Journal of Machine Learning Research 23 (2022) 1-63 Submitted 12/21; Revised 5/22; Published 7/22

Learning Operators with Coupled Attention

Georgios Kissas * GKISSAS@SEAS.UPENN.EDU
Department of Mechanical Engineering and Applied Mechanics
University of Pennsylvania
Philadelphia, PA 19104

Jacob H. Seidman * SEIDJ@SAS.UPENN.EDU
Graduate Group in Applied Mathematics and Computational Science
University of Pennsylvania
Philadelphia, PA 19104

Leonardo Ferreira Guilhoto GUILHOTO@SAS.UPENN.EDU
Graduate Group in Applied Mathematics and Computational Science
University of Pennsylvania
Philadelphia, PA 19104

Victor M. Preciado PRECIADO@SEAS.UPENN.EDU
Department of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104

George J. Pappas PAPPASG@SEAS.UPENN.EDU
Department of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104

Paris Perdikaris PGP@SEAS.UPENN.EDU
Department of Mechanical Engineering and Applied Mechanics
University of Pennsylvania
Philadelphia, PA 19104

Editor: Animashree Anandkumar

Abstract

Supervised operator learning is an emerging machine learning paradigm with applications to modeling the evolution of spatio-temporal dynamical systems and approximating general black-box relationships between functional data. We propose a novel operator learning method, LOCA (Learning Operators with Coupled Attention), motivated from the recent success of the attention mechanism. In our architecture, the input functions are mapped to a finite set of features which are then averaged with attention weights that depend on the output query locations. By coupling these attention weights together with an integral transform, LOCA is able to explicitly learn correlations in the target output functions, enabling us to approximate nonlinear operators even when the number of output function measurements in the training set is very small. Our formulation is accompanied by rigorous approximation theoretic guarantees on the universal expressiveness of the proposed model. Empirically, we evaluate the performance of LOCA on several operator learning scenarios involving systems governed by ordinary and partial differential equations, as well as a

[*] These authors contributed equally.

black-box climate prediction problem. Through these scenarios we demonstrate state of the art accuracy, robustness with respect to noisy input data, and a consistently small spread of errors over testing data sets, even for out-of-distribution prediction tasks.

Keywords: deep learning, reproducing kernel Hilbert spaces, wavelet scattering networks, functional data analysis, universal approximation

1. Introduction

The great success of modern deep learning lies in its ability to approximate maps between finite-dimensional vector spaces, as in computer vision (Santhanam et al., 2017), natural language processing (Vaswani et al., 2017), precision medicine (Rajkomar et al., 2019), bio-engineering (Kissas et al., 2020), and other data driven applications. A particularly successful class of such models are those built with the attention mechanism (Bahdanau et al., 2015). For example, the Transformer is an attention-based architecture that has recently produced state of the art performance in natural language processing (Vaswani et al., 2017), computer vision (Dosovitskiy et al., 2020; Parmar et al., 2018), and audio signal analysis (Gong et al., 2021; Huang et al., 2018).

Another active area of research is applying machine learning techniques to approximate operators between spaces of functions. These methods are particularly attractive for many problems in computational physics and engineering where the goal is to learn the functional response of a system from a functional input, such as an initial/boundary condition or forcing term. In the context of learning the response of systems governed by differential equations, these learned models can function as fast surrogates of traditional numerical solvers.

For example, in climate modelling one might wish to predict the pressure field over the earth from measurements of the surface air temperature field. The goal is then to learn an operator, $\mathcal{F}$, between the space of temperature functions to the space of pressure functions (see Figure 1). An initial attempt at solving this problem might be to take a regular grid of measurements over the earth for the input and output fields and formulate the problem as a (finite-dimensional) image to image regression task. While architectures such as convolutional neural networks may perform well under this setting, this approach can be somewhat limited. For instance, if we desired the value of the output at a query location outside of the training grid, an entirely new model would need to be built and tuned from scratch. This is a consequence of choosing to discretize the regression problem before building a model to solve it. If instead we formulate the problem and model at the level of the (infinite-dimensional) input and output function spaces, and *then* make a choice of discretization, we can obtain methods that are more flexible with respect to the locations of the point-wise measurements.

Formulating models with functional data is the topic of Functional Data Analysis (FDA) (Ramsay, 1982; Ramsay and Dalzell, 1991), where parametric, semi-parametric or non-parametric methods operate on functions in infinite-dimensional vector spaces. A useful class of non-parametric approaches are operator-valued kernel methods. These methods generalize the use of scalar-valued kernels for learning functions in a Reproducing Kernel Hilbert Space (RKHS) (Hastie et al., 2009) to RKHS's of operators. Kernel methods were thoroughly studied in the past (Hofmann et al., 2008; Shawe-Taylor et al., 2004) and have

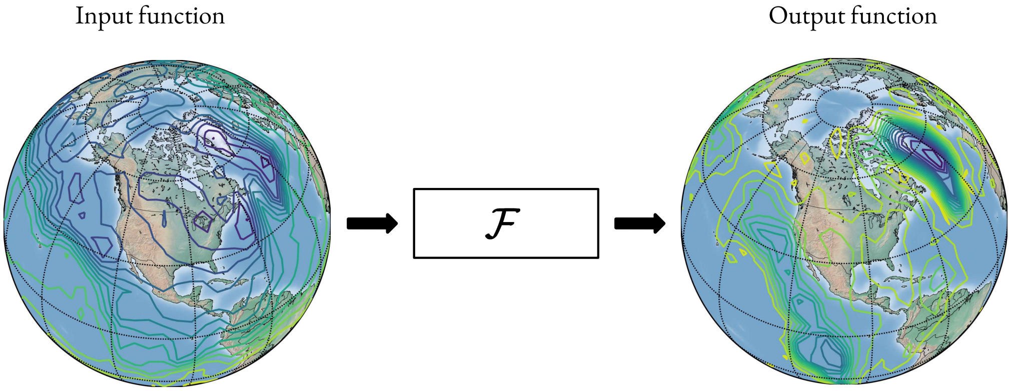


Figure 1: An example sketch of operator learning for climate modeling: By solving an operator learning problem, we can approximate an infinite-dimensional map between two functions of interest, and then predict one function using the other. For example, by providing the model with an input function, e.g. a surface air temperature field, we can predict an output function, e.g. the corresponding surface air pressure field.

been successfully applied to nonlinear and high-dimensional problem settings (Takeda et al., 2007; Dou and Liang, 2020). Previous work has successfully extended this framework to learning operators between more general vector spaces as well (Micchelli and Pontil, 2005; Caponnetto et al., 2008; Kadri et al., 2010, 2016; Owhadi, 2020). This framework is particularly powerful as the inputs can be continuous or discrete, and the underlying vector spaces are typically only required to be normed and separable.

A parametric-based approach to operator learning was introduced in Chen and Chen (1995) where the authors proposed a method for learning non-linear operators based on a one-layer feed-forward neural network architecture. Moreover, the authors presented a universal approximation theorem which ensures that their architecture can approximate any continuous operator with arbitrary accuracy. Lu et al. (2019) gave an extension of this architecture, called DeepONet, built with multiple layer feed-forward neural networks, and demonstrated its effectiveness in approximating the solution operators of various differential equations. In follow up work, error estimates were derived for some specific problem scenarios (Lanthaler et al., 2022), and several applications have been pursued (Cai et al., 2020; Di Leoni et al., 2021; Lin et al., 2021). An extension of the DeepONet was proposed by Wang et. al. (Wang et al., 2021c; Wang and Perdikaris, 2021; Wang et al., 2021b), where a regularization term is added to the loss function to enforce known physical constraints, enabling one to predict solutions of parametric differential equations, even in the absence of paired input-output training data.

Another parametric approach to operator learning is the Graph Neural Operator proposed by Li et al. (2020c), motivated by the solution form of linear partial differential equations (PDEs) and their Greens' functions. As an extension of this work, the authors also proposed a Graph Neural Operator architecture where a multi-pole method is used sample the spatial grid (Li et al., 2020b) allowing the kernel to learn in a non-local manner. In later published work, this framework has been extended to the case where the integral kernel is stationary, enabling one to efficiently compute the integral operator in the Fourier domain (Li et al., 2020a) resulting in Fourier Neural Operators. Universality results and error estimates for Fourier Neural Operators were explored in Kovachki et al. (2021a).

Other parametric-based models include a deep learning approach for directly approximating the Green's function of differential equations (Gin et al., 2021), using a neural network to learn a transformation between finite dimensional PCA representations of functional data (Bhattacharya et al., 2021), a multi-wavelet approach for learning projections of an integral kernel operator to approximate the true operator (Gupta et al., 2021) and a random feature approach for learning the solution map of PDEs (Nelsen and Stuart, 2020).

While some of the previously described operator learning methods can be seen as generalizations of deep learning architectures such as feed-forward and convolutional neural networks, in this paper we are motivated by the success of the attention mechanism to propose a new operator learning framework. Specifically, we draw inspiration from the Badhanau attention mechanism (Bahdanau et al., 2015), which first constructs a feature representation of the input and then averages these features with a distribution that depends on the argument of the output function to obtain its value. We will additionally couple these distributions together in what we call a *Kernel-Coupled Attention* mechanism. This will allow our framework to explicitly model correlations within the output functions of the operator and train better with fewer output function measurements. Moreover, we prove that under certain assumptions the model satisfies a universal approximation property.

The main contributions of this work can be summarized in the following points:

- **Novel Architecture:** We propose an operator learning framework inspired by the attention mechanism, operator approximation theory, and the Reproducing Kernel Hilbert Space (RKHS) literature. A novel *Kernel-Coupled Attention* mechanism is introduced to explicitly model correlations between the output functions' query locations.

- **Theoretical Guarantees:** We prove that the proposed framework satisfies a universal approximation property, that is, it can approximate any continuous operator with arbitrary accuracy.

- **Data Efficiency:** By modelling correlations between output queries, our model can achieve high performance when trained with only a small fraction (6-12%) of the total available labeled data compared to competing methods.

- **Robustness:** Compared to existing methods, our model demonstrates superior robustness with respect to noise corruption in the training and testing inputs, as well as randomness in the model initialization. Our model's performance is stable in that the errors on the test data set are consistently concentrated around the median with significantly fewer outliers compared to other methods.

- **Generalization:** On a real data set of Earth surface air temperature and pressure measurements, our model is able to learn the functional relation between the two fields with high accuracy and extrapolate beyond the training data. On synthetic data we demonstrate that our model is able to generalize better than competing methods over increasingly out-of-distribution examples.

These contributions are accompanied with systematic experiments examining the effect of the architecture components of LOCA, both in isolation and together. The rest of the paper is structured as follows. In Section 2 we introduce the supervised operator learning problem. In Section 3, we introduce the general form of the model and in following subsections present the construction of its different components. In Section 4 we prove theoretical results on the approximation power of this class of models. In Section 5 we present the specific architecture choices made for implementing our method in practice. Section 6 discusses the similarities and differences of our model with related operator learning approaches and known operator learning error estimates. In Section 7, we demonstrate the performance of the proposed methodology across different benchmarks in comparison to other state-of-the-art methods. In addition, we provide a new metric for assessing how close trained models align with an "optimal" representation of output functions and evaluate this metric on a synthetic example. In Section 8, we discuss our main findings, outline potential drawbacks of the proposed method, and highlight future directions emerging from this study.

2. Problem Formulation

We now provide a formal definition of the operator learning problem. Given $\mathcal{X} \subset \mathbb{R}^{d_x}$, $\mathcal{Y} \subset \mathbb{R}^{d_y}$, we will refer to a point $x \in \mathcal{X}$ as an *input location* and a point $y \in \mathcal{Y}$ as a *query location*. Denote by $C(\mathcal{X}, \mathbb{R}^{d_u})$ and $C(\mathcal{Y}, \mathbb{R}^{d_s})$ the spaces of continuous functions from $\mathcal{X} \to \mathbb{R}^{d_u}$ and $\mathcal{Y} \to \mathbb{R}^{d_s}$, respectively. We will refer to $C(\mathcal{X}, \mathbb{R}^{d_u})$ as the space of *input functions* and $C(\mathcal{Y}, \mathbb{R}^{d_s})$ the space of *output functions*. For example, in Figure 1, if we aim to learn the correspondence between a temperature field over the earth and the corresponding pressure field, $u \in C(\mathcal{X}, \mathbb{R})$ would represent the temperature field and $s \in C(\mathcal{Y}, \mathbb{R})$ would be a pressure field, where $\mathcal{X} = \mathcal{Y}$ represents the surface of the earth. With a data set of of input/output function pairs, we formulate the supervised operator learning problem as follows.

Problem 1 *Given N pairs of input and output functions $\{u^\ell(x), s^\ell(y)\}_{\ell=1}^N$ generated by some possibly unknown ground truth operator $\mathcal{G} : C(\mathcal{X}, \mathbb{R}^{d_u}) \to C(\mathcal{Y}, \mathbb{R}^{d_s})$ with $u^\ell \in C(\mathcal{X}, \mathbb{R}^{d_u})$ and $s^\ell \in C(\mathcal{Y}, \mathbb{R}^{d_s})$, learn an operator $\mathcal{F} : C(\mathcal{X}, \mathbb{R}^{d_u}) \to C(\mathcal{Y}, \mathbb{R}^{d_s})$, such that for $\ell = 1, \dots, N$,*

$$\mathcal{F}(u^\ell) = s^\ell.$$

This problem also encompasses scenarios where more structure is known about the input/output functional relation. For example, u could represent the initial condition to a partial differential equation and s the corresponding solution. In this case, $\mathcal{G}$ would correspond to the true solution operator and $\mathcal{F}$ would be an approximate surrogate model. Similarly, u could represent a forcing term in a dynamical system described by an ordinary differential equation, and s the resulting integrated trajectory. In these two scenarios there do exist a suite of alternate methods to obtain the solution function s from the input u, but with an appropriate choice of architecture for $\mathcal{F}$ the approximate model can result in significant computational speedups. Sufficiently differentiable surrogates additionally permit the fast computation of sensitivities with respect to the inputs using tools like automatic differentiation.

Note that while the domains $\mathcal{X}$ and $\mathcal{Y}$ need not be discrete sets, in practice we may only have access to the functions u^ℓ and s^ℓ evaluated at finitely many locations. However, we take the perspective that it is beneficial to formulate the model with continuously sampled input data, and consider the consequences of discretization at implementation time. As we shall see, this approach will allow us to construct a model that is able to learn operators over multiple output resolutions simultaneously.

3. Proposed Model: Learning Operators with Coupled Attention (LOCA)

We will construct our model through the following two steps. Inspired by the attention mechanism (Bahdanau et al., 2015), we will first define a class of models where the input functions u are lifted to a feature vector $v(u) \in \mathbb{R}^{n\times d_s}$. Each output location $y \in \mathcal{Y}$ will define d_s probability distributions $\varphi(y) \in \prod_{i=1}^{d_s} \Delta^n$, where Δ^n is the the n-simplex. The forward pass of the model is then computed by averaging the rows of $v(u)$ over the probability distributions $\varphi(y)$.

Next, we augment this model by coupling the probability distributions $\varphi(y)$ across different query points $y \in \mathcal{Y}$. This is done by acting on a proposal score function $g : \mathcal{Y} \to \mathbb{R}^{n\times d_s}$ with a kernel integral operator. The form of the kernel determines the similarities between the resulting distributions. In Section 7.1 we will empirically demonstrate that the coupled version of our model is more accurate compared to the uncoupled version when the number of output function evaluations per example is small.

3.1 The Attention Mechanism

The attention mechanism was first formulated in Bahdanau et al. (2015) for use in language translation. The goal of their work was to translate an input sentence in a given language $\{u_1, \dots, u_{T_u}\}$ to a sentence in another language $\{s_1, \dots, s_{T_s}\}$. A context vector c_i was associated to each index of the output sentence, $i \in \{1, \dots, T_s\}$, and used to construct a probability distribution of the i-th word in the translated sentence, s_i. The attention mechanism is a way to construct these context vectors by averaging over features associated with the input in a way that depends on the output index i.

More concretely, the input sentence is first mapped to a collection of features $\{v_1, \dots, v_{T_u}\}$. Next, depending on the input sentence and the location/index i in the output (translated) sentence, a discrete probability distribution $\{\varphi_{i1}, \dots, \varphi_{iT_u}\}$ is formed over the input indices such that

$$\varphi_{ij} \geq 0, \quad \sum_{j=1}^{T_u} \varphi_{ij} = 1.$$

The context vector at index i is then computed as

$$c_i = \sum_{j=1}^{T_u} \varphi_{ij} v_j.$$

If the words in the input sentence are represented by vectors in $\mathbb{R}^d$, and the associated features and context vector are in $\mathbb{R}^l$, the attention mechanism can be represented by the following diagram.

$$\begin{tikzcd}
{[T_s] \times \mathbb{R}^{T_u \times d}} \arrow[r, "\mathrm{Attn}"] \arrow[d, "{(\varphi,v)}"'] & \mathbb{R}^l \\
\Delta^{T_u} \times \mathbb{R}^{T_u \times l} \arrow[ur, "\mathbb{E}"'] &
\end{tikzcd}$$

We will apply this attention mechanism to learn operators between function spaces by mapping an input function u to a finite set of features $v(u) \in \mathbb{R}^{n \times d_s}$, and taking an average over these features with respect to d_s distributions $\varphi(y) \in \prod_{k=1}^{d_s} \Delta^n$ that depend on the query location $y \in \mathcal{Y}$ for the output function. That is,

$$\mathcal{F}(u)(y) := \mathbb{E}_{\varphi(y)}[v(u)],$$

where $v(u) \in \mathbb{R}^{n \times d_s}$, φ is a function from $y \in \mathcal{Y}$ to d_s copies of the n-dimensional simplex Δ^n, and $\mathbb{E} : \prod_{k=1}^{d_s} \Delta^n \times \mathbb{R}^{n \times d_s} \to \mathbb{R}^{d_s}$ is an expectation operator that takes $(\varphi, v) \mapsto \sum_i \varphi_i \odot v_i$, where $\odot$ denotes an element-wise product. This can be represented by the following diagram.

$$\begin{tikzcd}
\mathcal{Y} \times C(\mathcal{X}, \mathbb{R}^{d_u}) \arrow[r, "\mathcal{F}"] \arrow[d, "{(\varphi,v)}"'] & \mathbb{R}^{d_s} \\
\prod_{k=1}^{d_s} \Delta^n \times \mathbb{R}^{n \times d_s} \arrow[ur, "\mathbb{E}"'] &
\end{tikzcd}$$

In summary, we propose an operator learning architecture where the value of the output function at a location y is determined by attending to an input vector $v(u)$ with a probability distribution $\varphi(y)$, dependent on y. It remains to choose the functional form for the mappings $v : C(\mathcal{X}, \mathbb{R}^{d_u}) \to \mathbb{R}^n$ and $\varphi : \mathcal{Y} \to \prod_{i=1}^{d_s} \Delta^n$, which we describe in the following sections.

