}$$

[1]Sometimes also referred to as features; however we avoid this terminology as the terminology random features are used to describe a different concept [112] and we wish to avoid confusion.

We say the $\mathcal{R}$ operator *lifts* the input to the channels and that the $\mathcal{Q}$ operator *projects* the channels to the output. Note that $\mathcal{R}u$ is well-defined in $C_{\text{per}}(D;\mathbb{R}^{d_f})$, the space of periodic continuous functions of dimension d_f, if $u \in \mathcal{U}$ and $\mathcal{U}$ is continuously embedded into $C_{\text{per}}(D;\mathbb{R}^{d_i})$.

We now introduce notation for the l-th *Fourier Neural Layer* (FNL):

$$\mathcal{L}_l(v)(x,\theta) = \sigma\big(W_l v(x) + (\mathcal{K}v)(x;\gamma_l)\big), \tag{3.2}$$

where σ is an activation function, applied pointwise w.r.t. $x \in D$; $W_l \in \mathbb{R}^{d_f \times d_f}$ is a matrix applied pointwise to $v(x)$; $\mathcal{K}$ is a parameterized non-local operator, for which a variety of forms are commonly used [15]. In this paper we exclusively use the Fourier Neural Operator form (FNO):

$$\text{FNO}: \quad (\mathcal{K}v)(x;\gamma) = \mathcal{F}^{-1}(P(\gamma)(\mathcal{F}v))(x).$$

Here $\mathcal{F}$ denotes the Fourier transform of a periodic function $v : D \to \mathbb{R}^{d_f}$, so that $\mathcal{F}v : \mathbb{Z}^d \to \mathbb{C}^{d_f}$, and $\mathcal{F}^{-1}$ denotes its inverse, and for each point in the Fourier domain $\mathbb{Z}^d$, the action of $P(\gamma)$ is as an element of $\mathbb{C}^{d_f \times d_f}$. In this paper the FNO is implemented using a Fast Fourier Transform on a uniform lattice of pointwise evaluations of the functions v and $P(\gamma)\mathcal{F}(v)$. When evaluating $\mathcal{F}$, only the first $k_{\max}$ modes are kept. In this setting P reduces to a complex-valued $(k_{\max} \times d_f \times d_f)$-tensor, which is applied as an element of $\mathbb{C}^{d_f \times d_f}$ on each of the $k_{\max}$ Fourier modes. Then

$$\Psi_{FNO}(u;\theta) = \mathcal{Q} \circ \mathcal{L}_L \circ \cdots \mathcal{L}_2 \circ \mathcal{L}_1 \circ \mathcal{R}(u).$$

Here we denote by θ the collection of the hyperparameters θ_R and θ_Q and $\{W_l, \gamma_l\}$ for each Fourier neural layer l. As in the case of DeepONet, none of these hyperparameters are shared, but their choice is coupled through the empirical risk minimization. Note that, for simplicity, the layer width d_f is fixed, but this is not necessary. Furthermore, all numerical experiments reported here are conducted with Q and R being affine. In contrast, the experiments in [14, 113] use linear R but nonlinear Q.

Finally, we note that in the current implementation of FNO, the number of retained Fourier modes, $k_{\max}$, scales exponentially with the spatial dimension of the problem. In contrast, our implementations of PCA-Net and DeepONet do not have parameters that explicitly depend on the spatial dimension of the problem. The FNO scaling with spatial dimension may pose a computational hurdle in three spatial dimensions; however we note that this may be overcome, e.g. by employing wavelets in higher dimensions.

4 Numerical studies

This section compares numerically the aforementioned algorithms for approximating maps between infinite-dimensional function spaces. Four test problems are considered:

1. Navier-Stokes equation: the map between the forcing and the vorticity field at a later time is learned.

2. Helmholtz equation: the map between the (inhomogeneous) wavespeed field and the disturbance field (solution) is learned.

3. Structural mechanics equation: the map between an applied boundary load and the interior von Mises stress field is learned.

4. Advection equation: the solution operator from initial condition to solution at a later time is learned.

In our first two tests, the output functions are smooth, while the outputs of the latter two tests have discontinuities in the output space or its gradients.

This section is organized as follows: Section 4.1 describes our implementations of the neural networks from Section 3. Section 4.2 details, in Sections 4.2.1 to 4.2.4, each of the above four test problems; in the same subsection results are presented for each of the test problems, commenting qualitatively on the results for each test problem in its subsection. Then, Section 4.3 discusses our quantitative results for all test problems, highlighting the cost-accuracy trade-off for learning operators by neural networks.

4.1 Neural network architectures

Figure 4.1 summarizes the neural network architectures considered in our numerical experiments. In particular, the neural networks used in PCA-Net, PARA-Net, and the DeepONet branch and trunk networks have shared internal structure: they each use fully connected neural networks with three internal layers of constant fixed width w between the input and output layers, and ReLU functions are employed. The output layer for each of these networks is linear (no nonlinear activation function). The FNO network has three internal Fourier Neural Layers (defined in (3.2)) in between the initial lifting layer and the final projection layer (3.1), and uses Gaussian Error Linear Unit (GELU) activation functions. PCA-Net, DeepONet and PARA-Net are initialized based on the method described in [114] and FNO is initialized following [14,115].

To compare the online evaluation costs of the neural networks, we provide a cost analysis in terms of the requisite floating point operations (FLOPs) in Appendix B.2; the resulting costs are tabulated in Table 4.1. DeepONet and PCA-Net have the same evaluation cost because the DeepONet branch and PCA-Net have the same architecture in our implementations, and the DeepONet trunk defines basis functions which can be precomputed after the network is trained, before evaluating the trained network on new data.

We will study the cost and accuracy of the neural network approximations as the amount of data and the network size are varied. For PCA-Net, DeepONet, and PARA-Net, the widths tested are $w = \{16, 64, 128, 256, 512\}$, and the numbers of channels tested in FNO are $d_f = \{2, 4, 8, 16, 32\}$. Evaluation cost is measured in terms of FLOPs (Table 4.1). To quantify the expressive power of each of these networks, we provide a parameter complexity analysis of the networks in Table 4.2. Details can be found in Appendix B.1.

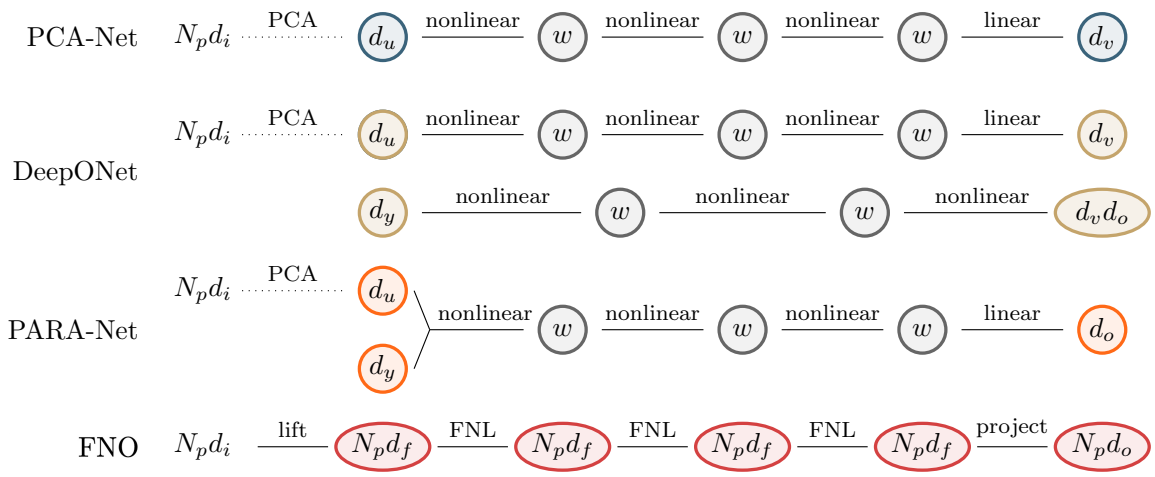

Figure 4.1: Schematic of neural network architectures in numerical experiments. Circles represent layers; the width of each layer is given in the circle. Edges represent transformations between layers; the type of transformation between each layer is noted above each edge. Nonlinear and linear transformations are standard fully-connected layers; the lift and project layers are defined in (3.1); the Fourier Neural Layer (FNL) is defined in (3.2).

Table 4.1: Evaluation cost of the four neural network architectures considered in this work.

Architecture	Evaluation cost	Cost scaling
PCA-Net	$d_u(2N_p d_i - 1) + 2d_u w + 4w^2 + 2d_v w + 3w + (2d_v - 1)N_p d_o$	$\mathcal{O}(N_p + w^2)$
DeepONet	$d_u(2N_p d_i - 1) + 2d_u w + 4w^2 + 2d_v w + 3w + (2d_v - 1)N_p d_o$	$\mathcal{O}(N_p + w^2)$
PARA-Net	$d_u(2N_p d_i - 1) + [2(d_u + d_y)w + 4w^2 + 2wd_o + 3w]N_p$	$\mathcal{O}(N_p w^2)$
FNO	$2N_p d_f(d_i + d_o) + 3(10d_f N_p log(N_p) + k_{\max}(2d_f^2 - d_f) + 2d_f^2 N_p)$	$\mathcal{O}(d_f N_p log(N_p) + N_p d_f^2)$

Table 4.2: Parameter complexity of the four neural network architectures considered in this work in terms of input and output space dimensions as well as network size parameter w or d_f.

Architecture	Parameter complexity
PCA-Net	$2w^2 + w(d_u + d_v) + 3w + d_v$
DeepONet	$4w^2 + w(d_u + d_v + d_y + d_v d_o) + 6w + d_v + d_v d_o$
PARA-Net	$2w^2 + w(d_o + d_u + d_y) + 3w + d_o$
FNO	$d_f d_i + d_f + d_f d_o + d_o + 3(d_f^2 + d_f^2 k_{\max})$

4.2 Test problems and qualitative results

This section introduces the four test problems and presents the results of our numerical comparisons of the four network architectures for each test problem. Sections 4.2.1, 4.2.2, 4.2.3 and 4.2.4, respectively, introduce the Navier-Stokes, Helmholtz, solid mechanics, and one-dimensional advection test problems, respectively. For each test problem, we show

comparisons of the true vs. neural network predicted fields at inputs that result in median and worst-case test errors and comment qualitatively on what these comparisons show about the neural network performance for each problem. Detailed quantitative discussion of the cost-accuracy trade-offs of the networks is deferred to Section 4.3.

4.2.1 Navier-Stokes equation

Formulation. We consider the vorticity-stream ($\omega - \psi$) formulation of the incompressible Navier-Stokes equations on a two-dimensional periodic domain, $D = D_u = D_v = [0, 2\pi]^2$:

$$\begin{aligned} &\frac{\partial \omega}{\partial t} + (v \cdot \nabla)\omega - \nu \Delta \omega = f', \\ &\omega = -\Delta \psi, \quad \int_D \psi = 0, \\ &v = \Big(\frac{\partial \psi}{\partial x_2}, -\frac{\partial \psi}{\partial x_1}\Big). \end{aligned} \tag{4.1}$$

We are interested in the map from the forcing f' to the vorticity field ω at time $t = T$. The forcing f' is assumed to be a centred Gaussian with covariance

$$\mathsf{C} = (-\Delta + \tau^2)^{-d};$$

here $-\Delta$ denotes the Laplacian on D subject to periodic boundary conditions on the space of spatial-mean zero functions, $\tau = 3$ denotes the inverse length scale of the random field and $d = 4$ determines its regularity; the choice of d then leads to up to 3 fractional derivatives for samples from this measure. The initial condition $\omega(0)$ is fixed and generated from the same distribution.

Our numerical experiments are conducted in the case where $\nu = 0.025$, similar to the setup in [116] [117, Chapter 2.2], and we use final time $T = 10$. For the value of ν used here, the solution at time 10 has decorrelated from the initial condition; it inherits the spatial pattern of the forcing f' but has larger amplitude, and smoother small scale features. Equation (4.1) is solved using a pseudo-spectral method on a 64×64 grid. To eliminate aliasing error, the Orszag 2/3-Rule [118] is applied and, therefore there are 42^2 Fourier modes (padding with zeros). Time-integration is performed using the CrankNicolson method with $\Delta t = 10^{-3}$. See Figure 4.2 for a visualization of sample input f' and resultant output $\omega(\cdot, T)$ fields.