3.2 Kernel-Coupled Attention Weights

In this section we provide a definition for the form of the attention weight function $\varphi : \mathcal{Y} \to \prod_{i=1}^{d_s} \Delta^n$. We first consider a φ defined by the softmax normalization of a score network $g : \mathcal{Y} \to \mathbb{R}^{n \times d_s}$. By augmenting this definition with a kernel convolution of the score function g, we arrive at the Kernel-Coupled Attention weights.

To define a trainable function mapping query locations $y \in \mathcal{Y}$ to attention weights $\varphi(y) \in \Delta^n$, we must ensure that $\varphi(y)$ forms a discrete probability distribution. A simple way of achieving this is to take a trainable un-normalized score function $g : \mathcal{Y} \to \mathbb{R}^n$ and acting on it with the softmax function

$$\begin{aligned}
\sigma : \mathbb{R}^n &\to \Delta^n, \\
g &\mapsto \left(\tfrac{\exp(g_1)}{\sum_{i=1}^n \exp(g_i)}, \quad \cdots \quad , \tfrac{\exp(g_n)}{\sum_{i=1}^n \exp(g_i)} \right)^\top .
\end{aligned}$$

We can extend this definition to create a function $\varphi : \mathcal{Y} \to \prod_{i=1}^{d_s} \Delta^n$ by using a score function $g : \mathcal{Y} \to \mathbb{R}^{n \times d_s}$ and acting on the rows of $g(y)$ with the softmax function as $\varphi(y) = \sigma(g(y))$.

The disadvantage of this formulation is that it solely relies on the form of the function g to capture relations between the distributions $\varphi(y)$ across different $y \in \mathcal{Y}$. Instead, we introduce the Kernel-Coupled Attention (KCA) mechanism to model these relations by integrating the function g against a coupling kernel $\kappa : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. This results in the score function

$$\tilde{g}(y) = \int_{\mathcal{Y}} \kappa(y, y')g(y')\, dy', \tag{1}$$

which can be normalized across its rows to form the probability distributions

$$\varphi(y) = \sigma\left(\int_{\mathcal{Y}} \kappa(y, y')g(y')\, dy'\right). \tag{2}$$

The form of the kernel κ will determine how these distributions are coupled across $y \in \mathcal{Y}$. For example, given a fixed y, the locations y' where $k(y, y')$ is large will enforce similarity between the corresponding score functions $\tilde{g}(y)$ and $\tilde{g}(y')$. If k is a local kernel with a small bandwidth then points y and y' will only be forced to have similar score functions if they are very close together. Thus, the kernel transformation forces pairs of attention weights $\varphi(y), \varphi(y')$ for the features $v(u)$ to be similar when y and y' are deemed similar by the kernel k.

We define a kernel for the transformation in (2) by first lifting the points $y \in \mathcal{Y}$ via a nonlinear parameterized mapping $q_\theta : \mathcal{Y} \to \mathbb{R}^l$. We then apply a universal kernel $k : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$ (Micchelli and Pontil, 2004) over the lifted space, such as the Gaussian RBF kernel,

$$k(z, z') = \gamma \exp(-\beta\|z - z'\|^2), \quad \gamma, \beta > 0. \tag{3}$$

Finally, we apply a normalization to the output of this kernel on the lifted points to create a similarity measure. The effect of the normalization is to maintain the relative scale of the proposal score function g. Overall, our kernel is defined as

$$\kappa(y, y') := \frac{k(q_\theta(y), q_\theta(y'))}{\left(\int_{\mathcal{Y}} k(q_\theta(y), q_\theta(z))dz\right)^{1/2}\left(\int_{\mathcal{Y}} k(q_\theta(y'), q_\theta(z))dz\right)^{1/2}}. \tag{4}$$

By tuning the parameters θ, β and γ in the functions q_θ and k, the kernel κ is able to learn the appropriate measures of similarity between the points in the output function domain $\mathcal{Y}$. We show in Section 7.4 that the inclusion of the kernel transformation in (2) can have a strong effect on the performance of the model in the small data regime.

3.3 Input Function Feature Encoding

The last architecture choice to be made concerns the functional form of the feature embedding $v(u)$. Here, we construct the map v as a composition of two mappings. The first is a function

$$\mathcal{D} : C(\mathcal{X}, \mathbb{R}^{d_u}) \to \mathbb{R}^d, \tag{5}$$

that maps an input function u to a finite-dimensional vector $\mathcal{D}(u) \in \mathbb{R}^d$. After creating the d-dimensional representation of the input function $\mathcal{D}(u)$, we pass this vector through a function f from a class of universal function approximators, such as fully connected neural

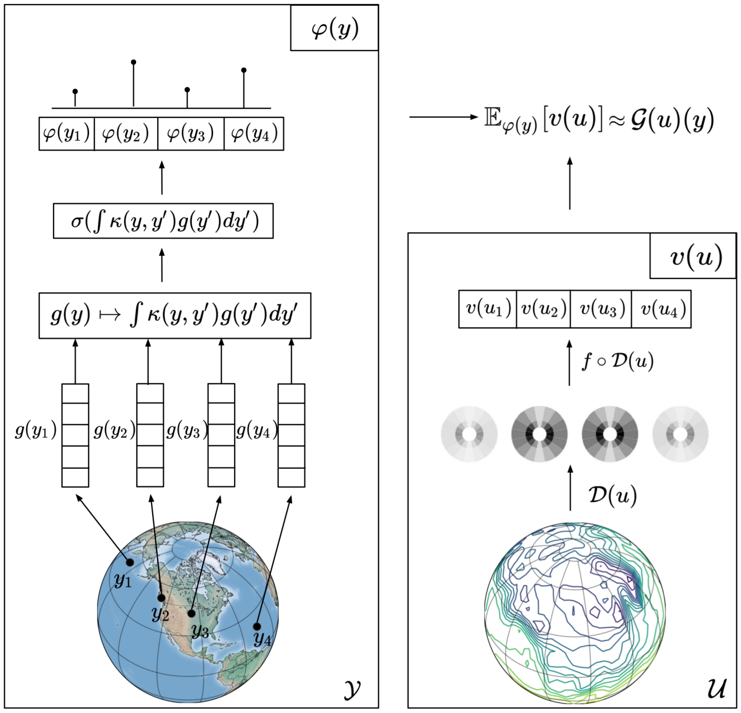

$\varphi(y)$
$\varphi(y_1)$ $\varphi(y_2)$ $\varphi(y_3)$ $\varphi(y_4)$
$\sigma(\int \kappa(y,y')g(y')dy')$
$g(y) \mapsto \int \kappa(y,y')g(y')dy'$
$g(y_1)$ $g(y_2)$ $g(y_3)$ $g(y_4)$
y_1 y_2 y_3 y_4
$\mathcal{Y}$
$\mathbb{E}_{\varphi(y)}[v(u)] \approx \mathcal{G}(u)(y)$
$v(u)$
$v(u_1)$ $v(u_2)$ $v(u_3)$ $v(u_4)$
$f \circ \mathcal{D}(u)$
$\mathcal{D}(u)$
$\mathcal{U}$

Figure 2: Schematic illustration of LOCA for $n = 4$ and $d_s = 1$: The LOCA method builds a feature representation, $v(u)$, of the input function and averages it with respect to $\varphi(y)$. The transform $\mathcal{D}$ is first applied to the input function to produce a list of features, illustrated by disks in this case, and then a fully-connected network is applied to construct $v(u)$. The score function g is applied to the output query locations y_i together with the softmax function to produce the score vector φ_i. The v_i and φ_i vectors are combined to evaluate the solution at each query location by computing $\mathbb{E}_{\varphi(y)}[v(u)]$ at the last step.

networks. The composition of these two operations forms our feature representation of the input function,

$$v(u) = f \circ \mathcal{D}(u). \tag{6}$$

One example for the operator $\mathcal{D}$ is the image of the input function under d linear functionals on $C(\mathcal{X}; \mathbb{R}^{d_u})$. For example, $\mathcal{D}$ could return the point-wise evaluation of the input function at d fixed points. This would correspond to the action of d translated δ-functionals. The drawback of such an approach is that the model would not be able to accept measurements of the input function at any other locations. As a consequence the input resolution could never vary across forward passes of the model.

Alternatively, if we consider an orthonormal basis for $L^2(\mathcal{X}; \mathbb{R}^{d_u})$, we could also have $\mathcal{D}$ be the projection onto the first d basis vectors. For example, if we use the basis of trigonometric polynomials the Fast Fourier Transform (FFT) (Cooley and Tukey, 1965) allows for efficient computation of these values and can be performed across varying grid resolutions. We could also consider the projection onto an orthogonal wavelet basis (Daubechies, 1988). In the case of complex valued coefficients for these basis functions, the range space dimension of $\mathcal{D}$ would be doubled to account for the real and imaginary part of these measurements.

3.4 Model Summary

Overall, the forward pass of the proposed model is written as follows, see Figure 2 for a visual representation.

$$\mathcal{F}(u)(y) = \mathbb{E}_{\varphi(y)}[v(u)] = \sum_{i=1}^{n} \sigma\left(\int_{\mathcal{Y}} \kappa(y, y')g(y')\, dy'\right)_i \odot v_i(u), \tag{7}$$

where $\kappa : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is the kernel of equation (1), σ is the softmax function, v the input feature encoder and g is the proposed score function. Some theoretical results on the expressiveness of this model are presented in the next section. Practical aspects related to the parametrization of κ, v and g, as well as the model evaluation and training will be discussed in section 5.

4. Theoretical Guarantees of Universality

In this section we give conditions under which the LOCA model is universal. There exist multiple definitions of universality present in the literature, for example see Sriperumbudur et al. (2011). To be clear, we formally state the definition we use below.

Definition 1 (Universality) *Given compact sets $\mathcal{X} \subset \mathbb{R}^{d_x}$, $\mathcal{Y} \subset \mathbb{R}^{d_y}$ and a compact set $\mathcal{U} \subset C(\mathcal{X}, \mathbb{R}^{d_u})$ we say a class of operators $\mathcal{A} \ni \mathcal{F} : C(\mathcal{X}, \mathbb{R}^{d_u}) \to C(\mathcal{Y}, \mathbb{R}^{d_s})$ is* universal *if it is dense in the space of operators equipped with the supremum norm. In other words, for any continuous operator $\mathcal{G} : C(\mathcal{X}, \mathbb{R}^{d_u}) \to C(\mathcal{Y}, \mathbb{R}^{d_s})$ and any $\epsilon > 0$, there exists $\mathcal{F} \in \mathcal{A}$ such that*

$$\sup_{u \in \mathcal{U}} \sup_{y \in \mathcal{Y}} \|\mathcal{G}(u)(y) - \mathcal{F}(u)(y)\|^2_{\mathbb{R}^{d_s}} < \epsilon.$$

To explore the universality properties of our model we note that if we remove the softmax normalization and the kernel coupling, the evaluation of the model can be written as

$$\mathcal{F}(u)(y) = \sum_{i=1}^{n} g_i(y) \odot v_i(u).$$

The universality of this class of models has been proven in Chen and Chen (1995) (when $d_s = 1$) and extended to deep architectures in Lu et al. (2019). We will show that our model with the softmax normalization and kernel coupling is universal by adding these components back one at a time. First, the following theorem shows that the normalization constraint $\varphi(y) \in \prod_{k=1}^{d_s} \Delta^n$ does not reduce the approximation power of this class of operators.

Theorem 2 (Normalization Preserves Universality) *If $\mathcal{U} \subset C(\mathcal{X}, \mathbb{R}^{d_u})$ is a compact set of functions and $\mathcal{G} : \mathcal{U} \to C(\mathcal{Y}, \mathbb{R}^{d_s})$ is a continuous operator with $\mathcal{X}$ and $\mathcal{Y}$ compact, then for every $\epsilon > 0$ there exists $n \in \mathbb{N}$, functionals $v_{j,k} : \mathcal{U} \to \mathbb{R}$, for $j \in [n]$, $k \in [d_s]$, and functions $\varphi_j : \mathcal{Y} \to \mathbb{R}^{d_s}$ with $\varphi_j(y) \in [0,1]^{d_s}$ and $\sum_{j=1}^{n} \varphi_j(y) = 1_{d_s}$ for all $y \in \mathcal{Y}$ such that*

$$\sup_{u \in \mathcal{U}} \sup_{y \in \mathcal{Y}} \left\| \mathcal{G}(u)(y) - \mathbb{E}_{\varphi(y)}[v(u)] \right\|_{\mathbb{R}^{d_s}}^2 < \epsilon.$$

Proof The proof is given in Appendix B. ■

It remains to show that the addition of the kernel coupling step for the functions φ also does not reduce the approximation power of this class of operators. By drawing a connection to the theory of Reproducing Kernel Hilbert Spaces (RKHS), we are able to state the sufficient conditions for this to be the case. The key insight is that, under appropriate conditions on the kernel κ, the image of the integral operator in (1) is dense in an RKHS $\mathcal{H}_\kappa$ which itself is dense in $C(\mathcal{Y}, \mathbb{R}^n)$. This allows (2) to approximate any continuous function $\varphi : \mathcal{Y} \to \prod_{i=1}^{d_s} \Delta^n$ and thus maintains the universality guarantee of Theorem 2.

Proposition 3 (Kernel Coupling Preserves Universality) *Let $\kappa : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a positive definite and symmetric universal kernel with associated RKHS $\mathcal{H}_\kappa$ and define the integral operator*

$$\begin{aligned} T_\kappa : & C(\mathcal{Y}, \mathbb{R}^n) \to C(\mathcal{Y}, \mathbb{R}^n), \\ & f \mapsto \int_{\mathcal{Y}} \kappa(y, z) f(z) dz. \end{aligned}$$

If $\mathcal{A} \subseteq C(\mathcal{Y}, \mathbb{R}^n)$ is dense, then $T_\kappa(\mathcal{A}) \subset C(\mathcal{Y}, \mathbb{R}^n)$ is also dense.

The statement of Proposition 3 requires that the kernel κ be symmetric, positive definite, and universal. We next show that by construction it will always be symmetric and positive definite, and under an assumption on the feature map q it will additionally be universal.

Proposition 4 (Universality of the Kernel κ) *The kernel defined in (4) is positive definite and symmetric. Further, if q is injective, it defines a universal RKHS.*

Proof The proof is provided in Appendix D. ■

Lastly, we present a result showing that a particular architecture choice for the input feature encoder v also preserves universality. We show that if there is uniform convergence of spectral representations of the input, projections onto these representations can be used to construct a universal class of functionals on $C(\mathcal{X}, \mathbb{R}^{d_u})$.

Proposition 5 (Spectral Encoding Preserves Universality) *Let $\mathcal{A}_d \subset C(\mathbb{R}^d, \mathbb{R}^n)$ be a set of functions dense in $C(\mathbb{R}^d, \mathbb{R}^n)$, and $\{e_i\}_{i=1}^{\infty}$ a set of basis functions such that for some compact set $\mathcal{U} \subseteq C(\mathcal{X}, \mathbb{R}^{d_u})$, $\sum_{i=1}^{\infty} \langle u, e_i \rangle_{L^2} e_i$ converges to u uniformly over $\mathcal{U}$. Let $\mathcal{D}_d : \mathcal{U} \to \mathbb{R}^d$ denote the projection onto $\{e_1, \ldots, e_d\}$. Then for any continuous functional $h : \mathcal{U} \to \mathbb{R}^n$, and any $\epsilon > 0$, there exists d and $f \in \mathcal{A}_N$ such that*

$$\sup_{u \in \mathcal{U}} \left\| h(u) - f \circ \mathcal{D}_d(u) \right\| < \epsilon.$$

Proof The proof is provided in Appendix E. ■

For example, if our compact space of input functions $\mathcal{U}$ is contained in $C^1(\mathcal{X}, \mathbb{R}^{d_u})$, and $\mathcal{D}$ is a projection onto a finite number of Fourier modes, the architecture proposed in equation (6) is expressive enough to approximate any functional from $\mathcal{U} \to \mathbb{R}$, including those produced by the universality result stated in Theorem 2.

5. Implementation Aspects

To implement our method, it remains to make a choice of discretization for computing the integrals required for updating the KCA weights $\varphi(y)$, as well as a choice for the input function feature encoding $v(u)$. Here we address these architecture choices, and provide an overview of the proposed model's forward evaluation.

5.1 Computation of the Kernel Integrals

To compute the kernel-coupled attention weights $\varphi(y)$, we are required to evaluate integrals over the domain $\mathcal{Y}$ in (1) and (4). Adopting an unbiased Monte-Carlo estimator using P points $y_1, \ldots, y_P \in \mathcal{Y}$, we can use the approximations

$$\int_{\mathcal{Y}} \kappa(y, y') g(y') \approx \frac{\mathrm{vol}(\mathcal{Y})}{P} \sum_{i=1}^{P} \kappa(y, y_i) g(y_i),$$

for equation (1), and

$$\int_{\mathcal{Y}} k(q(y), q(z)) dz \approx \frac{\mathrm{vol}(\mathcal{Y})}{P} \sum_{i=1}^{P} k(q(y), q(y_i)),$$

for use in equation (4). Note that due to the normalization in κ, the $\mathrm{vol}(\mathcal{Y})$ term cancels out. In practice, we allow the query point y to be one of the points $y_1, \ldots, y_P$ used for the Monte-Carlo approximation.

When the domain $\mathcal{Y}$ is low dimensional, as in many physical problems, a Gauss-Legendre quadrature rule with weights w_i can provide an accurate and efficient alternative to Monte Carlo approximation. Using Q Gauss-Legendre nodes and weights, we can approximate the required integrals as

$$\int_{\mathcal{Y}} \kappa(y, y')g(y')dy' \approx \sum_{i=1}^{Q} w_i \kappa(y, y'_i)g(y'_i),$$

for equation (1) and

$$\int_{\mathcal{Y}} k(q(y), q(z))dz \approx \sum_{i=1}^{Q} w_i k(q(y), q(z_i)),$$

for use in equation (4).

If we restrict the kernel κ to be translation invariant, there is another option for computing these integrals. As in Li et al. (2020a), we could take the Fourier transform of both κ and g, perform a point-wise multiplication in the frequency domain, followed by an inverse Fourier transform. However, while in theory the discrete Fourier transformation could be performed on arbitrarily spaced grids, the most available and computationally efficient implementations rely on equally spaced grids. We prefer to retain the flexibility of arbitrary sets of query points y and will therefore not pursue this alternate approach. In Section 7, we will switch between the Monte-Carlo and quadrature strategies depending on the problem at hand.

5.2 Positional Encoding of Output Query Locations

We additionally adopt the use of positional encodings, as they have been shown to improve the performance of attention mechanisms. For encoding the output query locations, we are motivated by the positional encoding in Vaswani et al. (2017), the harmonic feature expansion in Di Leoni et al. (2021), and the work of Wang and Liu (2019) for implementing the encoding to more than one dimensions. The positional encoding for a one dimensional query space is given by

$$\begin{aligned} e(y^i, 2j + (i-1)H) &= \cos(2^j \pi y^i), \\ e(y^i, 2j + 1 + (i-1)H) &= \sin(2^j \pi y^i), \end{aligned} \tag{8}$$

where H the number of encoding coefficients, $j = 1, ..., H/2$, y^i the query coordinates in different spatial dimensions and $i = 1, ..., d_y$. In contrast to Vaswani et al. (2017) we consider the physical position of the elements of the set y as the position to encode instead of their index position in a given list, as the index position in general does not have a physically meaningful interpretation.

5.3 Wavelet Scattering Networks as a Spectral Encoder

While projections onto an orthogonal basis allows us to derive a universality guarantee for the architecture, there can be some computational drawbacks. For example, it is known that the Fourier transform is not always robust to small deformations of the input (Mallat, 2012). More worrisome is the lack of robustness to noise corrupting the input function. In real world applications it will often be the case that our inputs are noisy, hence, in practice we are motivated to find an operator $\mathcal{D}$ with stronger continuity with respect to these small perturbations.

To address the aforementioned issues, we make use of the scattering transform (Bruna and Mallat, 2013), as an alternate form for the operator $\mathcal{D}$. The scattering transform maps an input function to a sequence of values by alternating wavelet convolutions and complex modulus operations (Bruna and Mallat, 2013). To be precise, given a mother wavelet ψ and a finite discrete rotation group G, we denote the wavelet filter with parameter $\lambda = (r, j) \in G \times \mathbb{Z}$ as

$$\psi_\lambda(u) = 2^{d_x j}\psi(2^j r^{-1} x).$$

Given a path of parameters $p = (\lambda_1, \dots, \lambda_m)$, the scattering transform is defined by the operator

$$S[p]u = ||||u \star \psi_{\lambda_1}| \star \psi_{\lambda_2}| \cdots | \star \psi_{\lambda_m}| \star \phi(x), \tag{9}$$

where $\phi(x)$ is a low pass filter. We allow the empty path $\emptyset$ as a valid argument of S with $S[\emptyset]u = u \star \phi$. As shown in Bruna and Mallat (2013), this transform is Lipschitz continuous with respect to small deformations, while the modulus of the Fourier transform is not. This transform can be interpreted as a deep convolutional network with fixed filters and has been successfully applied in multiple machine learning contexts (Oyallon et al., 2017; Chang and Chen, 2015). Computationally, the transform returns functions of the form (9) sampled at points in their domain, which we denote by $\hat{S}[p](u)$.

By choosing d paths $p_1, \dots, p_d$, we may define the operator $\mathcal{D}$ as

$$\mathcal{D}(u) = \left(\hat{S}[p_1](u), \dots, \hat{S}[p_d](u)\right)^\top.$$

In practice, the number of paths used is determined by three parameters: J, the maximum scale over which we take a wavelet transform; L, the number of elements of the finite rotation group G, and m_0, the maximum length of the paths p. While Proposition 5 does not necessarily apply to this form of $\mathcal{D}$, we find that empirically this input encoding gives the best performance in combination with the KCA and normalization of attention weight functions φ. We perform an ablation study in Appendix F.3 and show that, while the scattering transform can help the performance of other models as well, it is not the only component responsible for the competitive performance of LOCA.

5.4 Loss Function and Training

The proposed model is trained by minimizing the empirical risk loss over the available training data pairs,

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{\ell=1}^{P}(s^i(y_\ell^i) - \mathcal{F}_\theta(u^i)(y_\ell^i))^2, \tag{10}$$

where $\theta = (\theta_q, \theta_f, \theta_g)$ denotes all trainable model parameters. This is the simplest choice that can be made for training the model. Other choices may include weighting the mean square error loss using the $\mathcal{L}_1$ norm of the ground truth output (Di Leoni et al., 2021; Wang et al., 2021b), or employing a relative $\mathcal{L}_2$ error loss (Li et al., 2020a). The minimization is performed via stochastic gradient descent updates, where the required gradients of the loss with respect to all the trainable model parameters can be conveniently computed via reverse-mode automatic differentiation.

5.5 Implementation Overview

Algorithm 1 provides an overview of the steps required for implementing the LOCA method. The training data set is first processed by passing the input functions through a wavelet scattering network (Bruna and Mallat, 2013), followed by applying a positional encoding to the query locations and the quadrature/Monte-Carlo integration points. The forward pass of the model is evaluated and gradients are computed for use with a stochastic gradient descent optimizer. After training, we make one-shot predictions for super-resolution grids, and we compute the relative $\mathcal{L}_2$ error between the ground truth output and the prediction.

Algorithm 1 Implementation summary of the LOCA method

```
Require:
• Input/output function pairs $\{u^i, s^i\}_{i=1}^N$.
• Query locations $y^i$ for evaluating $s^i$.
• Quadrature points $z^i$.
Pre-processing:
• Apply transformation (6) on the input function to get $\hat{u}$, the input features.
• Apply positional encoding (8) to query coordinates $y, z$, to get $\hat{y}, \hat{z}$.
• Choose the network architectures for functions $q_{\theta_q}$, $f_{\theta_f}$, and $g_{\theta_g}$.
• Initialize the trainable parameters $\theta = (\theta_q, \theta_f, \theta_g)$, and choose a learning rate $\eta$.
Training:
  for $i = 0$ to $I$ do
      Randomly select a mini-batch of $(\hat{u}, \hat{y}, \hat{z}, s)$.
      Evaluate $g_{\theta_g}(q_{\theta_q}(\hat{z}))$.
      Compute the Coupling Kernel $\kappa(q_{\theta_q}(\hat{y}), q_{\theta_q}(\hat{z}))$ (4).
      Numerically approximate the KCA (1) and compute $\varphi(y)$.
      Evaluate $f_{\theta_f}(\hat{u})$, as in Equation (6).
      Evaluate the expectation (7) and get $s^*$, the model prediction.
      Evaluate the training loss (10) and compute its gradients $\nabla_\theta \mathcal{L}(\theta_i)$.
      Update the trainable parameters via stochastic gradient descent: $\theta_{i+1} \leftarrow \theta_i - \eta \nabla_\theta \mathcal{L}(\theta_i)$.
  end for
```

6. Connections to Existing Operator Learning Methods

In this section, we provide some insight on the connections between our method and similar operator learning methods.

6.1 DeepONets

Note that if we identify our input feature map, $v(u)$, with the DeepONet's branch network, and the location dependent probability distribution, $\varphi(y)$, with the DeepONet's trunk network, then the last step of both models is computed the same way. We can recover the DeepONet architecture from our model under three changes to the architecture in the forward evaluation. First, we would remove the normalization step in the construction of φ. Next, we remove the KCA mechanism that is applied to the candidate score function g (equivalently we may fix the kernel κ to be δ-distributions along the diagonal). Finally, in the construction of the input feature map $v(u)$, instead of the scattering transform we would act on the input with a collection of δ distributions at the fixed sensor locations. These differences between DeepONets and LOCA result in increased performance of our model, as we will see in Section 7.

6.2 Neural Operators

The connection between Neural Operators and DeepONets has been presented in Kovachki et al. (2021b) and Kovachki et al. (2021a), where it is shown that a particular choice of neural operator architecture produces a DeepONet with an arbitrary trunk network and a branch network of a certain form. In particular, a Neural Operator layer has the form,

$$v^{(\ell+1)}(z) = \sigma\left(W^{(\ell)}v^{(\ell)}(z) + \int_{\mathcal{Z}} k^{(\ell)}(s,z)v^{(\ell)}(s)ds\right), \tag{11}$$

where here σ is a point-wise nonlinearity. It is shown in Kovachki et al. (2021b) that this architecture can be made to resemble a DeepONet under the following choices. First, set $W^{(\ell)} = 0$. Next, lift the input data to n tiled copies of itself and choose a kernel k that is separable in s and z. If the output of the layer is then projected back to the original dimension by summing the coordinates, the architecture resembles a DeepONet.

The correspondence between our model and DeepONets described above allows us to transitively connect our model to Neural Operators as well. We additionally note that the scattering transform component of our architecture can be viewed as a collection of multiple-layer Neural Operators with fixed weights. Returning to (11), when $W^{(\ell)} = 0$ for all ℓ, the forward pass of the architecture is a sequence of integral transforms interleaved with point-wise nonlinearities. Setting σ to be the complex modulus function and $k^{(\ell)}$ to be a wavelet filter ψ_{λ_ℓ} we may write

$$v^{(\ell+1)} = |v^\ell * \psi_\lambda|.$$

When we compose L of these layers together, we recover (9) up to the application of the final low pass filter (again a linear convolution)

$$v^{(L)} = ||||u * \psi_{\lambda_1}| * \psi_{\lambda_2}| \cdots | * \psi_{\lambda_L}|.$$

Thus, we may interpret the scattering transform as samples from a collection of Neural Operators with fixed weights. This connection between the scattering transform and convolutional neural architectures with fixed weights was noticed during the original formulation of the wavelet scattering transform by Bruna and Mallat (2013), and thus also extends to

Neural Operators via the correspondence between Neural Operators and (finite-dimensional) convolutional neural networks (Kovachki et al., 2021b).

We would like to emphasize that our application of a kernel transform is different than how it is used for a Neural Operator. While Neural Operators apply the kernel transformation to functions of the input, we are applying it to an auxiliary score function of the output function domain that does not depend on the input at all.

6.3 Other Attention-Based Architectures

Here we compare our method with two other recently proposed attention-based operator learning architectures. The first is the Galerkin/Fourier Transformer (Cao, 2021). This method operates on a fixed input and output grid, and most similarly represents the original sequence-to-sequence Transformer architecture (Vaswani et al., 2017) with different choices of normalization. As in the original sequence-to-sequence architecture, the attention weights are applied across the indices (sensor locations) of the input sequence. By contrast, in our model the attention mechanism is applied to a finite-dimensional feature representation of the input that is not indexed by the input function domain. Additionally, our attention weights are themselves coupled over the domain $\mathcal{Y}$ via the KCA mechanism (2) as opposed to being defined over the input function domain in an uncoupled manner.

A continuous attention mechanism for operator learning was also proposed as a special case of Neural Operators in Kovachki et al. (2021b). There, it was noted that if the kernel in the Neural Operator was (up to a linear transformation) of the form

$$k(v(x), v(y)) = \left(\int \exp\left(\frac{\langle Av(s), Bv(y) \rangle}{\sqrt{m}} \right) ds \right)^{-1} \exp\left(\frac{\langle Av(x), Bv(y) \rangle}{\sqrt{m}} \right),$$

with $A, B \in \mathbb{R}^{m \times n}$, then the corresponding Neural Operator layer can be interpreted as the continuous generalization of a transformer block. Further, upon discretization of the integral this recovers exactly the sequence-to-sequence discrete Transformer model.

The main difference of this kind of continuous transformer with our approach is again how the attention mechanism is applied to the inputs. The Neural Operator Transformer is similar to the Galerkin/Fourier Transformer in the sense that the attention mechanism is applied over the points of the input function itself, whereas our model first creates a different finite dimensional feature representation of the input function which the attention is applied to. We note that our model does make use of attention weights defined over a continuous domain, but it is the domain of the output functions $\mathcal{Y}$ as opposed to $\mathcal{X}$.

6.4 Existing Error Estimate Frameworks

Recent work in Lanthaler et al. (2022) and Kovachki et al. (2021a) has explored error estimates for DeepONets and pseudo-spectral Fourier Neural Operators (Ψ-FNO), respectively. Note that the Ψ-FNO is a modification to the original FNO architecture introduced to facilitate the analysis in Kovachki et al. (2021a). In both cases a fundamental lower bound to the error of the architecture is derived in terms of a subspace of functions containing the outputs of the operator network. If $\mathcal{F}$ is an operator approximating $\mathcal{G}$ and $\mathcal{F}$ always produces functions that lie in some subspace $V \subset \mathcal{H}$, where $\mathcal{H}$ is a Hilbert space of functions from $\mathcal{Y} \to \mathbb{R}^{d_s}$, then there is a lower bound for the worst case error (see Kovachki et al. (2021a))

$$\sup_{u\in\mathcal{U}} \|\mathcal{F}(u) - \mathcal{G}(u)\|_{\mathcal{H}} \geq \sup_{u\in\mathcal{U}} \|P_V\mathcal{G}(u) - \mathcal{G}(u)\|_{\mathcal{H}}, \tag{12}$$

where $P_V : \mathcal{H} \to \mathcal{H}$ is the projection operator onto the subspace V. If the input functions are sampled from a probability measure μ on $C(\mathcal{X}, \mathbb{R}^{d_u})$ then an analogous result is shown in an $L^2(\mu)$ sense (see Lanthaler et al. (2022)),

$$\|\mathcal{F} - \mathcal{G}\|_{L^2(\mu)} \geq \|P_V \circ \mathcal{G} - \mathcal{G}\|_{L^2(\mu)}. \tag{13}$$

In the DeepONet case, the subspace V is spanned by the trunk functions. For the (pseudo-spectral) Fourier Neural Operator, this subspace is spanned by the first N trigonometric polynomials. These results are intuitive in the sense that the approximation error for a particular output is lower bounded by the minimum distance from the image of the operator network to the target function. This lower bound applies to LOCA as well, where the subspace V is given by the span of the attention weight functions $\{\varphi_i\}_{i=1}^n$.

In the L^2 case (13), it was shown in Lanthaler et al. (2022) that if the distribution of output functions has zero mean and finite second moment, the n-dimensional subspace V which minimizes the lower bound is given by the leading eigenspace of the covariance operator for this distribution. Moreover, if this covariance operator has eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots$, the overall error has the lower bound

$$\|\mathcal{F} - \mathcal{G}\|_{L^2(\mu)} \geq \sqrt{\sum_{k>n} \lambda_k}, \tag{14}$$

which quantifies the error incurred from the eigenspaces of the covariance that cannot be accessed by the output of the model.

This gives a notion of an optimal subspace of output functions for the image of the operator learning architecture. We find that the inclusion of the KCA mechanism and the normalization in the φ_i is able to create subspaces V that are closer in Grassmann distance to this optimal subspace. More details on this notion of optimality and supporting experiments on a synthetic example will be presented in Section 7.4.

The results in Lanthaler et al. (2022) were derived within the following interpretation of operator learning models, which applies directly to both LOCA and the DeepONet. These models are constructed from three components. First, an encoding map $\mathcal{E} : L^2(\mathcal{X}) \to \mathbb{R}^m$ maps the input function to a finite set of features. Then, an approximation map $\mathcal{A} : \mathbb{R}^m \to \mathbb{R}^n$ transforms these features. Finally, an affine reconstruction map $\mathcal{R}$ takes the image of the approximation map and embeds it into $L^2(\mathcal{Y})$ as follows

$$\begin{aligned} \mathcal{R} :& \mathbb{R}^n \longrightarrow L^2(\mathcal{Y}), \\ & \beta \longmapsto \sum_{i=1}^n \beta_i \tau_i, \end{aligned}$$

where $\tau_i \in L^2(\mathcal{Y})$. For the DeepONet, the map $\mathcal{E}$ is a pointwise evaluation at m sensors, while the maps τ_i are given by the trunk network. For LOCA, the map $\mathcal{E}$ is given by a wavelet scattering transform, and the maps τ_i correspond to the kernel-coupled attention weights,

$$\tau_i = \varphi_i = \sigma\left(\int_{\mathcal{Y}} k(z,y)g(z)\,dz\right)_i.$$

Another fundamental limitation of operator learning architectures is how the number parameters scale with respect to the desired error. In general, this scaling is exponential and referred to as the "curse of dimensionality". This has been shown for DeepONets (Lanthaler et al., 2022) and Fourier Neural Operators (Kovachki et al., 2021a), though in both cases there are particular PDE-based operator learning problems where this unfavorable scaling can be improved (by showing the architectures can emulate known solvers for which can be emulated by relatively small networks). These results can likely be replicated for LOCA as well, though this is outside the scope of the current paper and is a topic for future research.

7. Experimental Results

In this section we provide a comprehensive collection of experimental comparisons designed to assess the performance of the proposed LOCA model against two state of the art operator learning methods, the Fourier Neural Operator (FNO) (Li et al., 2020a) and the DeepONet (DON) (Lu et al., 2019). We will show that our method requires less labeled data than competing methods, is robust against noisy data and randomness in the model initialization, has a smaller spread of errors over testing data sets, and is able to successfully generalize in out-of-distribution testing scenarios. Evidence is provided for the following numerical experiments, see Figure 3 for a visual description.