Results. Figure 4.3 shows the input, true output, neural network-predicted output, and output errors for the inputs resulting in the median and largest test errors (left and right) for each network architecture. The output $\omega(T)$ for this problem is well-correlated with the input f', and all neural networks succeed in predicting the main features of the vorticity field. Note that the vorticity fields predicted by PARA-Net are grainier than those of other networks, which is reflected by the smaller length scale of the PARA-Net error fields. This graininess is due to the pointwise prediction of the network. For this problem,

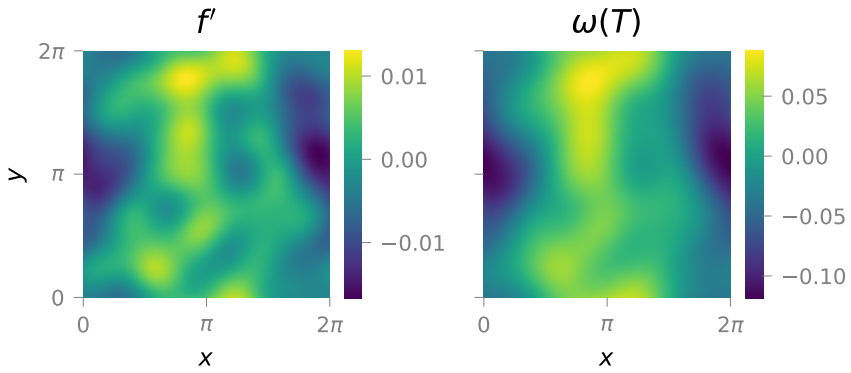

Figure 4.2: Navier-Stokes problem: sample input and output functions (left and right, respectively).

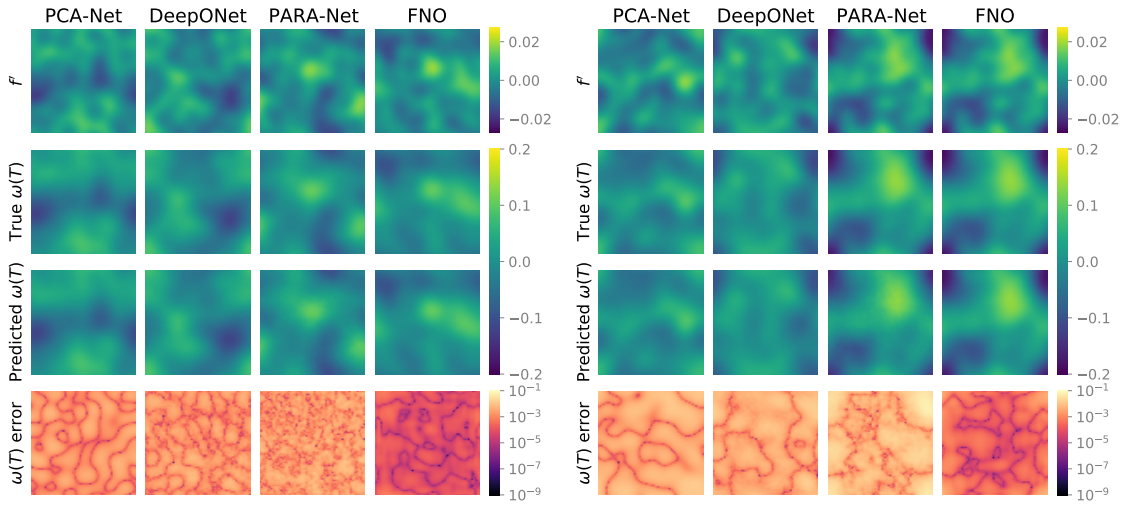

Figure 4.3: Navier-Stokes test problem: learned model vorticity predictions for inputs resulting in **median (left)** and **largest (right)** test errors for networks of size $w = 128$ / $d_f = 16$ trained on $N = 10000$ data.

FNO errors are significantly lower than those of other methods, reflecting that the problem specification is particularly well-adapted to a spectral representation.

4.2.2 Helmholtz equation

Formulation. We consider the Helmholtz equation on the domain $D = D_u = D_v = [0,1]^2$ shown in Figure 4.4. Given frequency $\omega = 10^3$ and wavespeed field $c : \Omega \to \mathbb{R}$, the excitation field $u : \Omega \to \mathbb{R}$ solves equation

$$\left(-\Delta - \frac{\omega^2}{c^2(x)}\right) u = 0 \quad \text{in } \Omega, \tag{4.2a}$$

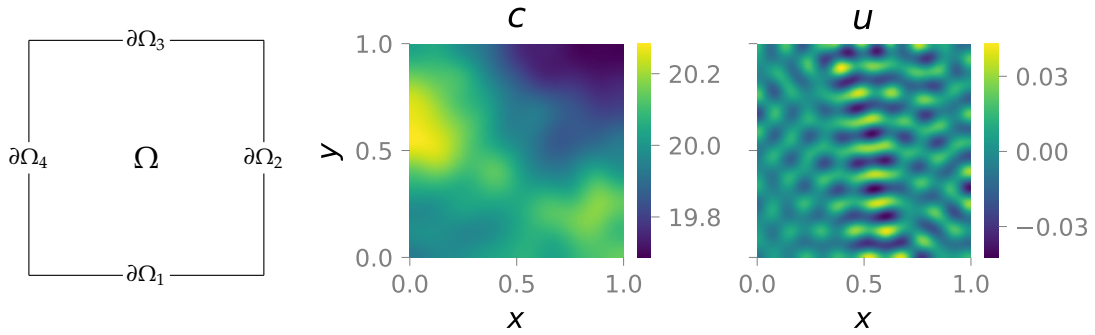

Figure 4.4: Helmholtz test problem: **Left:** schematic of unit domain with labeled boundaries. **Center:** Sample input wave speed field. **Right:** Sample output disturbance field.

$$\frac{\partial u}{\partial n} = 0 \quad \text{on } \partial\Omega_1,\ \partial\Omega_2,\ \partial\Omega_4, \tag{4.2b}$$

$$\frac{\partial u}{\partial n} = u_N \quad \text{on } \partial\Omega_3. \tag{4.2c}$$

Note that the Neumann boundary condition imposed on $\partial\Omega$ is non-zero only on the top edge. Throughout the experiments presented here u_N is fixed at $1_{\{0.35\leq x\leq 0.65\}}$. The wavespeed field $c(x)$ is assumed to be

$$c(x) = 20 + \tanh(\tilde{c}(x)),$$

where $\tilde{c}$ is a centred Gaussian

$$\tilde{c} \sim \mathbb{N}(0, \mathsf{C}) \quad \text{and} \quad \mathsf{C} = (-\Delta + \tau^2)^{-d};$$

here $-\Delta$ denotes the Laplacian on D_u subject to homogeneous Neumann boundary conditions on the space of spatial-mean zero functions, and we choose $d = 2$ and $\tau = 3$, the choice of d then leads to up to 1 fractional derivative for samples from this measure. We are interested in the map from the wavespeed field c to the solution u. Eq. (4.2) is solved using a finite element method on a 100×100 grid. See Figure 4.4 for a visualization of sample input (c) and resultant output u fields.

Results. Figure 4.5 shows the input, true output, neural network-predicted output, and output errors for the inputs resulting in the median and largest test errors (left and right panels) for each network architecture. For the median error cases, all neural networks tested yield disturbance predictions that match the true disturbance field in the 'eyeball norm'. However, differences in their behavior are revealed by the structure of the error fields: the error fields of PCA-Net and FNO have similar structures, length scales, and error magnitudes, indicating that their output spaces are similar. In contrast DeepONet and PARA-Net exhibit error fields with smaller length scales and different structures.

For the worst-case error cases, we note that all four networks achieve the worst-case error on the same test input c. This is because for this particular c, the frequency ω is

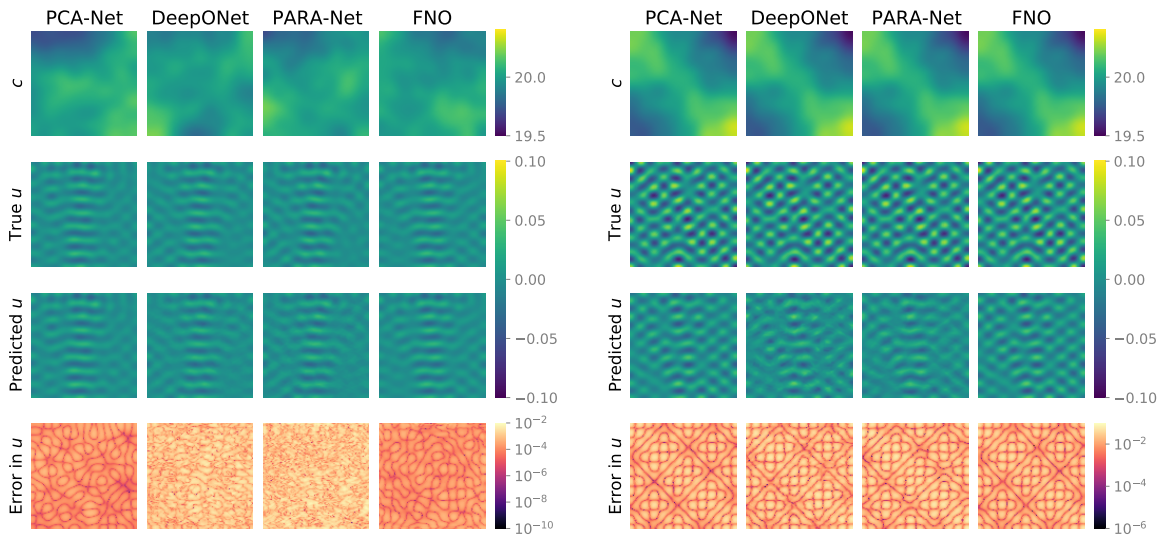

Figure 4.5: Helmholtz test problem: learned model predictions for inputs resulting in **median (left)** and **largest (right)** test errors for networks of size $w = 128$ / $d_f = 16$ trained on $N = 10000$ data.

close to an eigenfrequency of the Helmholtz operator, for which the $c \mapsto u$ map would not be uniquely defined. In Figure 4.6, we plot the normalized differences between the network predictions and the true solution, as well as the normalized eigenmode corresponding to the smallest eigenvalue of the Helmholtz operator for this input (the last row of the right panel of Figure 4.5 corresponds to the absolute value of the left four panels of Figure 4.6 plotted on a log scale). Note that these differences are close to scalar multiples of the eigenmode. A sample, c, drawn from the distribution for testing clearly can result in an eigenfrequency close the frequency already chosen for the simulations. The solution becomes large near eigenfrequencies, as is quantified, for example, in [119, Section 2.1]; hence, the accuracy of any prediction deteriorates accordingly independent of the architecture. This effect is visible in Figure 4.5: note that the left and right panels share a color scale and that the true u for the median error has more washed out colors than the true u for the largest error.

4.2.3 Structural mechanics equation

Formulation. The governing equation of an elastic solid undergoing infinitesimal deformations is

$$\begin{aligned} \nabla \cdot \sigma &= 0 \quad \text{in } \Omega, \\ u &= \bar{u} \quad \text{on } \Gamma_u, \\ \sigma \cdot n &= \bar{t} \quad \text{on } \Gamma_t, \end{aligned} \tag{4.3}$$

where u is the displacement vector and σ is the (Cauchy) stress tensor; Ω denotes the computational domain. The prescribed displacement $\bar{u}$ and the surface traction $\bar{t}$ respec-

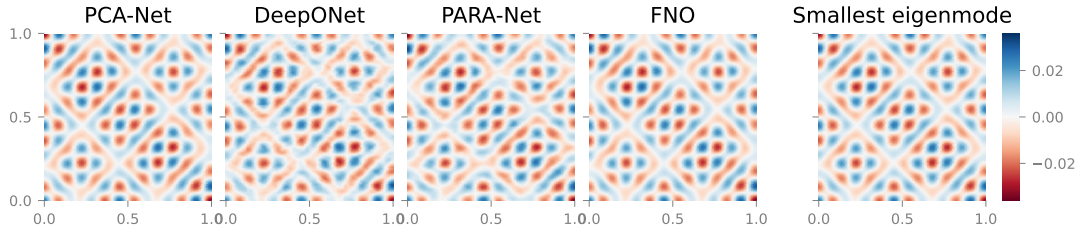

Figure 4.6: Helmholtz test problem largest test error case: normalized differences between learned model predictions and true solution (left four panels) and comparison to normalized smallest eigenmode of the Helmholtz operator for this test case.