- **Antiderivative:** Learning the antiderivative operator given multi-scale source terms.
- **Darcy Flow:** Learning the solution operator of the Darcy partial differential equation, which models the pressure of a fluid flowing through a porous medium with random permeability.
- **Mechanical MNIST:** Learning the mapping between the initial and final displacement of heterogeneous block materials undergoing equibiaxial extension.
- **Shallow Water Equations:** Learning the solution operator for a partial differential equation describing the flow below a pressure surface in a fluid with reflecting boundary conditions.
- **Climate modeling:** Learning the mapping from the air temperature field over the Earth's surface to the surface air pressure field, given sparse measurements.

Finally, we show that, in an experiment where the input function probability measure μ and associated true pushforward measure $\mathcal{G}_{\#}\mu$ are known analytically, the KCA mechanism and scattering transform allow LOCA to better learn the principal eigenspaces of the covariance for the output function distribution $\mathcal{G}_{\#}\mu$, as measured by a normalized Grassmann distance.

For all experiments the training data sets will take the following form. For each of the N input/output function pairs, (u^i, s^i), we will consider m discrete measurements of each input function at fixed locations, $(u^i(x_1^i), \ldots, u^i(x_m^i))$, and M available discrete measurements of each output function $(s^i(y_1^i), \ldots, s^i(y_M^i))$, with the query locations $\{y_\ell^i\}_{\ell=1}^M$ potentially varying over the data set. Out of the M available measurement points $\{y_\ell^i\}_{\ell=1}^M$ for each output function s^i, we consider the effect of taking only P of these points for each input/output pair. For example, if we use 10% of labeled data, we set $P = \lfloor M/10 \rfloor$ and build a training data set where each example is of the form $(\{u^i(x_j^i)\}_{j=1}^m, \{s^i(y_\ell)\}_{\ell=1}^P)$. We round the percentages to the nearest integer or half-integer for clarity. We present details on the input and output data construction, as well as on the different problem formulations in Section F.7 of the Appendix.

In each scenario the errors are computed between both the models output and ground truth at full resolution. Throughout all benchmarks, we employ Gaussian Error Linear unit activation functions (GELU) (Hendrycks and Gimpel, 2016), and initialize all networks using the Glorot normal scheme (Glorot and Bengio, 2010). All networks are trained via mini-batch stochastic gradient descent using the Adam optimizer with default settings (Kingma and Ba, 2014). The detailed hyper-parameter settings, the associated number of parameters for all examples, the computational cost, and other training details are provided in Appendix F.2. All code and data accompanying this manuscript will be made publicly available at `https://github.com/PredictiveIntelligenceLab/LOCA`.

7.1 Data Efficiency

In this section we investigate the performance of our model when the number of labeled output function points is small. In many applications labeled output function data can be scarce or costly to obtain. Therefore, it is desirable that an operator learning model is able to be successfully trained even without a large number of output function measurements. We investigate this property in the Darcy flow experiment by gradually increasing the percentage of labeled output function measurements used per input function example. Next, we compare the performance of all models for the Shallow Water benchmark in the small data regime. Lastly, we demonstrate that the proposed KCA weights provide additional training stability specifically in the small data regime. One important aspect of learning in the small data regime is the presence of outliers in the error statistics, which quantify the worst-case-scenario predictions. In each benchmark we present the following error statistics across the testing data set: the error spread around the median, and outliers outside the third quantile.

Figure 4 shows the effect of varying the percentage of labeled output points used per training example in the Darcy flow prediction example. The box plot shows the distribution of errors over the test data set for each model. We see that the proposed LOCA model is able to achieve low prediction errors even with 1.5% of the available output function measurements per example. It also has a consistently smaller spread of errors with fewer outliers across the test data set in all scenarios. Moreover, when our model has access to 6% of the available output function measurements it achieves lower errors against both the DON and FNO trained with any percentage (up to 100%) of the total available labeled data.

Figure 6 shows the spread of errors across the test data set for the Shallow Water benchmark when the LOCA model is trained on 2.5% of the available labeled data per input-output function pair. We observe that our model outperforms DON and FNO in

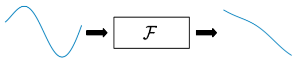
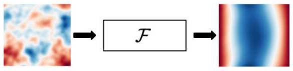
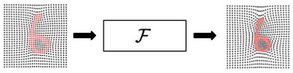
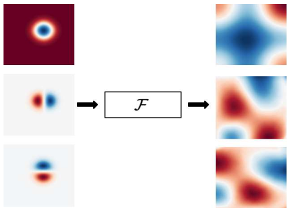
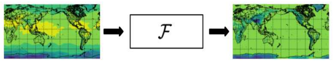
Example	Input Function $u(x)$	Output Function $s(y)$	Visualization of the operator $\mathcal{F}$
Antiderivative	Smooth Univariate Function $u(x) \sim GP(0, o \exp\left(\frac{\|x-x'\|}{l}\right)$	Antiderivative solution $\frac{ds(x)}{dx} = u(x), \quad s(x) = s_0 + \int_0^x u(\tau)d\tau$	
Darcy Flow	Random permeability $u_0 \sim \mathcal{N}(0, 7^{3/2}(-\Delta + 49I)^{1.5})$ $u(x) = \exp(u_0 cos(x))$	Pressure field $\nabla \cdot (u(x)\nabla s(x)) = f(x)$	
Mechanical MNIST	Initial displacement vector field $v_1(x; d_1), v_2(x; d_1)$	Later displacement vector field $v_1(x; d_2), v_2(x; d_2)$	
Shallow Water Equations	Random initial condition $\rho(0, x_1, x_2) = 1 + h\exp^{-((x_1-\xi)^2+(x_2-\zeta)^2)/w}$ $v_1(0, x_1, x_2) = v_1(dt, x_1, x_2),$ $v_2(0, x_1, x_2) = v_2(dt, x_1, x_2)$ $h = \mathcal{U}(1.5, 2.5)$ $w = \mathcal{U}(0.002, 0.008)$ $\xi = \mathcal{U}(0.4, 0.6)$ $\zeta = \mathcal{U}(0.4, 0.6)$	Solution at specified time instances $\frac{\partial \overrightarrow{U}}{\partial t} + \frac{\partial \overrightarrow{F}}{\partial x_1} + \frac{\partial \overrightarrow{G}}{\partial x_2} = 0$ $\overrightarrow{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \end{pmatrix}, \quad \overrightarrow{F} = \begin{pmatrix} \rho u \\ \rho u^2 + \frac{1}{2}g\rho^2 \\ \rho uv \end{pmatrix}, \quad \overrightarrow{G} = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + \frac{1}{2}g\rho^2 \end{pmatrix}$	
Climate Modeling	Surface air temperature $T(x)$	Surface air pressure $P(y)$	

Figure 3: A schematic visualization of the operator learning benchmarks considered in this work. Shown are the input/output function and a description of their physical meaning, as well as the operator that we learn for each example. In the Mechanical MNIST example, for visual clarity we do not present the map that the model is actually learning, which is the displacement in the vertical and the horizontal directions, but the position of each pixel under a specified displacement. See Appendix Section F.7 for more details on the data set generation.

predicting the wave height, ρ, and provides similar errors to the FNO for the two velocity components, v_1 and v_2. Despite the fact that the two methods perform in a similar manner for the median error, LOCA consistently provides a much smaller standard deviation of errors across the test data set, as well as far fewer outliers.

We hypothesize that the KCA mechanism is responsible for good performance of LOCA observed in the small labelled data regime. An interpretation for this phenomenon is that the kernel coupling allows evaluations of the score network g over the entire query domain to be used in estimating gradients with respect to its parameters, even when the number of query points is small. When a small number of output function measurements are available, the empirical loss function is approximating (up to a constant) the L^2 error of an output function to a target function with a small sum over the observed query locations $\{y_1, \dots, y_P\}$,

$$\|\mathcal{F}(u) - s\|_{L^2}^2 \propto \sum_{k=1}^{P} \|\mathcal{F}(u)(y_k) - s(y_k)\|^2 = \sum_{k=1}^{P} \left\| \sum_{i=1}^{n} v(u)_i \varphi(y_k)_i - s(y_k) \right\|^2. \tag{15}$$

Ignoring the softmax normalization for now, if $\varphi_i = g_i$ without the kernel coupling, the gradient of the loss with respect to the network parameters of g_i^θ, denoted θ, will only use information from the network g_i at the few query locations $\{y_1, \dots, y_p\}$,

$$\nabla_\theta \varphi(y_k)_i = \nabla_\theta g^\theta(y_k)_i. \tag{16}$$

However, if $\varphi(y_k)_i$ is given as a kernel convolution of $g(y_k)_i$, then the gradient with respect to the network parameters θ will use information about the functional output of g^θ across the query domain $\mathcal{Y}$,

$$\nabla_\theta \left(\int_{\mathcal{Y}} k(y_k, z) g^\theta(z)_i \, dz \right) = \int_{\mathcal{Y}} k(y_k, z) \nabla_\theta g^\theta(z)_i \, dz. \tag{17}$$

This additionally has the effect of allowing the gradients $\nabla_\theta \varphi_i$ to inherit the regularity properties of the kernel resulting in smoother gradients over the varying queries. When there are few available queries for training, this smoother variation in $\nabla_\theta \varphi_i(y)$ over $y \in \mathcal{Y}$ can reduce the noise in the gradients of (15).

To empirically test the effect of the KCA mechanism, we train LOCA on the Darcy flow problem with varying amounts of input/output function samples $N = [100, 200, 500]$ and output measurements $P = [32, 64, 128, 256, 512, 1024]$ both with and without the KCA mechanism, while considering the same architecture as above. We present the results in the form of a checkerboard for the case where the KCA is considered and the case where no KCA is considered inside the softmax function, in Figure 5. We observe in Figure 5 that the model presents higher maximum error for all cases, starting with larger differences for small N, around 4% and moving to smaller differences for larger N around 2%. There are two differences in this experiment compared to the Darcy case above, the first is that we train the model while considering full batch stochastic gradient descent to avoid possible noise in the gradients induced by the choice of the batch and we train for $5,000$ iterations in order to avoid possible over-fitting of the models in the small data regime.

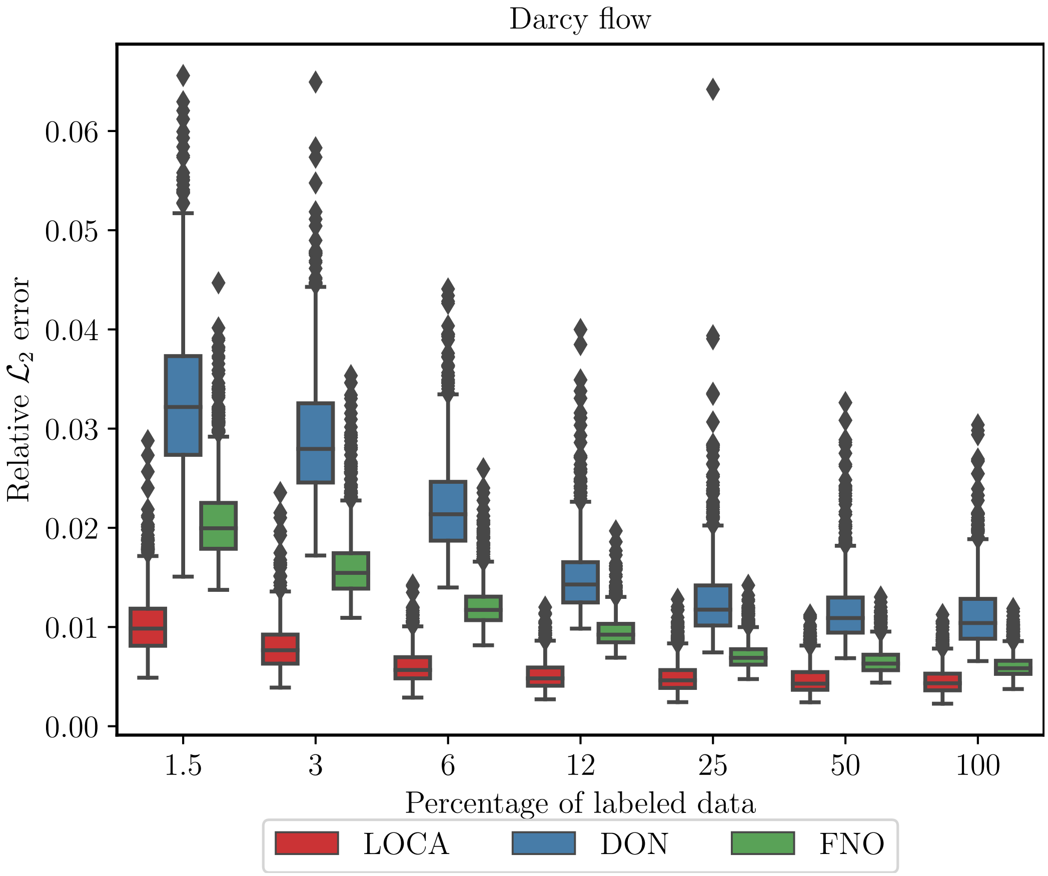


Figure 4: (Data Efficiency) Relative $\mathcal{L}_2$ error boxplots for the solution of Darcy flow: We present the error statistics for the case of the Darcy flow in the form of boxplots for the case where we train on $[1.5, ..., 100]\%$ of the available output function measurements per example. We observe that our model presents fast convergence to a smaller median error than the other methods and the error spread is more concentrated around the median with fewer outliers.

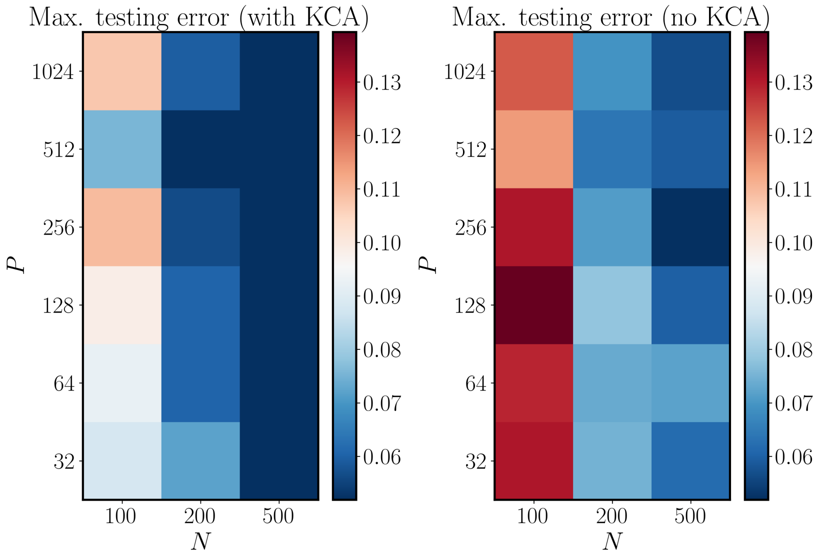


Figure 5: (Data Efficiency) Maximum relative testing $\mathcal{L}_2$ error for the solution of Darcy flow for different number of input/output function samples and output measurements when considering and not considering the KCA mechanism: We present the both the training (right) and testing (left) mean errors for the Darcy flow for $N = [100, 200, 500]$ and $P = [32, 64, 128, 256, 512, 1024]$.

7.2 Robustness

Operator learning can be a powerful tool for cases where we have access to clean simulation data for training, but wish to deploy the model on noisy experimental data. Alternatively, we may have access to noisy data for training and want to make predictions on noisy data as well. We will quantify the ability of our model to handle noise in the data by measuring the percentage increase in mean error clean to noisy data scenarios. For all experiments in this section, we consider 7% of the available labeled data.

We use the Mechanical MNIST benchmark to investigate the robustness of our model with respect to noise in the training and testing data. We consider three scenarios: one where the training and the testing data sets are clean, one where the training data set is clean, but the output data set is corrupted Gaussian noise sampled from $\mathcal{N}(0, .15I)$, and one where both the input and the output data sets are corrupted by Gaussian noise sampled from $\mathcal{N}(0, .15I)$. In Figure 7 we present the distribution of errors across the test data set for each noise scenario. We observe that for the case where both the training and the testing data are clean, the FNO achieves the best performance. In the scenario where the training data set is clean but the testing data set is noisy, we observe a percentage increase to the approximation error of all methods.

For the Clean to Noisy scenario the approximation error of the FNO method is increased by $1,930\%$ and $2,238\%$ for the displacement in the horizontal and vertical directions, respectively. For the DON method, the percentage increase is 112% and 96% for the displacement in the horizontal and vertical directions (labeled as v_1 and v_2), respectively.

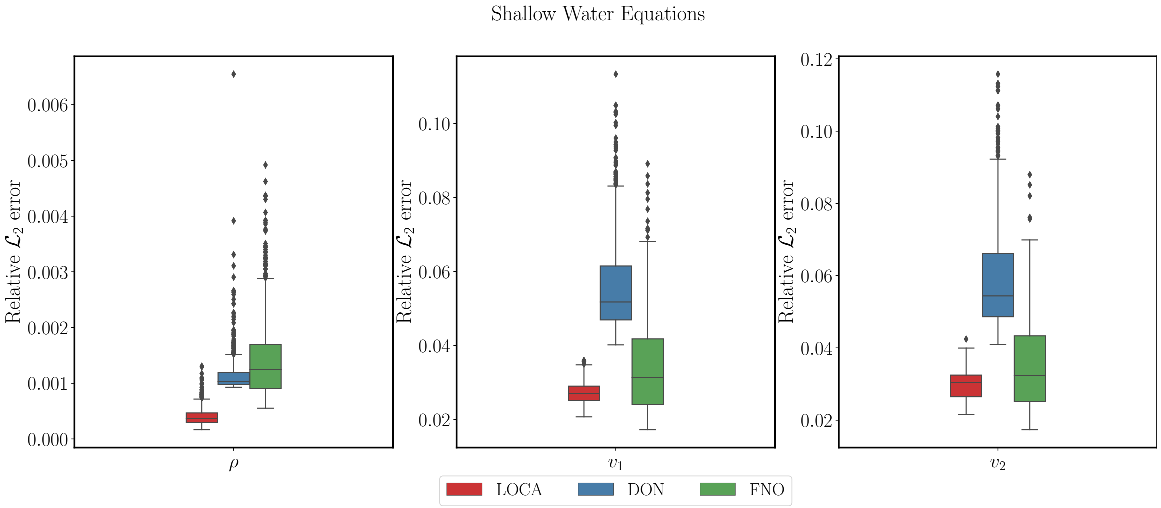


Figure 6: (Data Efficiency) Relative $\mathcal{L}_2$ error boxplots for the solution of the Shallow Water equations: We present the errors for each different predicted quantity of the Shallow Water equations. On the left, we present the ρ quantity which is the height of the water, and v_1 and v_2 which are the two components of the fluid velocity vector. We observe that LOCA achieves higher accuracy, and presents fewer outliers and more concentrated error spread compared its competitors.