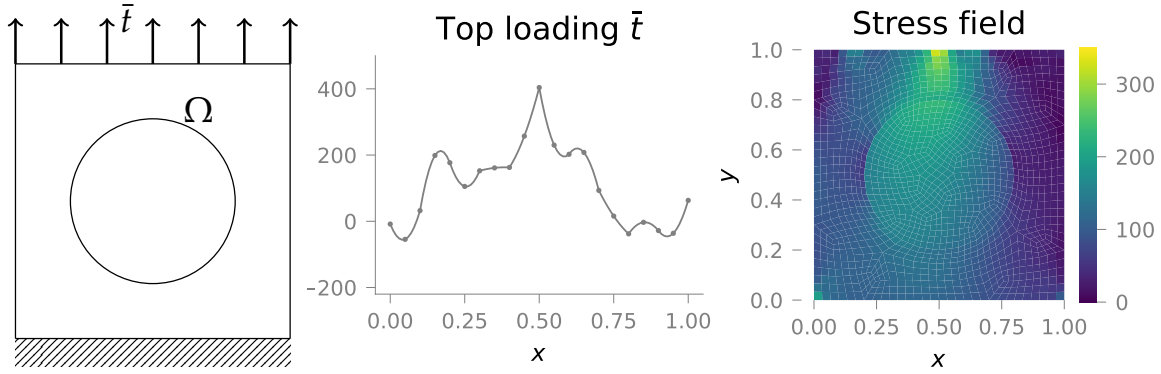

Figure 4.7: Solid mechanics test problem. Schematic of fiber-reinforced material and loading (**left**), sample load drawn from Gaussian distribution (**center**), and corresponding von Mises stress field (**right**).

tively, are imposed on the domain boundaries Γ_u and Γ_t respectively, with the outward unit normal n, where $\Gamma_u \cap \Gamma_t = \emptyset$ and $\Gamma_u \cup \Gamma_t = \partial\Omega$. The unit cell problem is depicted in Figure 4.7, which is clamped on the bottom edges. Tension traction is applied on the top edge, and the distributed load is $\bar{t}$. To solve for the displacement u from Eq. (4.3), we also need the constitutive model, which maps the deformation gradient to the stress. The matrix is made of incompressible Rivlin-Saunders material [120] with density $\rho = 0.8$ and energy density function parameters $C_1 = 1.863 \times 10^5$, $C_1 = 9.79 \times 10^3$ and the cylindrical fiber at the center is made of linear elastic material with density $\rho = 3.2$, Young's modulus $E = 4 \times 10^6$ and Poisson ratio $\nu = 0.35$.

We are interested in the map from the one-dimensional load $\bar{t}$ to the von Mises stress field τ_{vM} on the two dimensional domain Ω. The load $\bar{t}$ is drawn from a Gaussian random field with mean 100 and covariance $400^2\mathsf{C}$ with

$$\bar{t} \sim \mathbb{N}\left(100, 400^2\mathsf{C}\right) \quad \text{and} \quad \mathsf{C} = (-\Delta + \tau^2)^{-d}.$$

Here $-\Delta$ denotes the Laplacian on D subject to homogeneous Neumann boundary conditions on the space of spatial-mean zero functions, $\tau = 3$ denotes the inverse length scale of the random field and $d = 1$ determines its regularity (upto $1/2$ a fractional derivative for

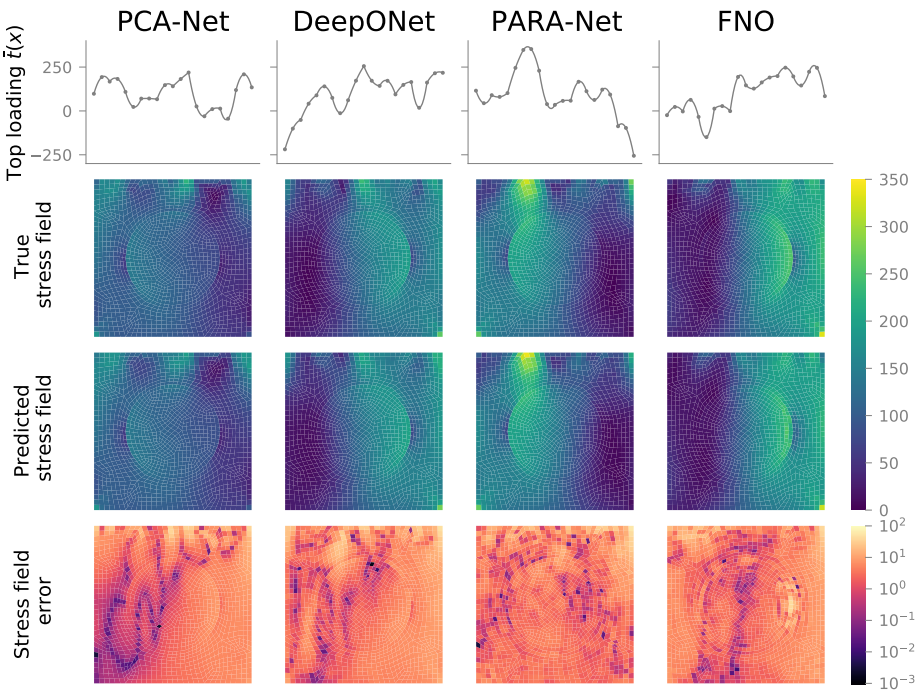

Figure 4.8: Structural mechanics test problem: learned model stress field predictions for inputs resulting in **median** test errors for networks of size $w = 128$ / $d_f = 16$ trained on $N = 10000$ data.

samples from this measure). The data is generated using the `NNFEM` library [121,122]. The mesh consists of 189 quadratic quadrilateral elements and the top edge is discretized by 10 quadratic elements and hence 21 points. The inputs are the load on the 21 points, and the outputs are the stress field (see Figure 4.7) on Gaussian quadrature points (9×189). Since FNO operates on uniform grids and requires the input and output data have the same dimensions, the stress field is interpolated on a 41×41 grid via radial basis function interpolation, and the load is interpolated on a 41 grid and extruded in the y direction. We note that in generating the input and output data for this problem, the discretized input data for the load $\bar{t}$ are drawn from a Gaussian, but because the finite element solver employs quadratic elements the effective load in the solver is smoother (as depicted in Figure 4.7).

Results. Figures 4.8 and 4.9 show the input, true output, neural network-predicted output, and output errors for inputs resulting in the median and largest test errors, respectively, for each network architecture. For this problem with discontinuous outputs, all four neural networks yield qualitatively similar predictions on both their median and worst-case error inputs, as seen in the similar color scaling of the error field plots. We call attention to the stress jump at the material interface which is generally well-captured by all four networks. We note the FNO results have oscillations at the material interface due to interpolation on the uniform grid.

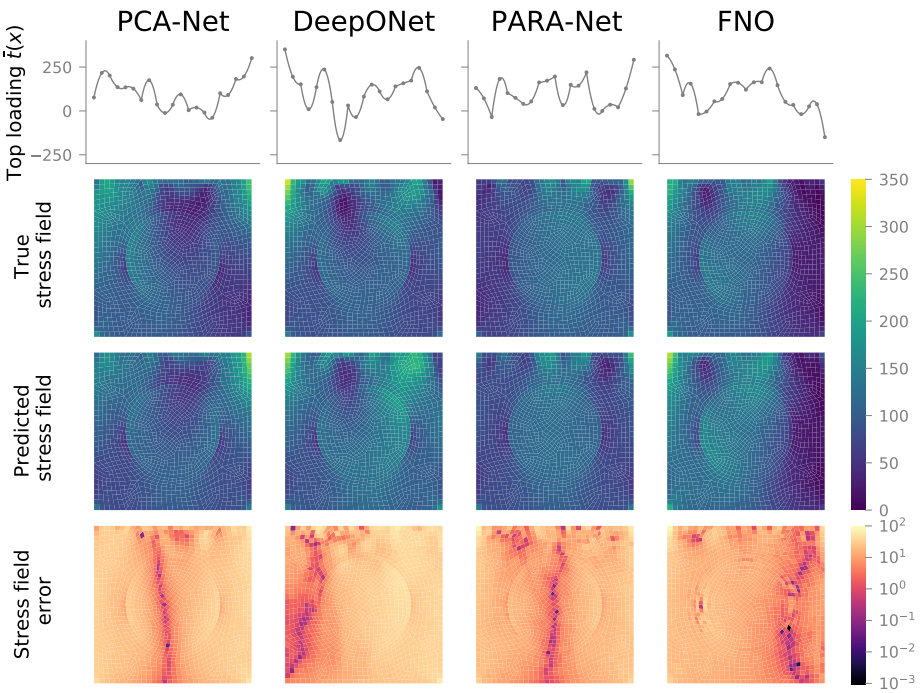

Figure 4.9: Structural mechanics test problem: learned model stress field predictions for inputs resulting in **largest** test errors for networks of size $w = 128$ / $d_f = 16$ trained on $N = 10000$ data.

4.2.4 Advection equation

Formulation. The 1D advection equation in $\Omega = [0, 1)$ is

$$\begin{aligned} &\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \quad x \in \Omega, \\ &u(0) = u_0, \end{aligned} \tag{4.4}$$

where $c = 1$ is the constant advection speed, and periodic boundary conditions are imposed. We are interested in the map from the initial u_0 to solution $u(\cdot, T)$ at $T = 0.5$. The initial condition u_0 is assumed to be

$$u_0 = -1 + 2\mathbb{1}\{\widetilde{u_0} \geq 0\},$$

where $\widetilde{u_0}$ a centered Gaussian

$$\widetilde{u_0} \sim \mathbb{N}(0, \mathsf{C}) \quad \text{and} \quad \mathsf{C} = (-\Delta + \tau^2)^{-d};$$

here $-\Delta$ denotes the Laplacian on D subject to periodic conditions on the space of spatial-mean zero functions, $\tau = 3$ denotes the inverse length scale of the random field and $d = 2$ determines the regularity of $\widetilde{u_0}$, which is upto $3/2$ derivatives. A pair of sample input and output data is depicted Figure 4.10. Note that multiple discontinuities exist in this input

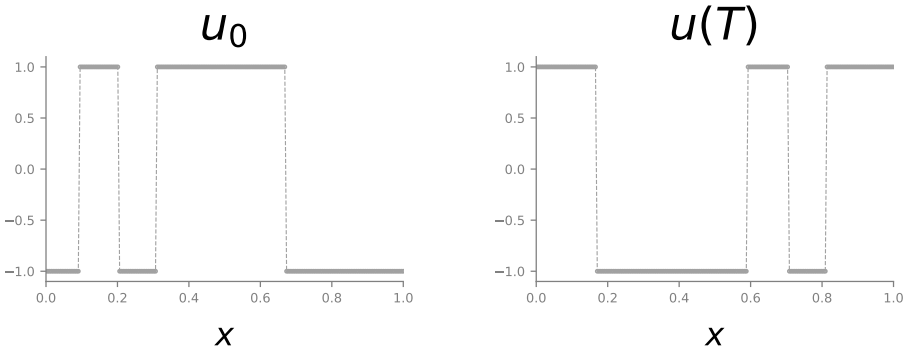

Figure 4.10: Sample input/output functions for the advection equation, **left:** u_0, **right:** $u(T)$.

sample; in general the probability of drawing a function with a given number p of discontinuities decreases with p but, in principle, draws with any number of discontinuities are possible.

Results. Eq. (4.4) is solved analytically on a 200-point uniform grid. Figure 4.11 shows the input, true output, and neural network-predicted output, for inputs resulting in the median and largest test errors (top and bottom panels) for each neural network architecture. We note that the median test cases for all four networks are similar; they each have just two discontinuities that are about half the domain apart. All four neural networks yield predictions which accurately reflect these discontinuities in the median case, although PCA-Net and DeepONet suffer from oscillatory Gibbs phenomena near the discontinuities. The worst-case test cases across all four network architectures also share similar characteristics: they have many discontinuities, including up-and-down jumps within a length scale of about one-tenth of the domain. In these challenging cases, the PCA-Net and DeepONet predictions suffer from Gibbs phenomena throughout the domain due to the many discontinuities. The FNO prediction reflects discontinuities at longer length scales as well and, additionally, suffers from significant overshooting near to small-length-scale discontinuities. Lastly, the PARA-Net worst-case prediction outputs a piecewise smooth solution that does not exhibit Gibbs phenomena but does not reflect the true discontinuities of the solution. It is important to note that all of PCA-Net, DeepONet and PARA-Net use the same input space representation, based on projection onto a finite number of PCA modes, and this is not well-adapted to discontinuous inputs. However DeepONet and PARA-Net have the possibility of recovering from this, since they learn the output space representation and it is in the output space that the error is measured.