For the LOCA method the percentage increase is 80% and 85% for the displacement in the horizontal and vertical directions, respectively. For the Noisy to Noisy scenario the approximation error of the FNO method is increased by 280% and 347% for the displacement in the horizontal and vertical directions, respectively. For the DON method, the percentage increase is 128% and 120%, and for LOCA is only 26% and 25% for each displacement component, respectively. We present the mean prediction error for each scenario and the corresponding percentage error increase in Table 1.

We observe that even though the FNO is very accurate for the case where both training and test data sets are clean, a random perturbation of the test data set can cause a huge decrease in accuracy. On the other hand, even though the DON method presents similar accuracy as our model in the clean to clean case, the standard deviation of the error is greater and its robustness to noise is inferior. LOCA is clearly superior in the case where the testing data are corrupted with Gaussian noise. We again emphasise that the metric in which we assess the performance is not which method has the lowest relative prediction error, but which method presents the smallest percentage increase in the error when noise exists in testing (and training in the case of Noisy to Noisy) data compared to the case where there exist no noise.

Next, we examine the variability of the models' performance with respect to the random initialization of the network parameters. We consider the Mechanical MNIST benchmark where the input data is clean but the output data contain noise. We train each model 10 times with different random seeds for initialization and record the maximum error in each

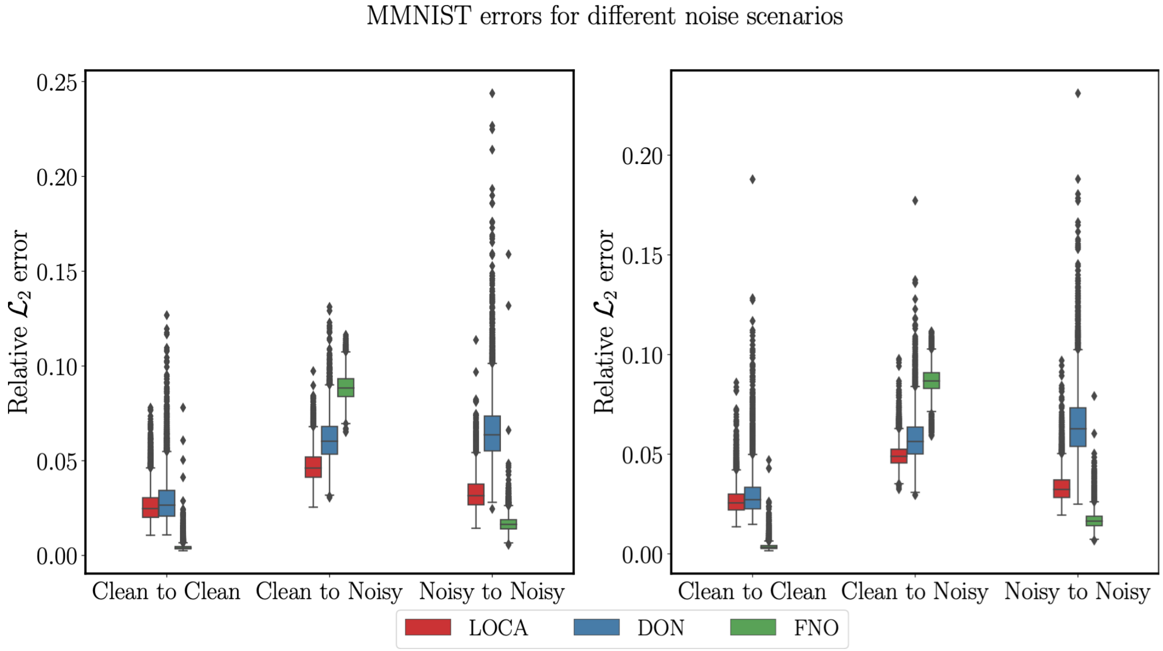


Figure 7: (Robustness) Relative $\mathcal{L}_2$ error boxplots for the Mechanical MNIST benchmark with noisy data: The left figure gives the distribution of errors for the displacement in the horizontal axis, v_1, and the right figure gives the displacement in the vertical axis v_2. For all cases we consider 7% of the whole training data set as labeled data used during training.

case. In Figure 8 we present the distribution of maximum prediction errors under different random seeds for the displacement in horizontal and vertical directions, respectively. We observe that LOCA displays a smaller spread of error for the case of displacement in the horizontal direction, v_1, and similar performance to the FNO for the case of displacement in the vertical direction, v_2.

7.3 Generalization

The ultimate goal of data-driven methods is to perform well outside of the data set they are trained on. This ability to generalize is essential for these models to be practically useful. In this section we investigate the ability of our model to generalize in three scenarios. We first consider an extrapolation problem where we predict the daily Earth surface air pressure from the daily surface air temperature. Our training data set consists of temperature and pressure measurements from 2000 to 2005 and our testing data set consists of measurements from 2005 to 2010. In Figure 9, we present the results for the extrapolation problem when considering 4% of the available pressure measurements each day for training. We observe that our method achieves the lowest error rates while also maintaining a small spread of these errors across the testing data set. While the DON method achieves a competitive

Noise scenario and error \ Model	FNO	DON	LOCA
Clean to Clean (CC)	**[0.004, 0.003]**	[0.028, 0.029]	[0.026, 0.026]
Clean to Noisy (CN)	[0.088, 0.087]	[0.061, 0.057]	**[0.047, 0.049]**
Noisy to Noisy (NN)	**[0.016, 0.016]**	[0.065, 0.064]	[0.032, 0.033]
Percentage error increase from CC to CN	[1,930 %, 2,238 %]	[112%, 96%]	**[80%, 85%]**
Percentage error increase from CC to NN	[280 %, 347%]	[128%,120%]	**[26%, 25%]**

Table 1: (Robustness) Mechanical MNIST prediction error with noisy data: The first three rows present the mean relative $\mathcal{L}_2$ errors of the vertical and horizontal displacements $[\text{Error}(v_1), \text{Error}(v_2)]$. The last two rows show the percentage increase in mean error from the noiseless case to the scenarios where the testing input data is corrupted by noise and to the scenario that both the training and testing input data sets are corrupted by noise. For each case we consider 7% of the total data set as labeled data for training. We observe that our method shows the least percentage increase for each noise scenario.

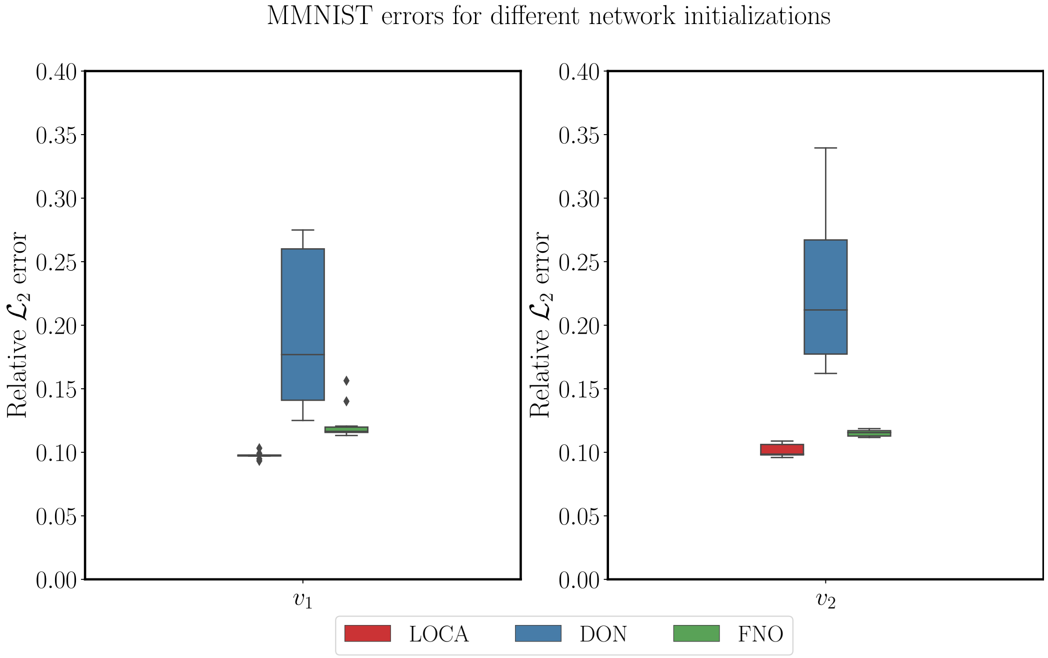


Figure 8: (Robustness) Maximum relative $\mathcal{L}_2$ error boxplots for Mechanical MNIST with over random model initializations: The left and right subplots show the distribution of maximum errors over the testing data set for the horizontal and vertical displacements, respectively. We consider 7% of the available output function measurments for training and run the model for 10 different random initializations. We observe that our method shows better performance than the other methods for both parameters v_1 and v_2.

performance with respect to the median error, the error spread is larger than both LOCA and FNO with many outliers.

Next, we examine the performance of our model under a distribution shift of the testing data. The goal of the experiment is to learn the antiderivatve operator where the training and testing data sets are sampled from a Gaussian process. We fix the length-scale of the testing distribution at 0.1 and examine the effect of training over 9 different data sets with length-scales ranging from 0.1 to 0.9. In Figure 10, we present the error on the testing data set after being trained on each different training data set. The error for each testing input is averaged over 10 random network initializations. We observe that while the LOCA and FNO methods present a similar error for the first two cases, the FNO error is rapidly increasing. On the other hand, the DON method while presenting a larger error at first, eventually performs better than the FNO as the training length-scale increases. We find that LOCA outperforms its competitors for all cases.

Lastly, we examine the performance of the three models when the training and testing data set both contain a wide range of scale and frequency behaviors. We consider this set-up as a simple model of a multi-task learning scenario and we want to explore the generalization capabilities of our model for this case. We construct a training and testing data set by sampling inputs from a Gaussian process where the length-scale and amplitude are chosen over ranges of 2 and 4 orders of magnitude, respectively. In Figure 11, we present samples from the input distribution, the corresponding output functions, and the distribution of errors on the testing data set. We observe that our method is more accurate and the error spread is smaller than DON and Fourier Neural Operators. While the FNO method shows a median that is close to the LOCA model, there exist many outliers that reach very high error values.

7.4 Optimal Subspace Alignment

In this section we will investigate the magnitude of the optimal error lower bounds presented in Section 6.4. In particular, we consider a scenario where the input functions are drawn from a probability distribution μ on $L^2(\mathcal{X}, \mathbb{R}^{d_u})$ and an operator $\mathcal{G} : L^2(\mathcal{X}, \mathbb{R}^{d_u}) \to L^2(\mathcal{Y}, \mathbb{R}^{d_s})$. As explained in Section 6.4, given an operator learning architecture of the form

$$\mathcal{F}(u)(y) = \sum_{i=1}^{n} \varphi_i(y) \odot v_i(u),$$

if $V = \text{span}\{\varphi_1, \dots, \varphi_n\}$ and $P_V : L^2(\mathcal{Y}, \mathbb{R}^{d_s}) \to L^2(\mathcal{Y}, \mathbb{R}^{d_s})$ is the projection onto the subspace V, then the $L^2(\mu)$ error satisfies the following lower bound

$$\|P_V \circ \mathcal{G} - \mathcal{G}\|_{L^2(\mu)} \leq \|\mathcal{F} - \mathcal{G}\|_{L^2(\mu)}. \tag{18}$$

This lower bound motivates the question: for fixed n what is the choice of $V = \text{span}\{\varphi_1, \dots, \varphi_n\}$ which minimizes the lower bound in (18)? It was shown in Lanthaler et al. (2022) that, if the pushforward measure $\mathcal{G}_{\#}\mu$ has a finite second moment and covariance operator $\Gamma : L^2(\mathcal{Y}, \mathbb{R}^{d_s}) \to L^2(\mathcal{Y}, \mathbb{R}^{d_s})$, the optimal choice of V is given by the leading n-dimensional eigenspace of Γ. That is, if $\{\lambda_i, \phi_i\}_{i=1}^{\infty}$ with $\lambda_1 \geq \lambda_2 \geq \dots$ gives an orthonormal

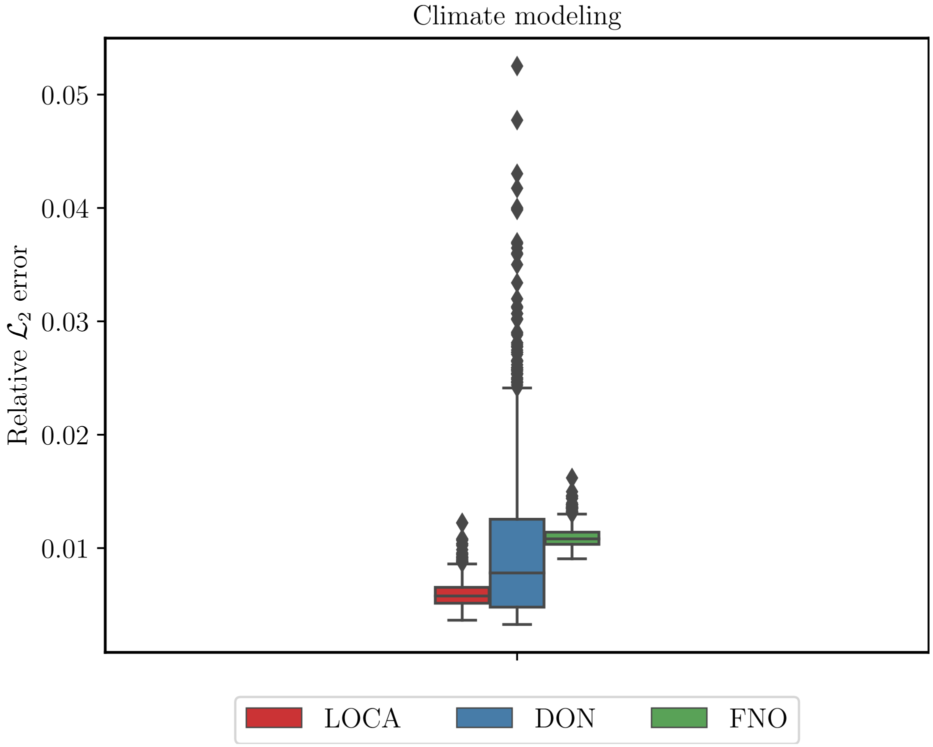


Figure 9: (Generalization) Relative $\mathcal{L}_2$ error boxplots for the climate modeling experiment: We present the errors for the temperature prediction task on the testing data set. We consider 4% of the whole data set as labeled data used for training. We observe that our method performs better than the other methods both with respect to the median error and the error spread.

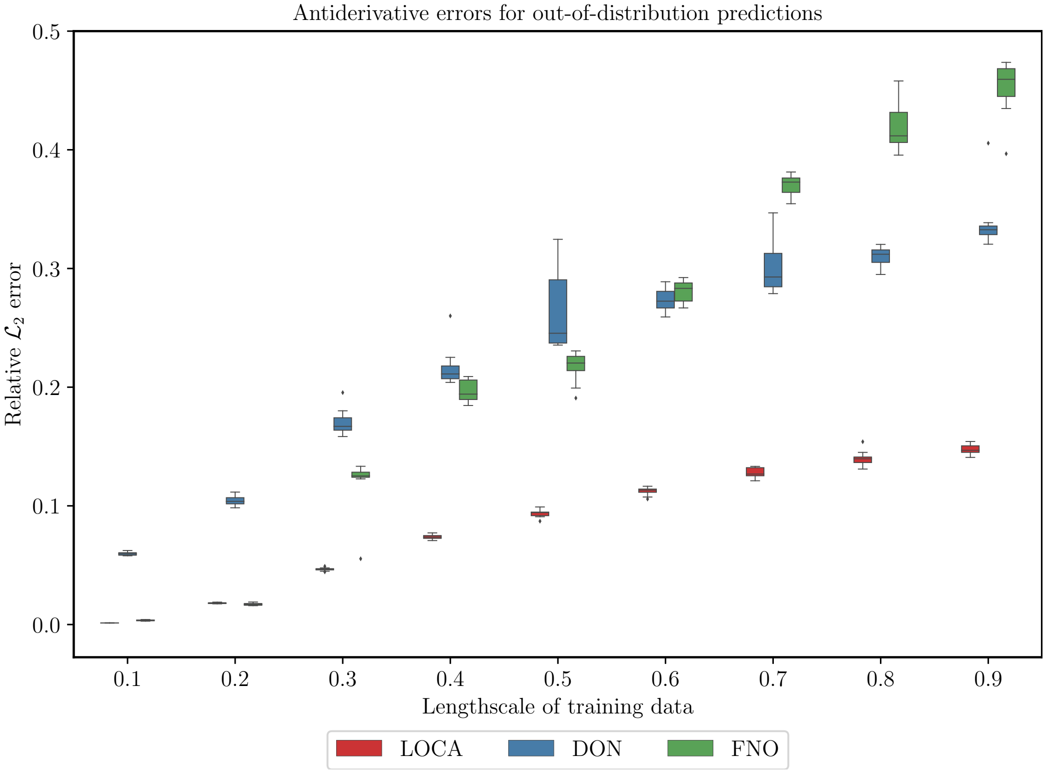


Figure 10: (Generalization) Antiderivative relative $\mathcal{L}_2$ error boxplots for out-of distribution testing sets: We show the performance of all models when trained on increasingly out of distribution data sets from the testing set. We use all available output function measurements for training.