4.3 Discussion of quantitative results

We now consider the complexity question as measured by test errors vs. cost for each of the neural networks. There are two main axes along which we can measure cost, training cost and online evaluation cost. For many PDE problems, the training cost is dominated by the

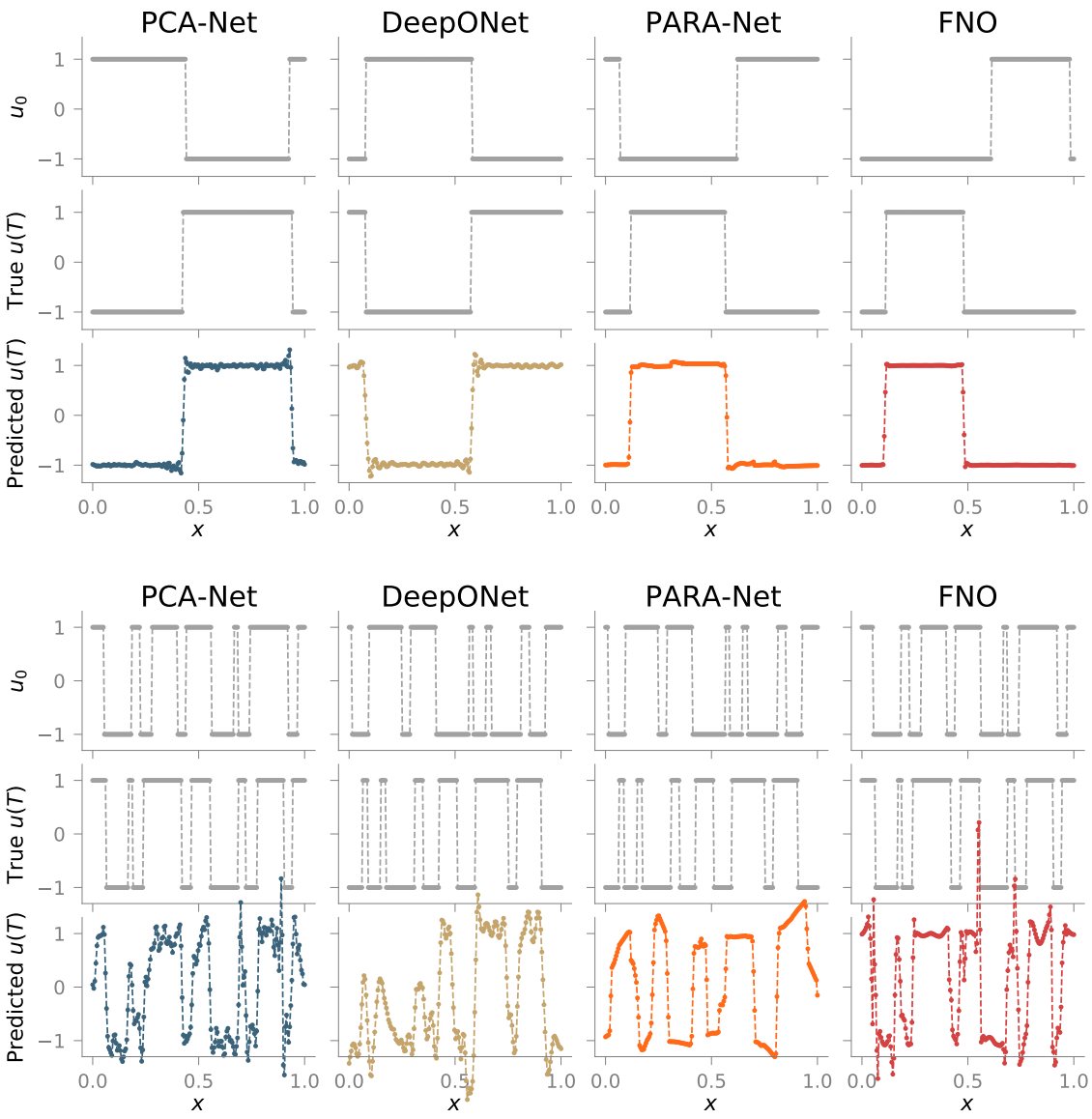

Figure 4.11: Advection test problem: learned model output predictions for inputs resulting in **median (top)** and **largest (bottom)** test errors for networks of size $w = 128$ / $d_f = 16$ trained on $N = 10000$ data.

cost of generating data by numerically solving the PDEs; we thus measure training cost in terms of available training data volume. In our experimental design, the online evaluation cost is directly related to the network size. The network size and training data volume have coupled influence on the network accuracy. The test error can thus be viewed as a surface in three-dimensional space where the two independent variables are the training volume and network size. In Section 4.3.1, we consider slices of this surface along the data

volume axis: we report and discuss the error-vs-training cost curves for each network and test problem at different network sizes. Then, in Section 4.3.2, we consider slices of the error in the other direction, along the network size axis, and report and discuss error-vs-size cost curves for each of the networks and test problems. Section 4.3.3 reports and discusses the error-vs-online evaluation cost curves. Section 4.3.4 provides a preliminary study of out-of-distribution generalization error. Section 4.3.5 compares the output space representations learned by PCA-Net and DeepONet. Finally, in Section 4.3.6 we comment on the influence of optimizer performance on the neural network prediction errors.

4.3.1 Accuracy vs. training cost

We now begin our discussion of the cost-accuracy trade-off by focusing on studying the test error as a function of the training cost as measured by the used training data volume. Figure 4.12 plots the test error vs. training data volume for each of our test problems across different network sizes and architectures, and plots the Monte Carlo rate $\mathcal{O}(N^{-\frac{1}{2}})$ [123] for reference.

We contrast the error-training data behavior of the two smooth problems (Navier-Stokes and Helmholtz, with that of the two problems with discontinuous outputs (the structural mechanics and advection problems. The paper [124] shows that for neural network learning of linear operators the Monte Carlo rate is obtained when the problem is smooth enough, but that for less smooth problems the reduction with respect to N is slower than the Monte Carlo rate. This theoretical result is also reflected in our numerical findings (which, with the exception of the advection problem, all concern nonlinear operators) where the smooth problems exhibit empirical error-data curves with slopes close to the $\mathcal{O}(N^{-\frac{1}{2}})$ rate for all but the smallest neural networks. For these small neural networks, the expressive power of the networks limits the convergence of error with respect to the training data. In contrast, for our non-smooth problems, the error-data curves generally exhibit slopes that are worse than the $\mathcal{O}(N^{-\frac{1}{2}})$ rate, and errors level out for all neural network sizes tested, indicating that the expressive power of the networks is more limiting for these non-smooth cases.

Finally, we note that PARA-Net differs from the other network types tested in that it often yields much higher test errors for low training data volumes than the other networks: this indicates that PARA-Net requires greater volumes of training data to yield good predictions. PCA-Net and DeepONet have similar error-training data curves on all problems except the Helmholtz problem, where the DeepONet trunk struggles to capture the high-frequency oscillations of the solution; we will discuss this in Section 4.3.5. FNO generally yields the lowest errors for a given training data volume and for the advection test problem actually appears to need even less data than the smallest data volume tested.

4.3.2 Accuracy vs. network size/expressivity

We next consider the error behavior of our neural network predictions as a function of the network width/number of channels, which is related both to the online evaluation cost of the network and to the network's expressive power. We focus initially here on

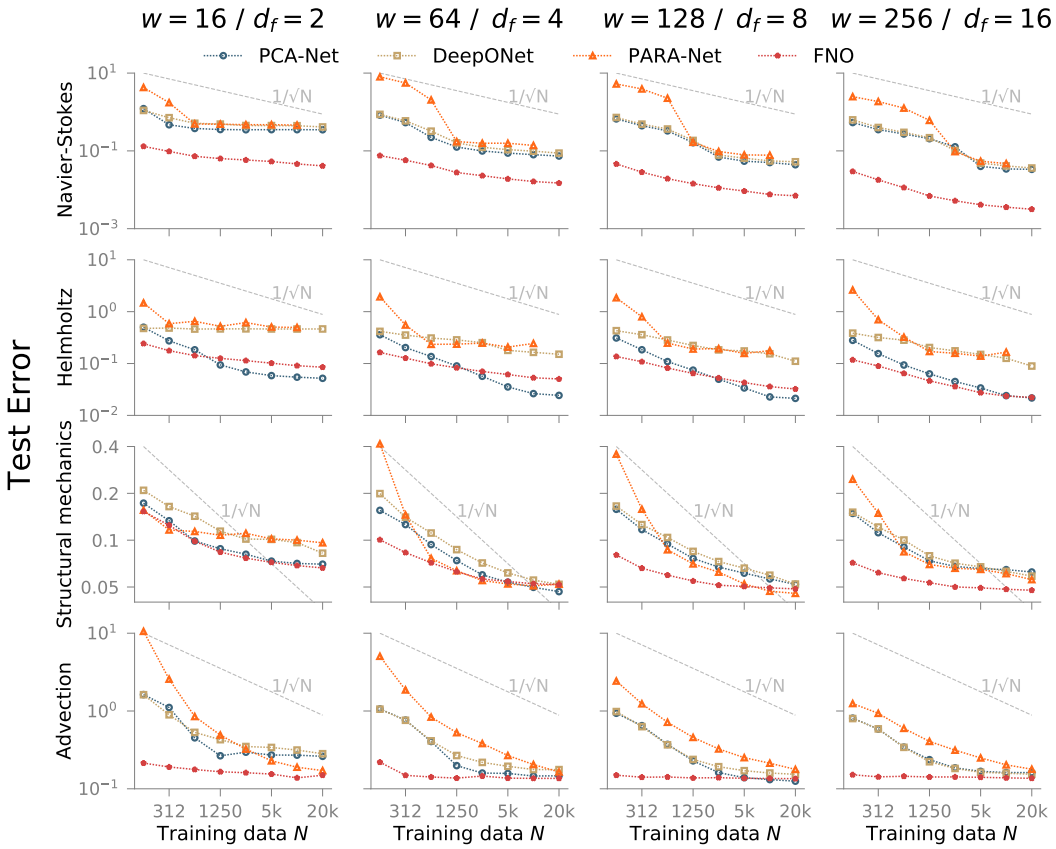

Figure 4.12: Test error vs. training data amount N for Navier-Stokes, Helmholtz, structural mechanical, and advection problems (top to bottom). Network size increases left to right.

the trade-off between accuracy (as measured by test error) and expressivity (as measured by number of parameters) and defer discussion of the accuracy-online evaluation cost tradeoff to the next section. Figure 4.13 plots the test error vs. network width for each problem and network type.

For the Navier-Stokes test problem, we note that PCA-Net, DeepONet, and PARA-Net all demonstrate overfitting[2] behavior—that is, error curves that increase with the size of the network—for small training data volumes. However, FNO does not exhibit overfitting behavior for the Navier-Stokes problem, reflecting that FNO's spectral representation of the solution is particularly suited to this problem. For the Helmholtz test problem, none of the network types exhibit overfitting behavior at any training data volume tested, indicating that $N = 2500$ data are sufficient for training the networks for this specific problem.

For our structural mechanics test problem, PCA-Net, DeepONet, and PARA-Net all exhibit overfitting behavior at all training data volumes, whereas FNO exhibits overfitting behavior only at the smaller data volumes. For the advection test problem, both PCA-Net

[2]The overfitting regime is one where the error grows as the network grows.

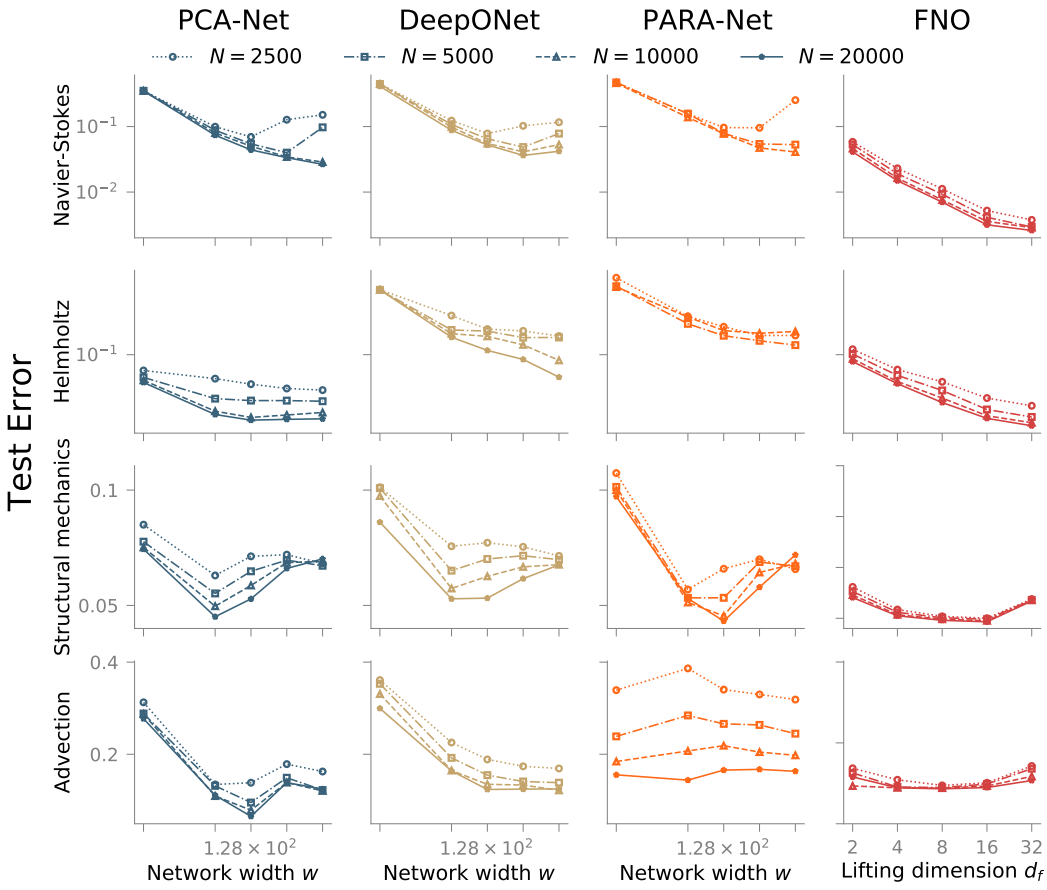

Figure 4.13: Test error vs. network size, as measured by internal layer width w for PCA-Net, DeepONet, and PARA-Net, and as measured by number of channels d_f for FNO. Different lines correspond to different training data volumes, N.

and FNO exhibit slight overfitting behavior at all tested data volumes, whereas DeepONet and PARA-Net appear more robust to this behavior. This may be because DeepONet and PARA-Net both define the output space in terms of a learned neural network, in contrast to PCA-Net and (our specific implementation of) FNO, for which the output space is the result of a linear decomposition or a prescribed set of Fourier bases.