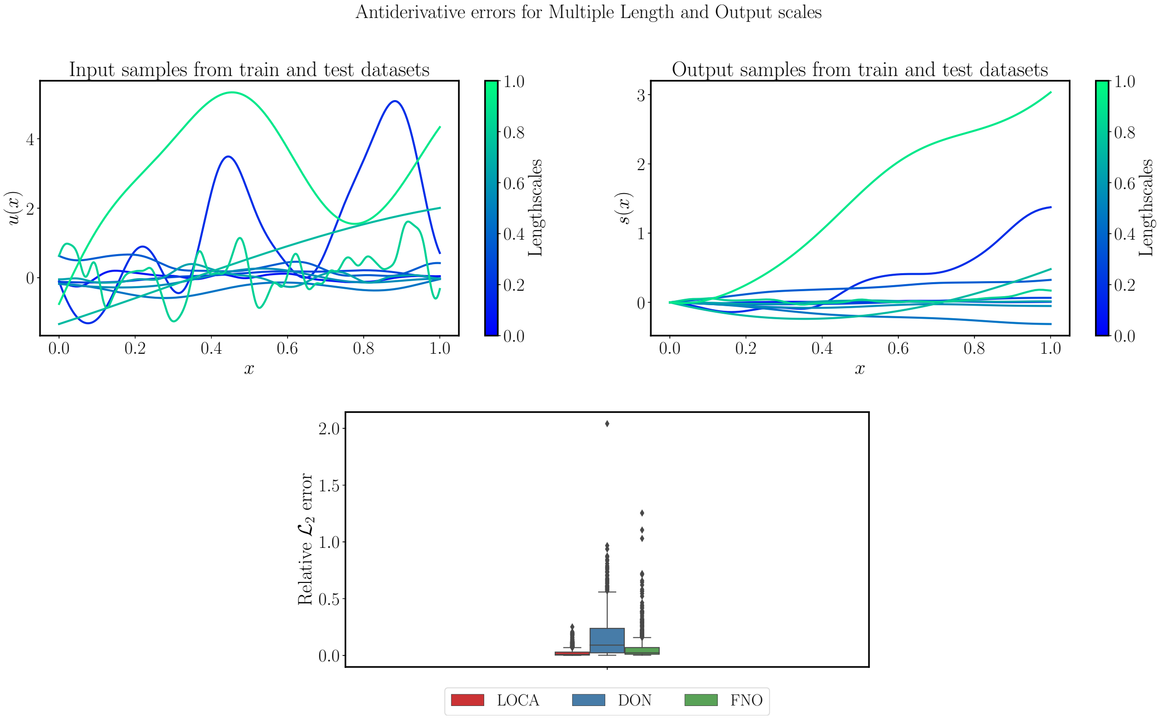


Figure 11: (Generalization) Antiderivative relative $\mathcal{L}_2$ error boxplots given input functions with multiple lenghtscales and amplitudes: We present samples of the input and output functions from the testing data set in the top left and right figures, respectively, as well as the test error boxplots for each method, bottom figure.

eigendecomposition for the covariance operator

$$\Gamma = \sum_{i=1}^{\infty} \lambda_i \phi_i \otimes \phi_i,$$

then the choice of V which minimizes the lower bound in (18) is given by $\hat{V} := \operatorname{span}\{\phi_1, \dots, \phi_n\}$. In the case where we can identify $\hat{V}$, we may examine how well the span of the functions $\{\varphi_i\}_{i=1}^n$ aligns with the optimal subspace $\hat{V}$ as a quantitative measure of how well the operator learning architecture has identified an optimal finite dimensional subspace to represent the output functions from $\mathcal{G}_{\#}\mu$.

To measure the alignment of two subspaces V and W of dimension n, we choose to use the geodesic Grassmann distance. To define this, we define the first principal angle $\cos\theta_1$ between two subspaces as

$$\cos\theta_1 := \max_{\substack{v\in V, w\in W \\ \|v\|=\|w\|=1}} \; v^T w, \tag{19}$$

and the remaining principle angles $\cos\theta_{i+1}$ are defined inductively as the solution to the above problem over the subspaces $V_i = V \cap \operatorname{span}\{v_1, \dots, v_i\}^{\perp}$ and $W_i = W \cap \operatorname{span}\{w_1, \dots, w_i\}^{\perp}$, where v_i and w_i are the minimizing arguments for $\cos\theta_i$. The geodesic Grassmann distance is then defined as

$$d(V, W) = \sqrt{\theta_1^2 + \ldots + \theta_n^2}. \tag{20}$$

Note that the largest distance two n-dimensional subspaces can have from each other is if they are completely orthogonal, in which case $d(V, W) = \sqrt{n}\pi/2$, and two subspaces are identical if and only if all principle angles are zero and $d(V, W) = 0$. In our experiments we normalize this distance by dividing by its maximum value, $\sqrt{n}\pi/2$. For more details on the geodesic Grassmann distance see Edelman et al. (1998). Details on how we numerically compute this distance for two subspaces are provided in Section F.4.

7.4.1 A Synthetic Experiment:

To be able to measure the geodesic Grassmann distance to the optimal subspace $\hat{V}$, we must know the leading eigenfunctions of the pushforward covariance $\mathcal{G}_{\#}\mu$. In general we will not have an analytic form for this, but in this section we devise an experiment where the ordered eigenfunctions of $\mathcal{G}_{\#}\mu$ are known exactly. To do so we construct a linear operator $\mathcal{G} : L^2(\mathcal{X}, \mathbb{R}) \to L^2(\mathcal{X}, \mathbb{R})$ as follows. Given any orthonormal basis $\{\phi_i\}_{i=1}^{\infty}$ of $L^2(\mathcal{X}, \mathbb{R})$ and a summable sequence of positive scalars λ_i, we define the probability distrubiton μ on $L^2(\mathcal{X}, \mathbb{R})$ as the mean zero Gaussian measure $N(0, \Gamma_0)$ with covariance operator

$$\Gamma_0 = \sum_{i=1}^{\infty} \lambda_i \; \phi_i \otimes \phi_i. \tag{21}$$

By the Karhunen-Loeve theorem (Adler, 1990), given $\xi_i \sim N(0, 1)$ standard i.i.d normal scalar random variables, the following random function is distributed according to μ,

$$u = \sum_{i=1}^{\infty} \xi_i \sqrt{\lambda_i} \phi_i. \tag{22}$$

For a summable sequence of positive scalars γ_i we define a linear operator

$$\mathcal{G} = \sum_{i=1} \gamma_i \; \phi_i \otimes \phi_i. \tag{23}$$

The pushforward measure $\mathcal{G}_{\#}\mu$ then has covariance

$$\Gamma = \sum_{i=1}^{\infty} \lambda_i \gamma_i \; \phi_i \otimes \phi_i. \tag{24}$$

After a potential reindexing of the sum in (24) so that $\gamma_i \lambda_i$ are in decreasing order, by construction we have that the optimal n-dimensional subspace of output functions for an operator learning architecture to learn is given by $\hat{V} = \operatorname{span}\{\phi_1, \dots, \phi_n\}$. Since we had the freedom to choose any orthonormal collection of ϕ_i, we are able to compute the geodesic Grassmann distance of any other n-dimensional subspace V to $\hat{V}$.

We will use this setup to train LOCA and its variants on a dataset $\{u^\ell, \mathcal{G}(u^\ell)\}$ where u^ℓ is sampled as in (22) and $\mathcal{G}$ is as defined in (23). We choose $\lambda_i = ((2\pi i)^2 + \tau^2)^{-\alpha}$ and $\phi_i(y) = \sqrt{2} \sin(2\pi i y)$, which are the eigenvalues and eigenfunctions of the operator $(I + \Delta)^{-\alpha}$ on $\mathcal{X} = [0, 1]$ with zero boundary conditions. To form the operator $\mathcal{G}$ we sample γ_i i.i.d. from a truncated normal distribution $\mathcal{TN}(0, 1)$. After training, we take the trained φ_i and compute the geodesic Grassmann distance between $\operatorname{span}\{\varphi_i, \dots, \varphi_n\}$ and $\hat{V}$ (see Section F.4 for implementation details). This allows us to see how the different architecture components of LOCA affect its ability to learn a subspace aligned with the optimal $\hat{V}$.

In Table 2 we report the normalized Grassmann distance of the subspace learned by an operator learning architecture to the optimal subspace $\hat{V}$. We see that LOCA with the KCA, kernel feature map q, and scattering transform gives the best alignment with $\hat{V}$, though LOCA with only the KCA and no map q or scattering transform also gives better alignment than the DeepONet. This suggests that while LOCA does not avoid the fundamental lower bound for the error in Lanthaler et al. (2022), we see that its novel architecture components allow us to get closer to it.

In Table 3 we report the normalized Grassmann distance to $\hat{V}$ after training LOCA with different kernels in the KCA mechanism. In particular, we see that for a problem whose functions have periodic structure, such as this, periodic kernels give the best alignment to the dominant eigenspace of $\Gamma_{\mathcal{G}_{\#}\mu}$. This shows that not only does the KCA itself lead to better representations of the output functions, but a proper choice of kernel can give a meaningful inductive bias for the attention weight functions $\varphi(y)$.

8. Discussion

This work proposes a novel operator learning framework with approximation theoretic guarantees. Drawing inspiration from the Bahdanau attention mechanism (Bahdanau et al., 2015), the model is constructed by averaging a feature embedding of an input function with a probability distribution dependent on the location at which we evaluate the output function.

A key novelty of our approach is the coupling of these probability distributions through a variation of the classic attention mechanism called *Kernel-Coupled Attention* (KCA). In KCA, the probability distributions for the input features are coupled to each other with a kernel

Method	Grassmann distance
LOCA	**0.263**
LOCA removing the q function and the scattering transform	0.393
DON	0.454

Table 2: Grassmann distance between the subspaces defined by the span of the real eigenfunctions and the eigenfunctions derived by orthonormalizing the learned φ functions. We consider the LOCA method with a periodic kernel and remove parts of it until we get the DeepONet method. We observe that the Grassmann distance increases as we go from LOCA to DON.

Method	Grassmann distance
No KCA	0.386
RBF kernel	0.461
Periodic Kernel	**0.263**
Local Periodic Kernel	0.367
Mattern 5/2 Kernel	0.433
Mattern 3/2 Kernel	0.430
Rational Quadratic Kernel	0.377

Table 3: Effect of the kernel choice in KCA to the Grassmann distance from the subspace spanned by the true eigenfunctions.

integral transform over the query domain $\mathcal{Y}$. This provides two main benefits. The first is that correlations between points of the output function can be learned and enforced by the action of a kernel integral transform. Additionally, when there are few available measurements of the output functions in the training dataset, this kernel integral transformation can provide more reliable gradients with respect to the score network parameters. We hypothesize, and support with experiments, that this property allows the model to learn very efficiently using a small fraction of labeled data. In order to have a feature encoder that is robust to small deformations and noise in the input, we employ a multi-resolution feature extraction method based on the wavelet scattering transform of Bruna and Mallat (2013). We empirically show that this is indeed a property of our model. Our experiments additionally show that the model is able to generalize across varying distributions of functional inputs, and is able to extrapolate on a functional regression task with global climate data.Lastly, we show in a synthetic experiment that each component of LOCA, together and in isolation, increases the alignment of the subspace of output functions the model learns with a subspace of output functions which optimizes a known lower bound to the error from Lanthaler et al. (2022).

As discussed in Section 6.4, the error for the methods used in the experiments has a lower bound that is controlled by the decay of the spectrum of the output function covariance operator. When this decay is slow (for example transport dominated problems) these models have difficulty learning an approximation of the true operator, see Lanthaler et al. (2021). While we cannot avoid this fundamental lower bound, we show in section 7.4 that the KCA component of our architecture is able to bring us closer to the optimal subspace of output functions for this lower bound. However, we show that not all choices for the coupling kernel perform equally well for the synthetic problem presented in 7.4. Thus, a potential drawback of the method is that a poorly chosen kernel might reduce the performance of the base model. Prior knowledge on the structure or regularity of output functions can help to avoid choosing kernels that would have this effect.

Another potential drawback of the proposed method is the computational cost needed for numerically approximating the integrals in the KCA mechanism. When using Monte-Carlo with P query points there is a complexity of $O(P^2)$. Instead if a quadrature approach is taken with P queries and Q nodes there is a complexity of $O(PQ+Q^2)$. The relative efficiency of these two approaches in general will depend on the number of quadrature points necessary for a good approximation, and thus on the dimension of the query domain. In general, integrals over high dimensional domains will become increasingly costly to compute.

Therefore, an immediate future research direction is to use further approximations to allow the kernel integral computations to scale to larger numbers of points and dimensions. A first approach is to parallelize this integral computation by partitioning the domain into pieces and summing the integral contributions from each piece. To lower the the computational complexity of the kernel computations between the query and integration points we can also use approximations of the kernel matrix. For example, in the seminal paper of Rahimi and Recht (Rahimi et al., 2007) the authors propose an approximation of the kernel using a random feature strategy. More recently, in the context of transformer architectures, a number of approximations have been proposed to reduce the complexity of such computations to be linear in the number of kernel points $O(N)$. A non-exhaustive list of references include Linformers (Wang et al., 2020), Performers (Choromanski et al., 2020), Nyströformers (Xiong et al., 2021) and Fast Transformers (Katharopoulos et al., 2020).

Another potential extension of our framework is to take the output of our model as the input function of another LOCA module and thus make a layered version of the architecture. While in our experiments we did not see that this modification significantly increased the performance of the model, it is possible that other variants of this modular architecture could give performance improvements. Lastly, recall that the output of our model corresponds to the context vector generated in the Bahdanau attention. In the align and translate model of Bahdanau et al. (2015) this context vector is used to construct a distribution over possible values at the output location. By using the output of our model as a context vector in a similar architecture, we can create a probabilistic model for the potential values of the output function, therefore providing a way to quantify the uncertainty associated with the predictions of our model.

A main application of operator learning methods is for PDEs, where they are used as surrogates for traditional numerical solvers. Since the forward pass of an operator learning model is significantly faster than classical numerical methods, the solution of a PDE under many different initial conditions can be expediently obtained. This can be a key enabler in design and optimal control problems, where many inputs must be tested in pursuit of identifying an optimal system configuration. A key advantage of operator learning techniques in this context is that they also allow the quick evaluation of sensitivities with respect to inputs (via automatic differentiation), thus enabling the use of gradient-based optimization. Conventional methods for computing sensitivities typically rely on solving an associated adjoint system with a numerical solver. In contrast, a well-trained operator learning architecture can compute these sensitivities at a fraction of the time. Therefore, we expect that successful application of operator learning methods to predict the output of physical systems from control inputs can have a significant impact in the design of optimal inputs and controls. Some preliminary work in this direction has been explored in Wang et al. (2021a).

Acknowledgments

G.K. and P.P. would like to acknowledge support from the US Department of Energy under under the Advanced Scientific Computing Research program (grant DE-SC0019116), the US Air Force (grant AFOSR FA9550-20-1-0060), and US Department of Energy/Advanced Research Projects Agency (grant DE-AR0001201). J.S. and G.P. would like to acknowledge support from the AFOSR under grant FA9550-19-1-0265 (Assured Autonomy in Contested Environments) and the NSF Simmons Mathematical and Scientific Foundations of Deep Learning (grant 2031985). We also thank the developers of the software that enabled our research, including JAX (Bradbury et al., 2018), Kymatio (Andreux et al., 2020), Matplotlib (Hunter, 2007), Pytorch (Paszke et al., 2019) and NumPy (Harris et al., 2020). We would also like to thank Andreas Kalogeropoulos and Alp Aydinoglu for their useful feedback on the manuscript.

Appendix A. Nomenclature

Table 4 summarizes the main symbols and notation used in this work.

Appendix B. Proof of Theorem 2

Proof The starting point of the proof is the following lemma, which gives justification for approximating operators on compact sets with finite dimensional subspaces as in Chen and Chen (1995); Lanthaler et al. (2022); Kovachki et al. (2021b). The lemma follows immediately from the fact that for any compact subset $\mathcal{U}$ of a Banach space $\mathcal{E}$ and any $\epsilon > 0$, there exists a finite dimensional subspace $\mathcal{E}_n \subset \mathcal{E}$ such that $d(\mathcal{U}, \mathcal{E}_n) < \epsilon$.

Lemma 6 *Let $\mathcal{U} \subset \mathcal{E}$ be a compact subset of a Banach space $\mathcal{E}$. Then for any $\epsilon > 0$, there exists $n \in \mathbb{N}$, $\phi_1, \dots, \phi_n \in \mathcal{E}$, and functionals $c_1, \dots, c_n$ with $c_i : \mathcal{E} \to \mathbb{R}$ such that*

$$\sup_{u \in \mathcal{U}} \|u - \sum_{i=1}^{n} c_i(u)\phi_i\|_{\mathcal{E}} < \epsilon.$$

Returning to the problem of learning a continuous operator $\mathcal{G} : \mathcal{U} \to C(\mathcal{Y}, \mathbb{R}^{d_s})$, since $\mathcal{U}$ is assumed to be compact and $\mathcal{G}$ is continuous, the image $\mathcal{G}(\mathcal{U})$ is compact in the co-domain. Thus, we may apply Lemma 6 to the set $\mathcal{G}(\mathcal{U})$. This shows that for any $\epsilon > 0$, there exists $c_1, \dots, c_n$ with each $c_i : \mathcal{U} \to \mathbb{R}$ linear and continuous and functions $\phi_1, \dots, \phi_n$ with $\phi_i : \mathcal{Y} \to \mathbb{R}^{d_s}$ such that

$$\sup_{u \in \mathcal{U}} \sup_{y \in \mathcal{Y}} \left\| \mathcal{G}(u)(y) - \sum_{i=1}^{n} c_i(u)\phi_i(y) \right\| < \epsilon. \tag{25}$$

$[n]$	The set $\{1, \ldots, n\} \subset \mathbb{N}$.
$u \odot v$	Hadamard (element-wise) product of vectors u and v.
$C(A, B)$	Space of continuous functions from a space A to a space B.
C^1	Space of continuous functions with continuous derivative.
L^2	Hilbert space of square integrable functions.
$\mathcal{H}_k$	Reproducing Kernel Hilbert Space with kernel k.
Δ^n	n-dimensional simplex.
$\mathcal{X}$	Domain for input functions.
$\mathcal{Y}$	Domain for output functions.
x	Input function arguments.
y	Output function arguments (queries).
u	Input function in $C(\mathcal{X}, \mathbb{R}^{d_u})$.
s	Output function in $C(\mathcal{Y}, \mathbb{R}^{d_s})$.
$\mathcal{F}$	Operator mapping input functions u to output functions s.
$g(y)$	Proposal score function.
$\tilde{g}(y)$	Kernel-Coupled score function.
$\varphi(y)$	Attention weights at query y.
$v(u)$	Feature encoder.
$\kappa(y, y')$	Coupling kernel.
$k(y, y')$	Base similarity kernel.

Table 4: (Nomenclature) A summary of the main symbols and notation used in this work.

Next, we show that the approximation of $\mathcal{G}(u)(y)$ given in (25) can be expressed equivalently as vector of averages of a modified collection of functionals c_i. These functionals c_i will form the coordinates of our input feature vector $v(u)$. First, for each $\phi_i : \mathcal{Y} \to \mathbb{R}^{d_s}$ we may form the positive and negative parts, whose coordinates are defined by

$$(\phi_i^+)_q = \max\{(\phi_i)_q, 0\}$$
$$(\phi_i^-)_q = -\min\{(\phi_i)_q, 0\}$$

Note that ϕ_i^+ and ϕ_i^- are continuous, non-negative, and that $\phi_i = \phi_i^+ - \phi_i^-$. For $j = 1, \ldots, 2n$ define a new collection of functions $\varphi_j : \mathcal{Y} \to \mathbb{R}^{d_s}$ by

$$\varphi_j = \begin{cases} \frac{1}{2n\|\phi_i^+\|_\infty}\phi_i^+ & \text{if } j = 2i \\ \frac{1}{2n\|\phi_i^-\|_\infty}\phi_i^- & \text{if } j = 2i-1 \end{cases}$$

and define

$$\varphi_{2n+1} := \mathbf{1}_{d_s} - \sum_{j=1}^{2n} \varphi_j.$$

By construction, for all y we have that $\varphi_j(y) \in [0,1]^{d_s}$,

$$\mathrm{span}\{\phi_i\}_{i=1}^n \subseteq \mathrm{span}\{\varphi_j\}_{j=1}^{2n+1},$$

and

$$\sum_{j=1}^{2n+1} \varphi_j(y) = \mathbf{1}_{d_s}.$$

In order to allow each output dimension of each φ_j to have its own coordinate function, (and thus have $v(u) \in \mathbb{R}^{n \times d_s}$), for each φ_j, we create d_s new functions,

$$\varphi_{j,k}(y) := e_k \odot \varphi_j(y),$$

where $e_k \in \mathbb{R}^{d_s}$ is the k-th standard basis vector in $\mathbb{R}^{d_s}$. Thus, we have constructed a collection of vectors $\varphi_{j,k}$ such that $\langle \varphi_{j,k}, e_m \rangle = 0$ if and only if $k \neq m$,

$$\sum_{j=1}^{2n+1} \sum_{k=1}^{d_s} \varphi_{j,k}(y) = \mathbf{1}_{d_s}, \quad \forall y \in \mathcal{Y},$$

and

$$\operatorname{span}\{\phi_i\}_{i=1}^{n} \subseteq \operatorname{span}\big\{\varphi_{j,k}\big\}_{\substack{j \in [2n+1] \\ k \in [d_s]}}.$$

Since from Lemma 6 we know

$$d(\operatorname{span}\{\phi_i\}_{i=1}^{n}, \mathcal{G}(\mathcal{U})) < \epsilon,$$

we conclude that

$$d(\operatorname{span}\big\{\varphi_{j,k}\big\}_{\substack{j \in [2n+1] \\ k \in [d_s]}}, \mathcal{G}(\mathcal{U})) < \epsilon,$$

and can conclude the statement of the theorem. ■

Appendix C. Proof of Proposition 3

Proof Note that $\operatorname{im}(T_k^{1/2}) = \mathcal{H}_\kappa$, (Paulsen and Raghupathi, 2016). Since κ is universal, $\operatorname{im}(T_\kappa^{1/2}) = \mathcal{H}_\kappa \subset C(\mathcal{Y}, \mathbb{R}^n)$ is dense. Thus, it suffices to show that $T_\kappa(\mathcal{A})$ is dense in $\operatorname{im}(T_\kappa^{1/2})$. We will make use of the following fact, which we state as a lemma.

Lemma 7 *If $f : X \to Y$ is a continuous map and $A \subset X$ is dense, then $f(A)$ is dense in* $\operatorname{im}(f)$.

By the above lemma, we have that $T_\kappa(\mathcal{A})$ is dense in $\operatorname{im}(T_\kappa)$. Now we must show that $\operatorname{im}(T_\kappa) \subset \operatorname{im}(T_\kappa^{1/2})$ is dense as well. This again follows from the above lemma by noting that $\operatorname{im}(T_k) = T_k^{1/2}(\operatorname{im}(T_k^{1/2}))$, and $\operatorname{im}(T_k^{1/2})$ is dense in the domain of $T_k^{1/2}$. ■

Appendix D. Proof of Proposition 4

In this section, we show that the coupling kernel is symmetric and positive semi-definite. These two conditions are necessary to obtain theoretical guarantees of universality. The symmetry of the kernel κ follows immediately from the symmetry of the base kernel k in (3) and the form of κ in (4). To prove κ is positive semi-definite we must show for any $v_1, \dots, v_n \in \mathbb{R}$ and $y_1, \dots, y_n \in \mathcal{Y}$,

$$\sum_{i,j=1}^{n} v_i v_j \kappa(y_i, y_j) \geq 0.$$

For ease of notation define

$$Z_i := \left(\int_{\mathcal{Y}} k(q(y_i), q(z))\, dz \right)^{1/2}.$$

Using the definition of κ from (4),

$$\begin{aligned} \sum_{i,j=1}^{n} v_i v_j \kappa(y_i, y_j) &= \sum_{i,j=1}^{n} v_i v_j \frac{k(q(y_i), q(y_j))}{Z_i Z_j} \\ &= \sum_{i,j=1}^{n} \frac{v_i}{Z_i} \frac{v_j}{Z_j} k(q(y_i), q(y_j)) \\ &\geq 0, \end{aligned}$$

where in the last line we have used the positive semi-definiteness of k.

Finally, the injectivity of the map g would imply that the overall feature map of κ is injective, which gives that the kernel is universal (Christmann and Steinwart, 2010).

Appendix E. Proof of Proposition 5

Proof Since $\mathcal{U}$ is compact, h is uniformly continuous. Hence, there exists $\delta > 0$ such that for any $\|u - v\| < \delta$, $\|h(u) - h(v)\| < \epsilon/2$. Define $u_d := \sum_{i=1}^{d} \langle u, e_i \rangle e_i$. By the uniform convergence of $u_d \to u$ over $u \in \mathcal{U}$, there exists d such that for all $u \in \mathcal{U}$, $\|u - u_d\| < \delta$. Thus, for all $u \in \mathcal{U}$

$$\|h(u) - h(u_d)\| < \frac{\epsilon}{2}.$$

If we define $r : \mathbb{R}^d \to C(\mathcal{X}, \mathbb{R}^{d_u})$ as

$$r(\alpha) := \sum_{i=1}^{d} \alpha_i e_i,$$

we may write $h(u_d) = (h \circ r)(\mathcal{D}_d(u))$. Now, note that $h \circ r \in C(\mathbb{R}^d, \mathbb{R}^n)$, and recall that, by assumption, the function class $\mathcal{A}_d$ is dense in $C(\mathbb{R}^d, \mathbb{R}^n)$. This means there exists $f \in \mathcal{A}_d$ such that $\|f - h \circ r\| < \epsilon/2$. Putting everything together, we see that

$$\|h(u) - f \circ \mathcal{D}_d(u)\| \leq \|h(u) - h(u_d)\| + \|(h \circ r)(\mathcal{D}_d(u)) - f \circ \mathcal{D}_d(u)\| < \epsilon.$$

■

Appendix F. Supplementary Information for Experiments

In this section, we present supplementary information on the experiments presented in Section 7.

F.1 Computational Complexity

In LOCA the most expensive operations are the integral computations in the KCA mechanism. Let $z_1, \ldots, z_Q$ be the integration nodes with $z = [z_1, \ldots, z_Q]^\top$, and let the associated weights be $w_1, \ldots, w_Q$, with $w = [w_1, \ldots, w_Q]^\top$. For evaluating the KCA mechanism at a single query location y_0 with Q integration nodes we are required to compute the matrices $k(y_0, z) \in \mathbb{R}^{1\times Q}$, with $[k(y_0, z)]_j = k(y_0, z_j)$ and $k(z, z) \in \mathbb{R}^{Q\times Q}$, with $[k(z, z)]_{ij} = k(z_i, z_j)$. These are combined to compute the kernel κ as

$$\kappa(y_0, z) \approx \frac{1}{(k(y_0, z)w)^{1/2}} k(y_0, z) \odot (k(z, z)w)^{-1/2},$$

where the exponent of $-1/2$ in the last factor is applied coordinate-wise. Carrying out this computation requires Q steps to compute $k(y_0, z)$ and Q^2 steps to compute $k(z, z)$, giving an overall complexity of $Q + Q^2$. When considering $P > 1$ the complexity for computing $k(y_0, z)$ becomes PQ and the overall complexity becomes $PQ + Q^2$ because we need to compute $k(z, z)$ only once. For the Monte Carlo case, $Q = P$ and $y = z$, so we only need to make one computation of $k(y, y)$. Therefore in this case, we have a complexity of Q^2.

Both methods have their benefits and disadvantages: in the Monte Carlo case, we need to perform P^2 computations once, but the cost scales exponentially with P. On the other hand, Gauss-Legendre quadrature requires $PQ + Q^2$ evaluations, but if Q is small the overall computational cost is less than Monte-Carlo integration.

F.2 Architecture Choices and Hyper-parameter Settings

In this section we present the neural network architecture choices, the training details, the training wall-clock time, as well as the number of training parameters for each model compared in the experiments. Specifically, for the DON and FNO models, we have performed an extensive number of simulations to identify settings for which these competing methods achieve their best performance.

For LOCA and DON we set the batch size to be 100, initial learning rate equal to $l_r = 0.001$, and an exponential learning rate decay with a decay-rate of 0.95 every 100 training iterations. For the FNO training, we set the batch size to be 100 and consider a learning rate $l_r = 0.001$, which we then reduce by 0.5 every 100 epochs and a weight decay of 0.0001. Moreover, for the FNO method we use the ReLU activation function.

F.2.1 LOCA

For the LOCA model, we present the structure of the functions g, f, and q in Table 5. In Table 6 we present the number of samples considered for the train and test data sets, the number of points where the input and the output functions are evaluated, the dimensionality of positional encoding, the dimensionality of the latent space where we evaluate the expectation $\mathbb{E}(u)(y)$, the batch size used for training, and the number of training iterations. We present

Example	g depth	g width	f depth	f depth	q depth	q width
Antiderivative	2	100	1	500	2	100
Darcy Flow	2	100	2	100	2	100
Mechanical MNIST	2	256	2	256	2	256
Shallow Water Eq.	1	1024	1	1024	1	1024
Climate Modeling	2	100	2	100	2	100

Table 5: LOCA Architectural choices for each benchmark considered in this work: We present the chosen architecture for g and q, the functions that constructs $\phi(y)$, and the function v which together build up the architecture of the LOCA model.

Example	N_{train}	N_{test}	m	P	n	H	l	Batch #	# of train iterations
Antiderivative	1000	1000	100	100	100	10	100	100	50000
Darcy Flow	1000	1000	1024	-	100	6	100	100	20000
Mechanical MNIST	60000	10000	784	56	500	10	100	100	100000
Shallow Water Eq.	1000	1000	1024	128	480	2	100	100	80000
Climate Modeling	1825	1825	5184	144	100	10	100	100	100000

Table 6: LOCA model parameters for each benchmark considered in this work: We present the numbers of training and testing data N_{train} and N_{test}, respectively, the number of input coordinate points m where the input function is evaluated, the number of coordinates P where the output function is evaluated, the dimension of the latent space n over which we evaluate the expectation, the number of positional encoding features H for the positional encoding, the dimensionality of the encoder l, the size of the batch B and the iterations for which we train the model.

the parameters of the wavelet scattering network in Table 7. The method used for computing the kernel integral for each example is presented in Table 8.

F.2.2 DON

For the DON model, we present the structure of b and t, the branch and the trunk functions, in Table 9. In Table 10 we present the number of samples considered for the train and test data sets, the number of points where the input and the output functions are evaluated, the dimensionality of the positional encoding, the dimensionality of the latent space, the batch size used for training, and the number of training iterations. In order to achieve competitive performance, we also adopted some of the improvements proposed in Lu et al. (2021), including the application of harmonic feature expansions to both input and outputs, as well as normalization of the output functions.

F.2.3 FNO

For the FNO model, we present the architecture choice in Table 11. In Table 12 we present the number of samples considered for the train and test data sets, the number of points where the input and the output functions are evaluated, the batch size used for training and the number of training epochs.

Example	J	L	m_o
Antiderivative	4	8	2
Darcy Flow	1	2	2
Mechanical MNIST	1	16	2
Shallow Water Eq.	1	3	2
Climate Modeling	1	8	2

Table 7: Chosen parameters for the wavelet scattering network: J represents the log-2 scatteting scales, L the angles used for the wavelet transform and m_o the maximum order of scattering coefficients to compute. The wavelet scattering network is implemented using the Kymatio library (Andreux et al., 2020).

Example	Integration method
Antiderivative	Quadrature
Darcy Flow	Quadrature
Mechanical MNIST	Quadrature
Shallow Water Eq.	Monte Carlo
Climate Modeling	Monte Carlo

Table 8: Integral computation method for each benchmark considered in this work: We present the method that is used to compute the required kernel integrals for the LOCA method.

Example	b depth	b width	t depth	t depth
Antiderivative	2	512	2	512
Darcy Flow	6	100	6	100
Mechanical MNIST	4	100	4	100
Shallow Water Eq.	11	100	11	100
Climate Modeling	4	100	4	100

Table 9: DON architecture choices for each benchmark considered in this work.

Example	N_{train}	N_{test}	m	P	n	H	Batch #	Train iterations
Antiderivative	1000	1000	1000	100	100	2	100	50000
Darcy Flow	1000	1000	1024	-	100	6	100	20000
Mechanical MNIST	60000	10000	784	56	500	10	100	100000
Shallow Water Eq.	1000	1000	1024	128	480	2	100	80000
Climate Modeling	1825	1825	5184	144	100	10	100	100000

Table 10: DON model parameter for each benchmark considered in this work: We present the numbers of training and testing data N_{train} and N_{test}, respectively, the number of input coordinate points m where the input function is evaluated, the number of coordinates P where the output function is evaluated, the dimension of the latent space n over which we evaluate the inner product of the branch and the trunk networks, the number of positional encoding features H, the size of the batch B and the iterations for which we train the model.

Example	# of modes	width	# of FNO layers
Antiderivative	32	100	4
Darcy Flow	8	32	4
Mechanical MNIST	12	32	4
Shallow Water Eq.	8	25	4
Climate Modeling	12	32	4

Table 11: FNO architecture choices for each benchmark considered in this work.

Example	N_{train}	N_{test}	m	P	Batch #	Train Epochs
Antiderivative	1000	1000	1000	100	100	500
Darcy Flow	1000	1000	1024	-	100	500
Mechanical MNIST	60000	10000	784	56	100	200
Shallow Water Eq.	1000	1000	1024	128	100	400
Climate Modeling	1825	1825	5184	144	73	250

Table 12: FNO model parameter for each benchmark considered in this work: We present the numbers of training and testing data N_{train} and N_{test}, respectively, the number of input coordinate points m where the input function is evaluated, the number of coordinates P where the output function is evaluated, the size of the batch B and the epochs for which we train the model.

Example	LOCA	DON	FNO
Antiderivative	1,677,300	2,186,672	1,333,757
Darcy Flow	381,000	449,400	532,993
Mechanical MNIST	2,475,060	3,050,300	1,188,514
Shallow Water Eq.	5,528,484	5,565,660	5,126,690
Climate Modeling	1,239,500	5,805,800	1,188,353

Table 13: Total number of trainable parameters for each model, and for each benchmark considered in this work.

Method / Error	Testing Error	Training Error
LOCA-S	$5.3410^{-3} \pm 1.8210^{-3}$	$3.2810^{-3} \pm 5.2010^{-4}$
LOCA-WS	$1.2910^{-2} \pm 9.9010^{-3}$	$3.5910^{-3} \pm 6.7310^{-4}$
DON-S	$1.2410^{-2} \pm 2.7010^{-3}$	$1.0610^{-2} \pm 1.2110^{-3}$
DON-WS	$2.0110^{-2} \pm 6.6610^{-3}$	$1.9210^{-2} \pm 3.1210^{-3}$
FNO-S	$2.8710^{-2} \pm 7.3410^{-3}$	$1.8210^{-2} \pm 3.6710^{-3}$
FNO-WS	$9.7210^{-3} \pm 1.7310^{-3}$	$7.8610^{-3} \pm 1.2910^{-3}$

Table 14: Training and testing relative $\mathcal{L}_2$ error for the case of the Darcy flow for all the methods considered in this study with scattering transform as an input feature extractor (denoted as '-S' above) and without scattering (denoted as '-WS' above).

F.3 Effect of the wavelet scattering transform as a spectral encoder

Here we perform an ablation study to determine the effect of the wavelet scattering transform on the performance of the methods employed in this manuscript. To this end, we examine its effect on the relative $\mathcal{L}_2$ error on the training and testing data sets as a measure of approximation and generalization capabilities, respectively. We train LOCA, the DON and the FNO methods with and without the scattering transform on the Darcy example and 12.5% of the data set used for training. The Darcy flow problem set-up is explained in section F.7.2 in detail. The results are presented in Table 14.