4.3.3 Accuracy vs. evaluation cost

Finally, we directly consider the error behavior of our neural network predictions as a function of their online evaluation cost (Table 4.1). Figure 4.14 plots the test error vs. network evaluation cost for each network type and test problem across the range of network sizes tested, for a fixed training data volume of $N = 10000$. Figure C.1 contains the complete results for all training data volumes tested; however our main conclusions can be

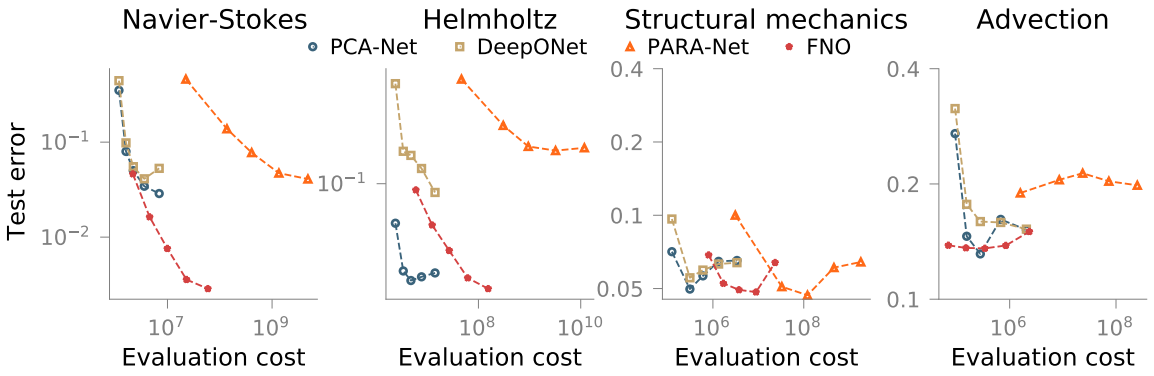

Figure 4.14: Test error vs. evaluation cost for all four test problems for fixed training data volume $N = 10000$. See Figure C.1 for error vs. evaluation cost curves for all training data volumes tested.

understood from just the $N = 10000$ results.

We note first that for all test problems, the cost of PARA-Net exceeds the cost of other networks by at least an order of magnitude with similar or worse error than the other networks. This is because the network must be evaluated at every spatial point in the discretization of the output function. As such it is not competitive with the other network architectures in the setting where entire output functions are desired; it could however be considered an alternative when outputs at only a few spatial points are desired.

Because our implementation of DeepONet uses a branch network identical to PCA-Net, their evaluation costs are the same for a given network width. Across our test problems PCA-Net generally yields errors similar to those of DeepONet, except for the Helmholtz test problem where the DeepONet trunk struggles to represent the high-frequency features of the solutions that the PCA basis captures. This motivates the use of PCA basis functions as the output space in DeepONet as introduced in [13].

Finally, our implementation of FNO generally achieves the lowest test errors across all four problems. For the one-dimensional advection equation, the evaluation cost of FNO is similar to that of PCA-Net and DeepONet; however for our two-dimensional examples, the evaluation cost of FNO is greater than that of PCA-Net and DeepONet. This is because the cost of FNO depends on the number of Fourier modes $k_{\max}$, which scales exponentially with the spatial dimension.

Since the cost-accuracy curves for FNO and for PCA-Net/DeepONet occupy different regions of the cost axis we cannot make definitive conclusions about the relative merits of these three methods. However, since FNO has the clearest signal of error decay as a function of cost, we compute the empirical power law

$$\text{test error} = a(\text{evaluation cost})^{-p}$$

of the FNO. The exponents p are 1.129, 0.7254, 0.0926, and 0.002506 for Navier-Stokes, Helmholtz, Structural mechanics and advection problems, respectively; the differences are presumed to relate to the regularities of the output spaces for these problems.

4.3.4 Out-of-distribution (OOD) generalization

Thus far, our numerical studies have exclusively focused on the ability of the neural networks to predict outputs for test inputs drawn from the same distribution as the training data set. In scientific and engineering applications, it is often desirable for a model to be able to accurately predict outputs for inputs outside of the distribution seen in training. We emphasize that the ability of the learned models to generalize to out-of-distribution (OOD) data will be highly dependent on both the problem and the distribution of the unseen data. The overall efficiency of a given surrogate model will depend on its ability to generalize since this will govern the extent to which the cost of training can be amortized. Here we provide an initial empirical study of the generalization error to OOD inputs, using the Navier-Stokes and structural mechanics test problems. There is some limited analysis of OOD test error in the linear setting [124], and our numerical experiments in the nonlinear setting add to what is known about this issue.

In both the Navier-Stokes and structural mechanics problems, training input data are drawn from a Gaussian random field with covariance C as described in Sections 4.2.1 and 4.2.3). To study OOD generalization error, we draw test inputs from a new Gaussian random field with covariance $C' = 4C$, such that the OOD inputs are roughly twice the size of the training inputs. Figure 4.15 plots the test error vs. training data volume for the largest network sizes tested, with $w = 256$ and $d_f = 16$. As expected, test errors for OOD input data are generally higher than test errors for in-distribution input data. We note that for the smooth Navier-Stokes problem, lower training data volumes lead to higher errors, but the gap between the in-distribution and OOD performance is smaller in these cases. In contrast, for the non-smooth structural mechanics problem, the gap in generalization performance remains large even for smaller data volumes. We finally note that while increasing training data volume improves OOD test error, the convergence rate of the OOD test error with respect to the training data volume is worse than the rate for the in-distribution test error.

4.3.5 PCA-Net vs. DeepONet output spaces

Both PCA-Net and DeepONet construct basis functions to represent the output functions, where PCA-Net constructs PCA bases from data directly and DeepONet learns bases from data with the trunk net. It is instructive to compare DeepONet with PCA-Net from the perspective of the output space. Figures 4.16 and 4.17 visualize the basis functions used to represent the output functions in PCA-Net and DeepONet for the four test problems we consider. For all test problems, we note that a large number (40%-70%) of the DeepONet basis functions after optimization are exactly zero everywhere, despite nonzero random initialization. This may be due to our use of the ReLU activation function, and other activation functions may yield different basis functions. However, our use of ReLU is consistent with the experiments reported in [13]. In our visualizations, we overrepresent the non-zero basis functions since these define the output space. Across all examples, we note that the learned DeepONet basis functions after training tend to be local functions, in contrast to the global functions that result when employing in PCA-Net. We therefore also compute and visualize the PCA modes of the DeepONet bases.

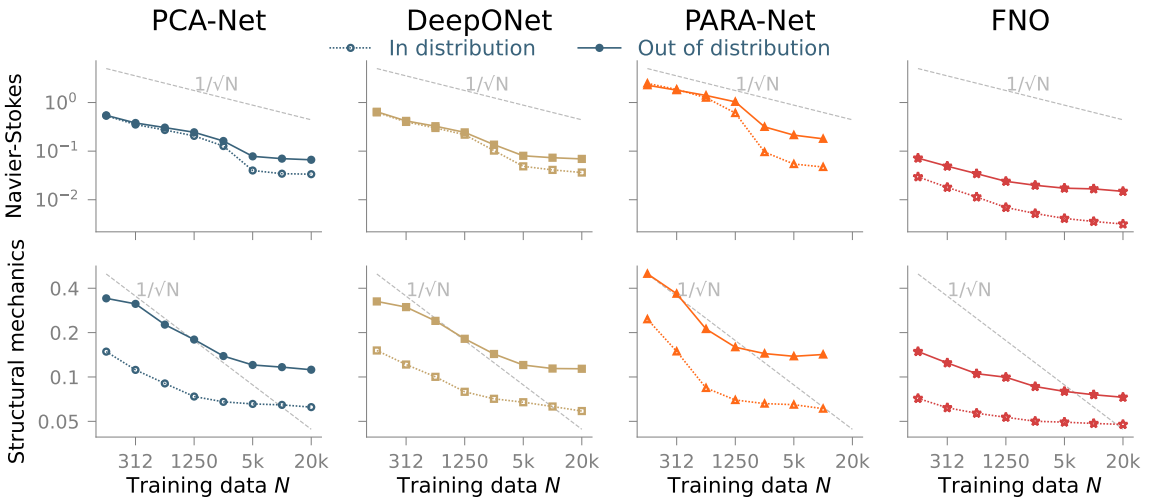

Figure 4.15: In- vs. out-of-distribution test error comparison for Navier-Stokes (**top**) and structural mechanics (**bottom**) test problems for largest networks tested ($w = 256$, $d_f = 16$).

For the Navier-Stokes problem, PCA-Net and DeepONet achieve similar errors; we attribute this to the fact that the empirical PCA basis of the trained DeepONet trunk functions is similar to that of the true PCA basis of the measure $(\Psi^\dagger)^\sharp \mu$. However, for the Helmholtz test problem, the PCA basis is able to represent the high-frequency oscillatory nature of the solution, but the empirical PCA basis learned by DeepONet trunk net is not (Figure 4.16). This explains why PCA-Net achieves lower errors than DeepONet for this problem. This is interesting because in principle if the DeepONet trunk network could learn the PCA basis functions, then DeepONet should be able to achieve similar performance to that of PCA-Net. This raises the question of whether the failure of the trunk to learn the PCA basis functions is due to a lack of expressivity of the trunk or due to optimization error. In our implementation, the DeepONet trunk network only has two hidden layers. However, increasing the number of hidden layers to three actually worsened results in our tests and thus we report the results for two hidden layers. Thus it may be the case that optimization error plays a larger role in the DeepONet error for the Helmholtz problem than for the other three problems.

4.3.6 Role of optimization

We first offer some comments on the performance of the stochastic gradient optimizer employed in training the neural networks. Figure 4.18 plots the test error vs. training error for all four test problems and all four neural network architectures. The different line styles correspond to different training data volumes. The points on each line from left to right correspond to increasing network sizes. We view the comparison between test and training errors as an indicator of the performance of the optimization.

In all cases, test errors are larger than or approximately equal to training errors. Test

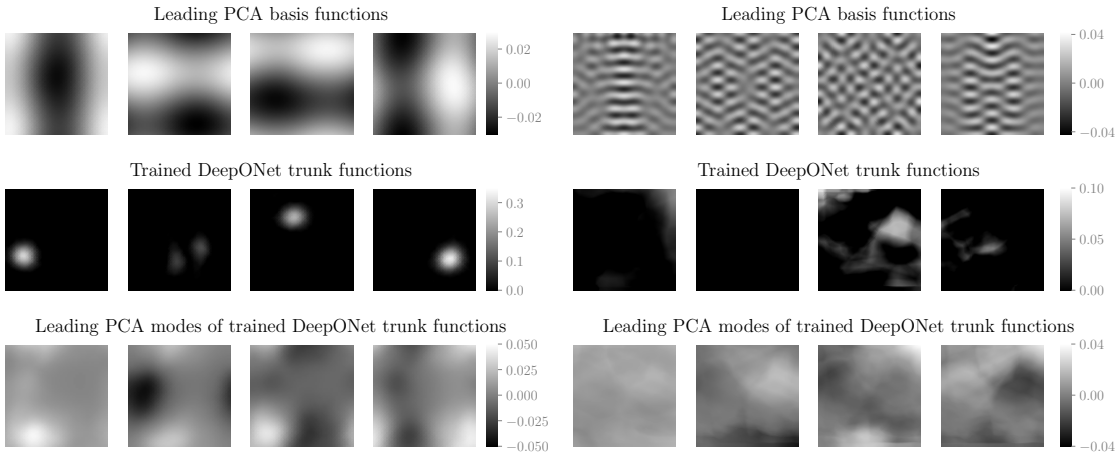

Figure 4.16: Navier-Stokes (**left**) and Helmholtz (**right**) test problems: Comparison of bases used to represent output functions in PCA-Net and DeepONet. Row 1: leading PCA basis functions of output training data. Row 2: Example DeepONet trunk functions after training. Row 3: Leading PCA modes of DeepONet trunk functions after training.

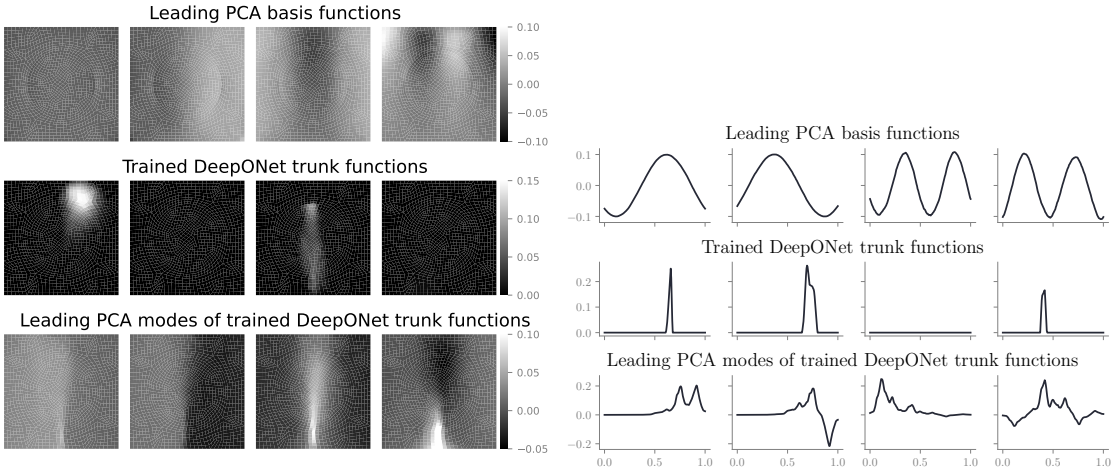

Figure 4.17: Structural mechanics (**left**) and advection (**right**) test problems: Comparison of bases used to represent output functions in PCA-Net and DeepONet. Row 1: leading PCA basis functions of output training data. Row 2: Example DeepONet trunk functions after training. Row 3: Leading PCA modes of DeepONet trunk functions after training.

errors that are noticeably larger than the training error (above the dashed gray lines) are an indication that the optimization has overfit to the training data. This effect is noticeable for the smaller training data volumes and for larger networks, which require more data to train effectively. When test errors are approximately the same as training errors (lying close to the dashed gray lines), the optimization has avoided overfitting to data. Given sufficient training data volume, all four network types avoid overfitting in the smooth

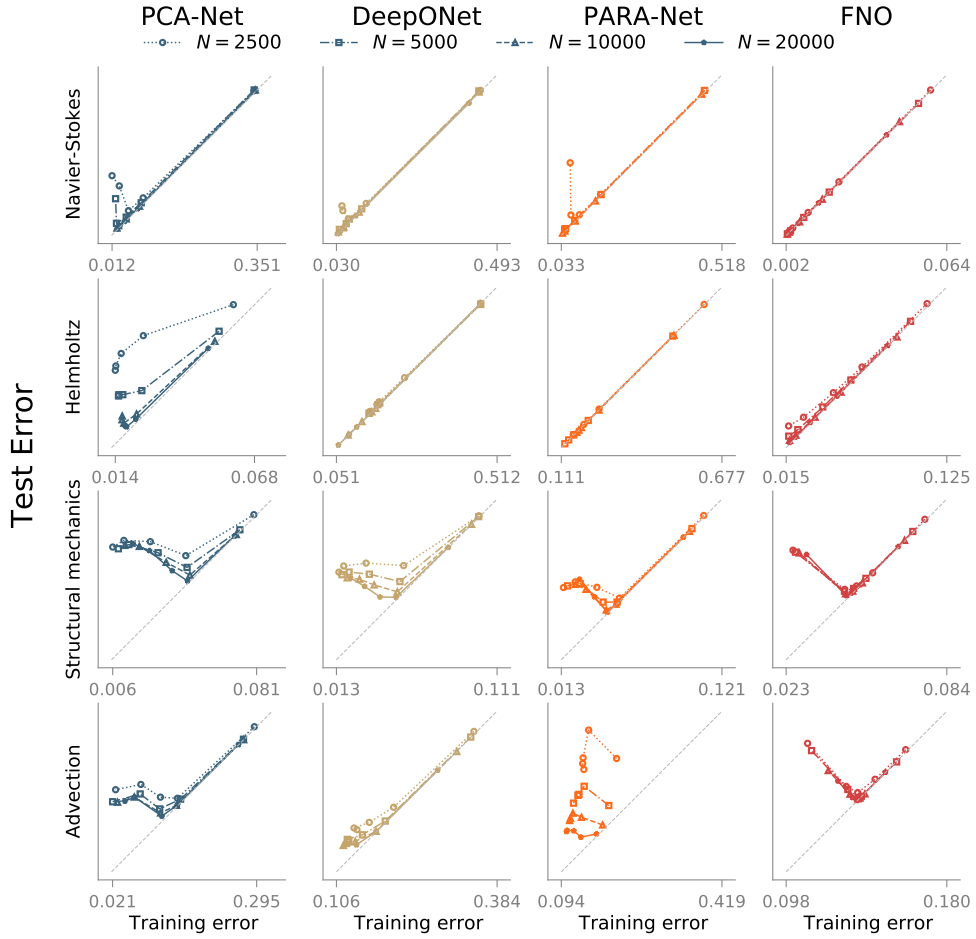

Figure 4.18: Test error vs. training error for all four test problems and all four neural network architectures. Different linestyles correspond to different training data volumes, N. The general increase in training error from left to right corresponds with increasing network size as measured by layer width w for PCA-Net, DeepONet, and PARA-Net, and number of channels d_f for FNO. Gray dashed lines have slope 1 and intercept 0.

test problems (Navier-Stokes and Helmholtz), whereas they generally suffer more from overfitting on the discontinuous test problems (structural mechanics and advection). We note that PARA-Net has particularly bad generalization performance on the advection problem, both compared to its performance on other problems and the performance of other network types on the advection problem. We also note that DeepONet is the most resistant to overfitting on the advection problem, compared to the other network types.

Finally we note that the training error is non-zero in all cases, indicating an imperfect fit of the neural networks to the training data. This raises questions of whether different optimization choices would be able to drive the training error lower. Such questions are worthy of study and the subject of future work. We note however that in the cases

described above where the neural network has overfit to the training data, better optimization results (in the sense of lower training losses) may not correspond to improved generalization performance.

5 Conclusions

We have presented a numerical study of the performance of four different neural network architectures for modeling operators (function-to-function maps) in problems that involve the solution of a PDE. In particular, we compare test errors for the four networks across a range of network sizes and training data volumes. Our results show that PARA-Net is not a competitive approach in this setting. Our results also suggest that for 1D and 2D problems in simple geometries FNO may be the best approach in terms of the cost-accuracy tradeoff, achieving the lowest errors and low-to-intermediate cost. However, FNO's cost does scale poorly with spatial dimension and its cost may be prohibitive in the $\geq$3D setting, where DeepONet and PCA-Net may be preferable. It is possible that investment in efficient software implementations, perhaps taking advantage of parallelization, may make FNO tractable in 3D settings [125]. Special treatments of FNO for more complex geometries, a class of problems not considered here, are introduced in [13, 100]; without, and maybe even with, such special treatments it appears that the additional flexibility of DeepONet gives it some advantages in such settings; a careful complexity study would be of value to establish this.

Our comparison of DeepONet and PCA-Net fixes the PCA-Net architecture to be the same as that of the DeepONet branch network, so that the key difference between the approaches lies in the functions used to represent the output space. In our comparison, PCA-Net generally yields lower or similar errors to those of DeepONet; investigation into the output functions learned by DeepONet shows that DeepONet does not always succeed at learning a good basis in which to express the output; for the Helmholtz equation the training process drives the output basis functions to highly localized bases not representative of the output space. This result suggests the value of including PCA basis functions in the output representation of DeepONet for some problems.

In summary, we provide some insights into the relative merits of different operator surrogates, and mechanisms underlying observed behaviors; whilst we do not reach definitive conclusions about the relative merits of PCA-Net, DeepONet and FNO, and indeed we would expect any such conclusions to be problem dependent, our numerical results highlight the need for, and can guide, future rigorous analyses of the complexity (cost-accuracy trade-off) for these methods. We have focused on computational studies of error as a function of number of parameters in the network (and hence cost of evaluating the operator surrogate) and as a function of data volume (and hence cost of data acquisition). There is some existing research that relates to these questions, and of course much more is required.

- Regarding error as a function of number of parameters, there is profound work for both DeepONet [126] and FNO [113] demonstrating the possibility of beating the curse of dimensionality on some PDE-based operator learning problems; in these

works this is interpreted as sub-exponential scaling of number of parameters with inverse error. For holomorphic maps arising in parametric elliptic PDEs there is a body of literature establishing that certain networks (different from those considered in this paper) can beat the curse of dimensionality [35, 127, 128]. It is important to appreciate that these results simply concern the *existence* of neural networks with stated properties; they do not address how to find them. Extending the work in these five papers to a broader range of problems, and addressing the issue of finding the desired parameter sets, remain open and challenging problems. The numerical experiments reported in this paper can guide such theory.

- Regarding error as a function of data volume, the sample complexity of the problem, there is recent theoretical work in the setting of learning linear operators [124] and it would be desirable, though challenging, to extend this work to nonlinear operator learning problems. The numerical experiments reports in this paper can also guide such theory.
- The issue of linking parameterization to data volume, to optimize measures of complexity, is not studied theoretically, to the best of our knowledge. Again the experiments in this paper suggest the need for theoretical guidance on this issue.
- Questions concerning distribution of parameters through complex network architectures are not studied in any detail, with the exception of studies of depth versus width in neural networks [129]. It will be important to develop further theory in this area. For example, for the FNO, how should parameter distribution between number of channels and number of Fourier modes be chosen? For DeepONet how should parameters be distributed between trunk and branch networks? These are hard questions with answers that will be consequential for the choice and implementation of operator learning surrogates.

Acknowledgments

MVdH was supported by U.S. Department of Energy, Office of Basic Energy Sciences, Chemical Sciences, Geosciences and Biosciences Division under grant number DE-SC0020345 and the Simons Foundation under the MATH + X program, and the corporate members of the Geo-Mathematical Imaging Group at Rice University. DZH was supported by the generosity of Eric and Wendy Schmidt by recommendation of the Schmidt Futures program. EQ was supported by Caltech's von Karman postdoctoral instructorship and partially supported by the Simons Foundation Award No. 663281 granted to the Institute of Mathematics of the Polish Academy of Sciences for the years 2021-2023. AMS was supported by the Office of Naval Research (award N00014-17-1-2079) and by the Air Force Office of Scientific Research (MURI award number FA9550-20-1-0358 – Machine Learning and Physics-Based Modeling and Simulation). The authors thank Mike Kirby, Shibo Li, Zongyi Li, Lu Lu, Michael Penwarden, Shandian Zhe, and Nicholas Nelsen for helpful comments on an earlier draft.

A Pointwise inputs to DeepONet

We accommodate the case of pointwise evaluations by writing

$$L_k u = \langle u(x_\ell), a_m \rangle_{\mathbb{R}^{d_i}}$$

to ensure a collection of real-valued linear functionals on $\mathcal{U}$. Note that k is doubly-indexed: $k = (\ell, m)$ is a multi-index over a set with cardinality defining d_u; the $\{a_m\}_{m=1}^{d_i}$ are canonical unit vectors in $\mathbb{R}^{d_i}$, in order to pick-out real-valued functionals, and the $\{x_\ell\}_{\ell=1}^{\ell'}$ denote the locations of the pointwise evaluations. Thus $d_u = d_i \times \ell'$. To unify the notation with the PCA input case we may then (abusing notation) relabel to index over $k \in \{1, \cdots, d_u\}$. In this setting it is simplest to think of $\mathcal{H} = L^2(D_u; \mathbb{R}^{d_i})$ and $\mathcal{U} = C(D_u; \mathbb{R}^{d_i})$; alternatively $\mathcal{U}$ may be a RKHS, such as a Sobolev space of fractional order greater than $d_x/2$, which is compactly embedded into $C(D_u; \mathbb{R}^{d_i})$.

B Complexity analysis

B.1 Network parameter complexity

Here we derive the number of parameters of each of the four neural network formulations as a function of the network input and output dimensions and of the network width w (or for FNO, the number of features d_f). Note that a standard fully-connected nonlinear or linear layer from $\mathbb{R}^n \to \mathbb{R}^m$ has nm weights and m biases (the 'linear' layer is technically an affine transformation).

PCA-Net consists of an initial nonlinear layer from $\mathbb{R}^{d_u} \to \mathbb{R}^w$, two internal nonlinear layers from $\mathbb{R}^w \to \mathbb{R}^w$, and a final linear layer from $\mathbb{R}^w \to \mathbb{R}^{d_v}$. The parameter complexity for PCA is thus $2w^2 + w(d_u + d_v) + 3w + d_v$. DeepONet has two networks, a branch and a trunk. The branch network has the same complexity as PCA-Net. The trunk network consists of an initial nonlinear layer from $\mathbb{R}^{d_y} \to \mathbb{R}^w$, two internal nonlinear layers from $\mathbb{R}^w \to \mathbb{R}^w$, and a final linear layer from $\mathbb{R}^w \to \mathbb{R}^{d_v d_o}$. The parameter complexity for DeepONet is thus $(2w^2 + w(d_u + d_v) + 3w + d_v) + (2w^2 + w(d_y + d_v d_o) + 3w + d_v d_o) = 4w^2 + w(d_u + d_v + d_y + d_v d_o) + 6w + d_v + d_v d_o$. PARA-Net has an initial nonlinear layer from $\mathbb{R}^{d_u + d_y} \to \mathbb{R}^w$, two internal nonlinear layers from $\mathbb{R}^w \to \mathbb{R}^w$, and a final linear layer from $\mathbb{R}^w \to \mathbb{R}^{d_o}$. The parameter complexity for PARA-Net is therefore $2w^2 + w(d_o + d_u + d_y) + 3w + d_o$. FNO has an initial lifting layer that lifts the input in $\mathbb{R}^{d_i}$ at each spatial point to channels in $\mathbb{R}^{d_f}$ at each spatial point. In our implementation, the lifting is a linear layer from $\mathbb{R}^{d_i} \to \mathbb{R}^{d_f}$. Similarly, the final projection layer of FNO is a pointwise linear layer from $\mathbb{R}^{d_f} \to \mathbb{R}^{d_o}$. There are three internal Fourier operators from $\mathbb{R}^{N_p d_f} \to \mathbb{R}^{N_p d_f}$. In each layer, there is a linear map from $\mathbb{R}^{d_f} \to \mathbb{R}^{d_f}$ that is applied pointwise with parameter complexity d_f^2. For each of the $k_{\max}$ Fourier modes, there are d_f^2 parameters in the linear transformation P_l. Each internal layer thus has parameter complexity $d_f^2 k_{\max}$. The total

Table B.1: Number of parameters in our implementations of the four neural network architectures for different network size parameters w or d_f, for each test problem considered.

	Navier-Stokes					Helmholtz				
					w / d_f					
Architecture	16/2	64/4	128/8	256/16	512/32	16/2	64/4	128/8	256/16	512/32
PCA-Net	5680	41152	131456	459520	1705472	4816	37696	124544	445696	1677824
DeepONet	7328	66176	230656	854528	3281920	6464	62720	223744	840704	3254272
PARA-Net	5264	33344	99456	329984	1184256	4400	29888	92544	316160	1156608
FNO	1747	6973	27865	111409	445537	1747	6973	27865	111409	445537
	Structural mechanics					Advection				
					w / d_f					
	16/2	64/4	128/8	256/16	512/32	16/2	64/4	128/8	256/16	512/32
PCA-Net	2256	27456	104064	404736	1595904	7984	50368	149888	496384	1779200
DeepONet	3904	52480	203264	799744	3172352	9632	75392	249088	891392	3355648
PARA-Net	1840	19648	72064	275200	1074688	7568	42560	117888	366848	1257984
FNO	1747	6973	27865	111409	445537	163	637	2521	10033	40033

parameter complexity for our implementation of FNO is therefore $d_f d_i + d_f + d_f d_o + d_o + 3d_f^2 k_{\max} + 3d_f^2$.

The total number of hyperparameters used in each network for each of the four test problems we consider are tabulated in Table B.1.

B.2 Network evaluation cost

Here we derive the evaluation cost of each of the four neural networks. A single standard nonlinear layer $\sigma(Ax + b)$ with $A \in \mathbb{R}^{n,m}$ has cost $2mn + n$, due to the matrix-vector product costing $(2m - 1)n$, the vector-vector sum costing n, and we approximate the activation function cost as n. A single standard linear layer costs $2mn$ since no activation function is applied. Projecting the solution $u \in \mathbb{R}^{N_p \times d_i}$ on d_u PCA bases has cost $d_u(2N_p d_i - 1)$ Recovering the solution $v \in \mathbb{R}^{N_p \times d_o}$ from d_v PCA coefficients has cost $(2d_v - 1)N_p d_o$.

PCA-Net has 3 internal layers, the cost is $d_u(2N_p d_i - 1) + 2d_u w + 4w^2 + 2d_v w + 3w + (2d_v - 1)N_p d_o$. The DeepONet with the precomputed trunk has the same cost as PCA-Net. PARA-Net has 3 internal layers. We need to evaluate it at N_p points, and hence its cost is $d_u(2N_p d_i - 1) + [2(d_u + d_y)w + 4w^2 + 2wd_o + 3w]N_p$. FNO has an initial lifting layer : $\mathbb{R}^{d_i} \to R^{d_f}$ with cost $2N_p d_f d_i$, a final projection layer: $\mathbb{R}^{d_f} \to R^{d_o}$ with cost $2N_p d_o d_f$, and 3 Fourier layers. The cost of the Fourier operator is approximated as $d_f 2 \times 5N_p log(N_p) + k_{\max}(2d_f^2 - d_f)$, the cost of σ is $d_f N_p$, and the cost of $W_l v(x)$ is $N_p(2d_f^2 - d_f)$. Hence the cost of FNO is $2N_p d_f d_i + 3(10 d_f N_p log(N_p) + k_{\max}(2d_f^2 - d_f) + d_f N_p + N_p(2d_f^2 - d_f)) + 2N_p d_o d_f$.

C Error vs. cost results

We report in Figure C.1 the test error vs. online evaluation cost results for all four test problems at all training data volumes tested. The results for each column are similar, and thus only the third column is extracted and reported in Figure 4.14 in the main text.

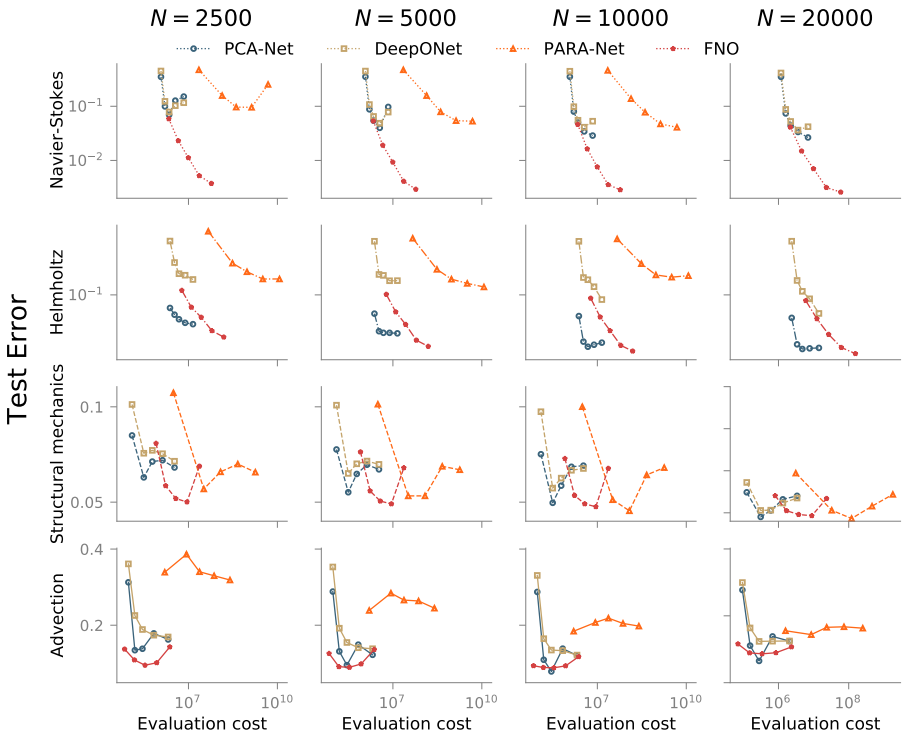

Figure C.1: Test error vs. evaluation cost for all training data volumes tested.

References

[1] Boško S. Jovanović and Endre Süli. *Analysis of Finite Difference Schemes: For Linear Partial Differential Equations with Generalized Solutions*, volume 46. Springer Science & Business Media, 2013.

[2] John C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. SIAM, 2004.

[3] Claes Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Courier Corporation, 2012.

[4] Claudio Canuto, M. Yousuff Hussaini, Alfio Quarteroni, A. Thomas Jr., et al. *Spectral Methods in Fluid Dynamics*. Springer Science & Business Media, 2012.

[5] Lloyd N. Trefethen. *Spectral Methods in MATLAB*. SIAM, 2000.

[6] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[7] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra*, volume 50. SIAM, 1997.

[8] Wolfgang Hackbusch. *Multi-Grid Methods and Applications*, volume 4. Springer Science & Business Media, 2013.

[9] Houman Owhadi and Clint Scovel. *Operator-Adapted Wavelets, Fast Solvers, and Numerical Homogenization: From a Game Theoretic Approach to Numerical Approximation and Algorithm Design*, volume 35. Cambridge University Press, 2019.

[10] Jan S. Hesthaven and Stefano Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018.

[11] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M. Stuart. Model reduction and neural networks for parametric PDEs. *The SMAI Journal of Computational Mathematics*, 7:121–157, 2021.

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

[13] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. *arXiv preprint arXiv:2111.05512*, 2021.

[14] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, et al. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2020.

[15] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *Journal of Machine Learning Research*, to appear, 2022.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[17] Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning*, pages 1225–1234. PMLR, 2016.

[18] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[19] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 1999.

[20] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[21] Eldad Haber, Felix Lucka, and Lars Ruthotto. Never look back–a modified EnKF method and its application to the training of neural networks without back propagation. *arXiv preprint arXiv:1805.08034*, 2018.

[22] Nikola B. Kovachki and Andrew M. Stuart. Ensemble kalman inversion: A derivative-free technique for machine learning tasks. *Inverse Problems*, 35(9):095005, 2019.

[23] Michael B. Giles. Multilevel Monte Carlo path simulation. *Operations Research*, 56(3):607–617, 2008.

[24] Michael B. Giles. Multilevel Monte Carlo methods. *Acta Numerica*, 24:259–328, 2015.

[25] Thomas D. Economon, Francisco Palacios, Sean R. Copeland, Trent W. Lukaczyk, and Juan J. Alonso. SU2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, 54(3):828–846, 2016.

[26] Joaquim R.R.A. Martins and Andrew B. Lambe. Multidisciplinary design optimization: A survey of architectures. *AIAA Journal*, 51(9):2049–2075, 2013.

[27] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: Theory, Methods, and Applications*. Springer Science & Business Media, 2003.

[28] Gabriele Boncoraglio and Charbel Farhat. Active manifold and model-order reduction to accelerate multidisciplinary analysis and optimization. *AIAA Journal*, 59(11):4739–4753, 2021.

[29] Andrew M. Stuart. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, 2010.

[30] James Martin, Lucas C. Wilcox, Carsten Burstedde, and Omar Ghattas. A stochastic Newton MCMC method for large-scale statistical inverse problems with application to seismic inversion. *SIAM Journal on Scientific Computing*, 34(3):A1460–A1487, 2012.

[31] Daniel Zhengyu Huang, Tapio Schneider, and Andrew M. Stuart. Iterated Kalman methodology for inverse problems. *Journal of Computational Physics*, 463:111262, 2022.

[32] Daniel Zhengyu Huang, Jiaoyang Huang, Sebastian Reich, and Andrew M. Stuart. Efficient derivative-free Bayesian inference for large-scale inverse problems. *arXiv preprint arXiv:2204.04386*, 2022.

[33] Matthias Morzfeld, Xin T. Tong, and Youssef M. Marzouk. Localization for MCMC: Sampling high-dimensional posterior distributions with local structure. *Journal of Computational Physics*, 380:1–28, 2019.

[34] Shunxiang Cao and Daniel Zhengyu Huang. Bayesian calibration for large-scale fluid structure interaction problems under embedded/immersed boundary framework. *arXiv preprint arXiv:2105.09497*, 2021.

[35] Lukas Herrmann, Christoph Schwab, and Jakob Zech. Deep ReLU neural network expression rates for Data-to-QoI maps in bayesian PDE inversion. *SAM Research Report*, 2020, 2020.

[36] Weinan E. *Principles of Multiscale Modeling*. Cambridge University Press, 2011.

[37] Jacob Fish, Kamlun Shek, Muralidharan Pandheeradi, and Mark S. Shephard. Computational plasticity for composite structures based on mathematical homogenization: Theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 148(1-2):53–73, 1997.

[38] Frédéric Feyel and Jean-Louis Chaboche. FE2 multiscale approach for modelling the elastoviscoplastic behaviour of long fibre SiC/Ti composite materials. *Computer Methods in Applied Mechanics and Engineering*, 183(3-4):309–330, 2000.

[39] Burigede Liu, Nikola Kovachki, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, Andrew M. Stuart, and Kaushik Bhattacharya. A learning-based multiscale method and its application to inelastic impact problems. *Journal of the Mechanics and Physics of Solids*, 158:104668, 2022.

[40] Nikola Kovachki, Burigede Liu, Xingsheng Sun, Hao Zhou, Kaushik Bhattacharya, Michael Ortiz, and Andrew Stuart. Multiscale modeling of materials: Computing, data science, uncertainty and goal-oriented optimization. *arXiv preprint arXiv:2104.05918*, 2021.

[41] Yifan Chen, Thomas Y. Hou, and Yixuan Wang. Exponential convergence for multiscale linear elliptic PDEs via adaptive edge basis functions. *Multiscale Modeling & Simulation*, 19(2):980–1010, 2021.

[42] Yifan Chen, Thomas Y. Hou, and Yixuan Wang. Exponentially convergent multiscale methods for high frequency heterogeneous Helmholtz equations. *arXiv preprint arXiv:2105.04080*, 2021.

[43] Houbing Song, Danda B. Rawat, Sabina Jeschke, and Christian Brecher. *Cyber-Physical Systems: Foundations, Principles and Applications*. Morgan Kaufmann, 2016.

[44] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[45] Clarence W. Rowley, Igor Mezić, Shervin Bagheri, Philipp Schlatter, and Dan S. Henningson. Spectral analysis of nonlinear flows. *Journal of Fluid Mechanics*, 641:115–127, 2009.

[46] Jonathan H. Tu. *Dynamic Mode Decomposition: Theory and Applications*. PhD thesis, Princeton University, 2013.

[47] Kunihiko Taira, Steven L. Brunton, Scott T.M. Dawson, Clarence W. Rowley, Tim Colonius, Beverley J. McKeon, Oliver T. Schmidt, Stanislav Gordeyev, Vassilios Theofilis, and Lawrence S. Ukeiley. Modal analysis of fluid flows: An overview. *AIAA Journal*, 55(12):4013–4041, 2017.

[48] Igor Mezić. Spectral properties of dynamical systems, model reduction and decompositions. *Nonlinear Dynamics*, 41(1):309–325, 2005.

[49] I. Mezić. Analysis of fluid flows via spectral properties of the Koopman operator. *Annual Review of Fluid Mechanics*, 45:357–378, 2013.

[50] Marc C. Kennedy and Anthony O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

[51] Emmet Cleary, Alfredo Garbuno-Inigo, Shiwei Lan, Tapio Schneider, and Andrew M. Stuart. Calibrate, emulate, sample. *Journal of Computational Physics*, 424:109716, 2021.

[52] Yifan Chen, Bamdad Hosseini, Houman Owhadi, and Andrew M. Stuart. Solving and learning nonlinear PDEs with Gaussian processes. *arXiv preprint arXiv:2103.12959*, 2021.

[53] Ronald A. DeVore. The theoretical foundation of reduced basis methods. *Model Reduction and Approximation: Theory and Algorithms*, pages 137–168, 2014.

[54] Kevin Carlberg, Charbel Farhat, Julien Cortial, and David Amsallem. The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows. *Journal of Computational Physics*, 242:623–647, 2013.

[55] Benjamin Peherstorfer and Karen Willcox. Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering*, 306:196–215, 2016.

[56] Peter Binev, Albert Cohen, Wolfgang Dahmen, Ronald DeVore, Guergana Petrova, and Przemyslaw Wojtaszczyk. Data assimilation in reduced modeling. *SIAM/ASA Journal on Uncertainty Quantification*, 5(1):1–29, 2017.

[57] Albert Cohen, Wolfgang Dahmen, and Ron DeVore. State estimation–the role of reduced models. *arXiv preprint arXiv:2002.00220*, 2020.

[58] Elizabeth Qian, Boris Kramer, Benjamin Peherstorfer, and Karen Willcox. Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems. *Physica D: Nonlinear Phenomena*, 406:132401, 2020.

[59] Elizabeth Qian, Ionut-Gabriel Farcas, and Karen Willcox. Reduced operator inference for nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 44(4), 2022.

[60] Sebastian Grimberg, Charbel Farhat, Radek Tezaur, and Charbel Bou-Mosleh. Mesh sampling and weighting for the hyperreduction of nonlinear Petrov–Galerkin reduced-order models with local reduced-order bases. *International Journal for Numerical Methods in Engineering*, 122(7):1846–1874, 2021.

[61] Ion Victor Gosea, Serkan Gugercin, and Christopher Beattie. Data-driven balancing of linear dynamical systems. *arXiv preprint arXiv:2104.01006*, 2021.

[62] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[63] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data. *Phys. Rev. Fluids*, 2:034603, Mar 2017.

[64] Jin-Long Wu, Heng Xiao, and Eric Paterson. Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework. *Phys. Rev. Fluids*, 3:074602, Jul 2018.

[65] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51(1):357–377, 2019.

[66] Jamshid Ghaboussi, David A. Pecknold, Mingfu Zhang, and Rami M. Haj-Ali. Autoprogressive training of neural network constitutive models. *International Journal for Numerical Methods in Engineering*, 42(1):105–126, 1998.

[67] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Deep potential molecular dynamics: A scalable model with the accuracy of quantum mechanics. *Physical Review Letters*, 120(14):143001, 2018.

[68] Philip Avery, Daniel Z. Huang, Wanli He, Johanna Ehlers, Armen Derkevorkian, and Charbel Farhat. A computationally tractable framework for nonlinear dynamic multiscale modeling of membrane woven fabrics. *International Journal for Numerical Methods in Engineering*, 122(10):2598–2625, 2021.

[69] Xin Liu, Su Tian, Fei Tao, and Wenbin Yu. A review of artificial neural networks in the constitutive modeling of composite materials. *Composites Part B: Engineering*, 224:109152, 2021.

[70] Denghui Lu, Han Wang, Mohan Chen, Lin Lin, Roberto Car, Weinan E, Weile Jia, and Linfeng Zhang. 86 PFLOPS deep potential molecular dynamics simulation of 100 million atoms with ab initio accuracy. *Computer Physics Communications*, 259:107624, 2021.

[71] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

[72] R. Swischuk, L. Mainini, B. Peherstorfer, and K. Willcox. Projection-based model reduction: Formulations for physics-based machine learning. *Computers and Fluids*, 179:704–717, 2019.

[73] Kookjin Lee and Kevin T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020.

[74] Qianxiao Li, Felix Dietrich, Erik M. Bollt, and Ioannis G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 2017.

[75] Jeremy Morton, Freddie D. Witherden, Antony Jameson, and Mykel J. Kochenderfer. Deep dynamical modeling and control of unsteady fluid flows. *arXiv preprint arXiv:1805.07472*, 2018.

[76] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as Gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[77] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[78] Liu Yang, Xuhui Meng, and George Em Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

[79] Jiequn Han, Arnulf Jentzen, et al. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[80] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[81] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[82] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[83] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[84] Somdatta Goswami, Cosmin Anitescu, Souvik Chakraborty, and Timon Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics*, 106:102447, 2020.

[85] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[86] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[87] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[88] Luning Sun and Jian-Xun Wang. Physics-constrained bayesian neural network for fluid flow reconstruction with sparse and noisy data. *Theoretical and Applied Mechanics Letters*, 10(3):161–169, 2020.

[89] Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[90] Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. ReLU deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.

[91] Weinan E, Chao Ma, and Lei Wu. Barron spaces and the compositional function spaces for neural network models. *arXiv preprint arXiv:1906.08039*, 2019.

[92] Yeonjong Shin, Jerome Darbon, and George Em Karniadakis. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs. *arXiv preprint arXiv:2004.01806*, 2020.

[93] Ingrid Daubechies, Ronald DeVore, Simon Foucart, Boris Hanin, and Guergana Petrova. Nonlinear approximation and (deep) ReLU networks. *Constructive Approximation*, pages 1–46, 2021.

[94] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[95] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A*, 476(2239):20200334, 2020.

[96] Zheyuan Hu, Ameya D. Jagtap, George Em Karniadakis, and Kenji Kawaguchi. When do extended physics-informed neural networks (XPINNs) improve generalization? *arXiv preprint arXiv:2109.09444*, 2021.

[97] Ameya D. Jagtap, Yeonjong Shin, Kenji Kawaguchi, and George Em Karniadakis. Deep Kronecker neural networks: A general framework for neural networks with adaptive activation functions. *Neurocomputing*, 468:165–180, 2022.

[98] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

[99] Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M. Benson. U-FNO–an enhanced fourier neural operator based-deep learning model for multiphase flow. *arXiv preprint arXiv:2109.03697*, 2021.

[100] Zongyi Li, Daniel Z. Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for PDEs on general geometries. *arXiv preprint arXiv:2207.05209*, 2022.

[101] Craig R. Gin, Daniel E. Shea, Steven L. Brunton, and J. Nathan Kutz. DeepGreen: Deep learning of Green's functions for nonlinear boundary value problems. *arXiv preprint arXiv:2101.07206*, 2020.

[102] Nicolas Boullé and Alex Townsend. Learning elliptic partial differential equations with randomized linear algebra. *Foundations of Computational Mathematics*, pages 1–31, 2022.

[103] Yuwei Fan and Lexing Ying. Solving electrical impedance tomography with deep learning. *Journal of Computational Physics*, 404:109119, 2020.

[104] Yuwei Fan and Lexing Ying. Solving inverse wave scattering with deep learning. *arXiv preprint arXiv:1911.13202*, 2019.

[105] Yuehaw Khoo and Lexing Ying. SwitchNet: A neural network model for forward and inverse scattering problems. *SIAM Journal on Scientific Computing*, 41(5):A3182–A3201, 2019.

[106] Georgios Kissas, Jacob Seidman, Leonardo Ferreira Guilhoto, Victor M. Preciado, George J. Pappas, and Paris Perdikaris. Learning operators with coupled attention. *arXiv preprint arXiv:2201.01032*, 2022.

[107] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[108] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[109] Somdatta Goswami, Minglang Yin, Yue Yu, and George Em Karniadakis. A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 391:114587, 2022.

[110] Maitreyee Sharma Priyadarshini, Simone Venturi, and Marcp Panesi. Application of DeepONet to model inelastic scattering probabilities in air mixtures. In *AIAA Aviation 2021 Forum*, page 3144, 2021.

[111] Ronald A. DeVore. Nonlinear approximation. *Acta Numerica*, 7:51–150, 1998.

[112] Nicholas H. Nelsen and Andrew M. Stuart. The random feature model for input-output maps between banach spaces. *SIAM Journal on Scientific Computing*, 43(5):A3212–A3243, 2021.

[113] Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for Fourier neural operators. *arXiv preprint arXiv:2107.07562*, 2021.

[114] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[115] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[116] Mohammad Farazmand and Themistoklis P. Sapsis. A variational approach to probing extreme events in turbulent dynamical systems. *Science Advances*, 3(9):e1701533, 2017.

[117] Andrew Majda and Xiaoming Wang. *Nonlinear Dynamics and Statistical Theories for Basic Geophysical Flows*. Cambridge University Press, 2006.

[118] Steven A. Orszag and G.S. Patterson Jr. Numerical simulation of three-dimensional homogeneous isotropic turbulence. *Physical Review Letters*, 28(2):76, 1972.

[119] Elena Beretta, Maarten V. De Hoop, Florian Faucher, and Otmar Scherzer. Inverse boundary value problem for the Helmholtz equation: Quantitative conditional Lipschitz stability estimates. *SIAM Journal on Mathematical Analysis*, 48(6):3962–3983, 2016.

[120] João Paulo Pascon. Large deformation analysis of plane-stress hyperelastic problems via triangular membrane finite elements. *International Journal of Advanced Structural Engineering*, 11(3):331–350, 2019.

[121] Daniel Z. Huang, Kailai Xu, Charbel Farhat, and Eric Darve. Learning constitutive relations from indirect observations using deep neural networks. *Journal of Computational Physics*, 416:109491, 2020.

[122] Kailai Xu, Daniel Z. Huang, and Eric Darve. Learning constitutive relations using symmetric positive definite neural networks. *Journal of Computational Physics*, 428:110072, 2021.

[123] Russel E Caflisch. Monte carlo and quasi-monte carlo methods. *Acta Numerica*, 7:1–49, 1998.

[124] Maarten V. de Hoop, Nikola B. Kovachki, Nicholas H. Nelsen, and Andrew M. Stuart. Convergence rates for learning linear operators from noisy data. *arXiv preprint arXiv:2108.12515*, 2021.

[125] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: An extensive benchmark for scientific machine learning.

[126] Samuel Lanthaler, Siddhartha Mishra, and George Em Karniadakis. Error estimates for DeepONets: A deep learning framework in infinite dimensions. *arXiv preprint arXiv:2102.09618*, 2021.

[127] Christoph Schwab and Jakob Zech. Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in UQ. *Analysis and Applications*, 17(01):19–55, 2019.

[128] Joost A.A. Opschoor, Ch. Schwab, and Jakob Zech. Exponential ReLU DNN expression of holomorphic maps in high dimension. *Constructive Approximation*, 55(1):537–582, 2022.

[129] Gitta Kutyniok, Philipp Petersen, Mones Raslan, and Reinhold Schneider. A theoretical analysis of deep neural networks and parametric PDEs. *Constructive Approximation*, 55(1):73–125, 2022.