We observe that for LOCA the scattering transform nearly does not affect the training error, but the testing error is affected. Given that the scattering transform is capable of extracting meaningful features from the training data set that are also present in the testing data set, it makes sense that the testing error is improved. For the DON case, the scattering transform does not affect the mean error but the standard deviation of the error vectors in both the training and testing case is slightly decreased. As a result, the scattering transform feature encoding can improve the worst case scenario for the DON case in this example. For the FNO case, the scattering transform makes both the mean and the distribution of the training and testing error vectors worse. This is probably a result of performing the Fourier transform on the scattering coefficients, which are already in the frequency domain.

Overall, we argue that the while the scattering transform can have benefits as an effective feature extractor for pre-processing data, this is not the sole reason for the differences in experimental performance between LOCA and other methods.

F.4 Computing the Grassmann Distance for Subspace Alignment

To compute the Grassmann distance between $V = \text{span}\{\varphi_1, \ldots, \varphi_n\}$ and $\hat{V} = \text{span}\{\phi_1, \ldots, \phi_n\}$, we first use Gram-Schmidt to compute an orthonormal basis for V, $\{\psi_1, \ldots, \psi_n\}$. Then we perform a matrix of pairwise inner products $[M]_{ij} = \langle \psi_i, \phi_j \rangle$. We perform a singular value decomposition on M and extract the singular values $\sigma_1, \ldots, \sigma_n$. The principal angles are given by $\cos^{-1}(\sigma_1), ..., \cos^{-1}(\sigma_P)$, which are then used to compute the Grassmann distance as $G(V, \hat{V}) = \Big(\sum_{i=1}^{k} \cos^{-1}(\sigma_i^2) \Big)^{1/2}$.

Example	LOCA	DON	FNO
Antiderivative	2.23	2.08	2.06
Darcy Flow ($P = 1024$)	5.51	3.5	1.50
Mechanical MNIST	21.70	16.61	22.87
Shallow Water Eq.	12.10	15.39	13.95
Climate Modeling	4.52	7.51	10.49

Table 15: Computational cost for training each model across all benchmarks considered in this work: We present the wall clock time *in minutes* that is needed to train each model on a single NVIDIA RTX A6000 GPU.

F.5 Computational Cost

We present the wall clock time, in minutes, needed for training each model for each different example presented in the manuscript in Table 15. For the case of the Darcy flow, the computational time is calculated for the case of $P = 1024$, meaning we use all available labeled output function measurements per training example. We choose this number of query points to show that even when the number of labeled data is large, the computational cost is still reasonable, despite the KCA computation bottleneck. We observe that the wall clock time for all methods lie in the same order of magnitude. All the models are trained on a single NVIDIA RTX A6000 GPU.

F.6 Comparison Metrics

Throughout this work, we employ the relative $\mathcal{L}_2$ error as a metric to assess the test accuracy of each model, namely:

$$\text{Test error metric} = \frac{||s^i(y) - \hat{s}^i(y)||_2^2}{||s^i(y)||_2^2},$$

where $\hat{s}(y)$ the model predicted solution, $s(y)$ the ground truth solution and i the realization index. The relative $\mathcal{L}_2$ error is computed across all examples in the testing data set, and different statistics of the error vector are calculated: the median, quantiles, and outliers. For all examples the errors are computed between the full resolution reconstruction and the full resolution ground truth solution.

F.7 Experiments

In this section, we present additional details about the experimental scenarios discussed in Section 7.

F.7.1 Antiderivative

We approximate the antiderivative operator for demonstrating the generalization capabilities of the LOCA in two inference scenarios. The antiderivative operator is defined as

$$\frac{ds(x)}{dx} = u(x), \quad s(x) = s_0 + \int_0^x u(\tau)d\tau,$$

where we consider $x \in \mathcal{X} = [0,1]$ and the initial condition $s(0) = 0$. For a given forcing term u the solution operator $\mathcal{G}$ of system (F.7.1) returns the antiderivative $s(x)$. In the notation of our model, the input and output function domains coincide, $\mathcal{X} = \mathcal{Y}$ with $d_x = d_y = 1$. Since the solution operator is a map between scalar functions, we also have $d_u = d_s = 1$. Under this setup, our goal is to learn the solution operator $\mathcal{G} : C(\mathcal{X}, \mathbb{R}) \to C(\mathcal{X}, \mathbb{R})$.

To construct the data sets we sample the forcing function $u(x)$ from a Gaussian process prior and measure these functions at 500 points. We numerically integrate them to obtain 100 measurements of each output function to use for training different operator learning models.

For investigating the performance of LOCA on out-of-distribution prediction tasks, we create training data sets by choosing $l_{train} \in [0.1, 0.9]$, and consider 9 cases of increasing l_{train} spaced by 0.1 each. The output scale of the Gaussian Process prior o is equal to one for all length scales. The training and testing data sets each have $N = 1,000$ solutions of equation (F.7.1), and we use 100% of all available output evaluation function points, both for training and testing.

For the case where we train and test on multiple length and output scales, we construct each example in the data set as follows. To construct each input sample, we first we sample a uniform random variable $\delta \sim \mathcal{U}(-2, 1)$, and set the corresponding input sample length-scale to $l = 10^{\delta}$. Similarly, we construct a random amplitude scale by sampling $\zeta \sim \mathcal{U}(-2, 2)$, and setting $o = 10^{\zeta}$. Then we sample $u(x)$ from a Gaussian Process prior $u(x) \sim GP(0, \text{Cov}(x, x'))$, where $\text{Cov}(x, x') = o \exp\left(-\frac{\|x - x'\|}{l}\right)$. The length and the outputs scales are different for each realization, therefore we have $1,000$ different length and outputs scales in the problem.

F.7.2 Darcy Flow

Fluid flow through porous media is governed by Darcy's Law (Bear, 2013), which can be mathematically expressed by the following partial differential equation system,

$$\nabla \cdot (u(x) \nabla s(x)) = f(x), \quad x \in \mathcal{X}, \tag{26}$$

subject to appropriate boundary conditions

$$\begin{aligned} s &= 0 \quad \text{on } \Gamma_{\mathcal{X}}, \\ (u(x) \nabla s(x)) \cdot n &= g \quad \text{on } \Gamma_N, \end{aligned}$$

2 where u is permeability of the porous medium, and s is the corresponding fluid pressure. Here we consider a domain $\mathcal{X} = [0,1] \times [0,1]$ with a Dirichlet boundary $\Gamma_D = \{(0, x) \cup (1, x) \mid x_2 \in [0,1] \subset \partial \mathcal{X}\}$, and a Neumann boundary $\Gamma_N = \{(x, 0) \cup (x, 1) \mid x \in [0,1] \subset \partial \mathcal{X}\}$.

For a given forcing term f and set of boundary conditions, the solution operator $\mathcal{G}$ of system (26) maps the permeability function $u(x)$ to the fluid pressure function $s(x)$. In the notation of our model, the input and output function domains coincide, $\mathcal{X} = \mathcal{Y}$ with $d_x = d_y = 2$. Since in this case the solution operator is a map between scalar functions, we also have $d_u = d_s = 1$. Under this setup, our goal is to learn the solution operator $\mathcal{G} : C(\mathcal{X}, \mathbb{R}) \to C(\mathcal{X}, \mathbb{R})$.

We set the Neumann boundary condition to be $g(x) = \sin(5x)$, the forcing term $f(x) = 5 \exp(-((x_1 - 0.5)^2 + (x_2 - 0.5)^2))$, and sample the permeability function $u(x)$ from a Gaussian

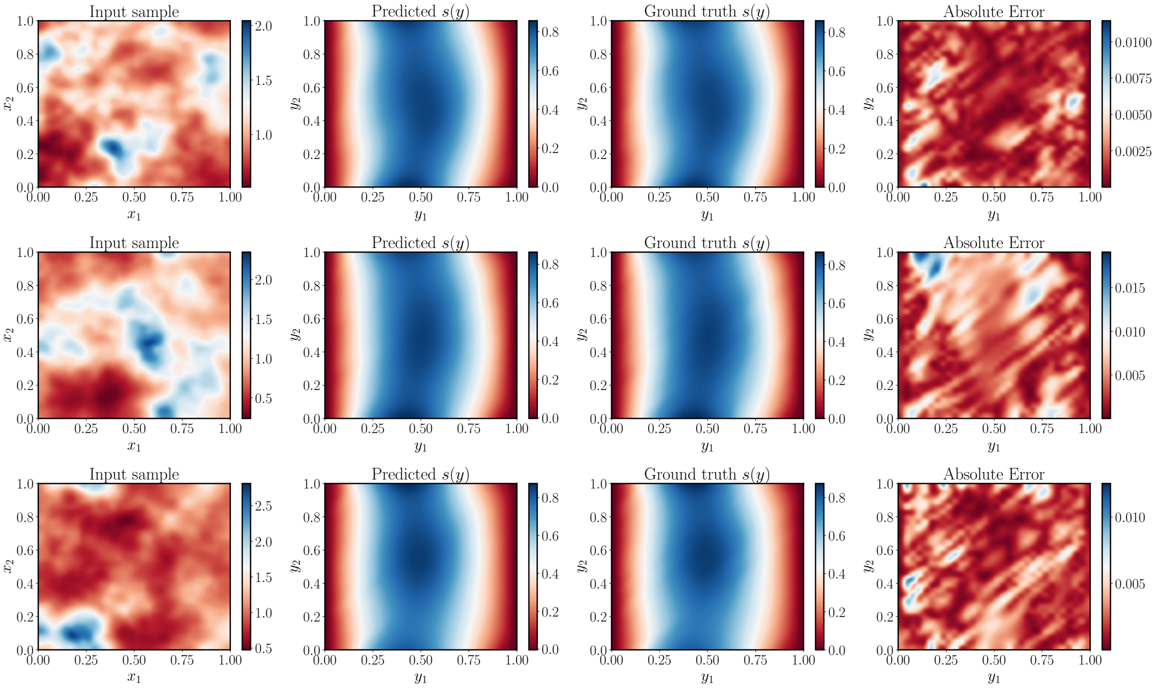


Figure 12: Comparison between the full resolution prediction and ground truth for the flow through porous medium testing data set: We present the input sample, the prediction, the ground truth and the absolute error for three realizations of the Darcy flow system. The first row corresponds to the example with minimum test error, the second row to the example with maximum test error, while the third row corresponds to a randomly chosen example from the testing data set.

measure, as $u(x) = \exp(u_0 \cos(x))$ with $u_0 \sim \mathcal{N}(0, 7^{3/2}(-\Delta + 49I)^{-1.5}$. The training and testing data sets are constructed by sampling the initial condition along a 32 × 32 grid and solving the forward problem with the Finite Element library, Fenics (Alnæs et al., 2015). This gives us access to 32 × 32 solution values to use for training different operator learning models. Sub-sampling these solution values in the manner described in Section 7 allows us to create training data sets to examine the effect of using only a certain percentage of the available data.

Figure 12 gives a visual comparison of the outputs of our trained model against the ground truth for three randomly chosen initial conditions, along with a plot of the point-wise error. We see that our model performs well across random initial conditions that were not present in the training data set.

F.7.3 Mechanical MNIST

For this example, our goal is to learn the operator that maps initial deformations to later-time deformations in the equi-biaxial extension benchmark from the Mechanical MNIST database

(Lejeune, 2020). The data set is constructed from the results of $70,000$ finite-element simulations of a heterogeneous material subject to large deformations. MNIST images are considered to define a heterogeneous block of material described by a compressible Neo-Hookean model (Lejeune, 2020).

In our case, we are interested in learning displacement fields at later times, given some initial displacement. The material constitutive law is described by Lejeune (2020)

$$\psi = \frac{1}{2}\mu\Big[\mathbf{F}:\mathbf{F} - 3 - 2\ln(\det\mathbf{F})\Big] + \frac{1}{2}\lambda\Big[((\det\mathbf{F})^2 - 1) - \ln(\det\mathbf{F})\Big], \tag{27}$$

where ψ is the strain energy, $\mathbf{F}$ is the deformation energy, μ and λ are Lamé constants that can be found from the Young's modulus and the Poisson ratio

$$E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu}, \qquad \nu = \frac{\lambda}{2(\lambda + \mu)}.$$

The Young's modulus is chosen based on the bitmap values to convert the image to a material as

$$E = \frac{b}{255}(100 - 1) + 1,$$

where b is the bitmap value. Here, the Poisson ratio is fixed to $\nu = 0.3$ for all block materials. This means that the pixels inside the digits are block materials that are much stiffer than the pixels that are outside of the digits. For the equi-biaxial extension experiments, Dirichlet boundary conditions are applied by considering different displacement values $\mathbf{d} = [0.0, 0.001, 0.01, 0.1, 0.5, 1, 2, 4, 6, 8, 10, 12, 14]$ for the right and top of the domain, and $-\mathbf{d}$ for the left and bottom of the domain.

In this benchmark the input and the output function domains coincide, $\mathcal{X} = \mathcal{Y}$ with $d_x = d_y = 2$, while the solution operator, $\mathcal{G}$, is a map between vector fields with $d_u = d_s = 2$. Consequently, our goal here is to learn the solution operator $\mathcal{G} : \mathcal{C}(\mathcal{X}, \mathbb{R}^2) \mapsto \mathcal{C}(\mathcal{X}, \mathbb{R}^2)$. Even though we create a map between displacement vectors, we present the magnitude of the displacement

$$s = \sqrt{v_1^2 + v_2^2},$$

for visual clarity of our plots.

The data set is constructed by sampling MNIST digits on a 28×28 grid and solving equation 27 using the Finite Element library, Fenics (Alnæs et al., 2015). Out of the $70,000$ realizations that the MNIST data set contains, $60,000$ are used for training and $10,000$ are used for testing, therefore $N_{train} = 60,000$ and $N_{test} = 10,000$. We randomly sub-sample the number of measurement points per output function, as explained in Section 7, to create a training data set to demonstrate that our model only needs a small amount of labeled data to provide accurate predictions.

We present a visual comparison of the outputs of the trained model against the ground truth solution for three randomly chosen initial conditions from the test data set in Figure 13. Figure 14 presents the same comparison, for the minimum error prediction, to show the change in the pixel position due to the applied displacement, which is not visible in the case where we present multiple solutions at the same time. The error reported in Figure 14 illustrates the discrepancy (shown in magenta) between the ground truth and the predicted pixel positions.

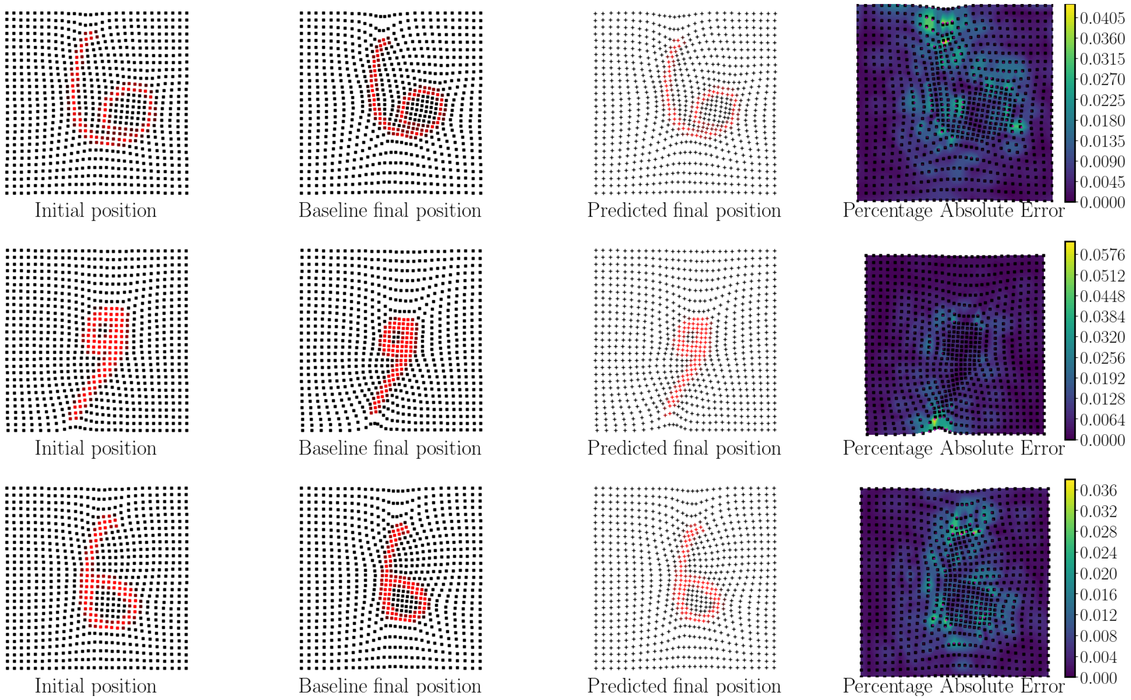
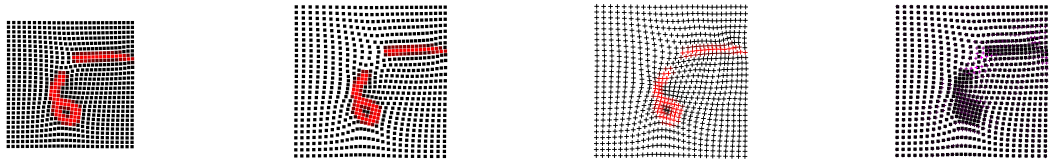


Figure 13: Comparison between the predicted and the ground truth displacement magnitudes for three random testing data set initial conditions (100, 1,000, 10,000) of the Mechanical MNIST case: We present the results of our model for 3 different MNIST digits under final displacement $d = 14$. Despite the our model having displacement vector fields as inputs and outputs, we present our inputs and results in the form of positions. For this purpose, we add the displacements in the horizontal and vertical directions to the undeformed positions of the MNIST digit pixels which we assume that lie on a regular grid. The normalized absolute error is computed with respect to the position and not the displacement in each direction.


Initial position

Baseline final position

Predicted final position

Error

Figure 14: Schematic comparison between the predicted and the ground truth final positions for the best testing data set prediction for the Mechanical MNIST benchmark: We present the ground truth final and the predicted final positions of the block material together with the point-wise error between them (shown in magenta), as well as the initial position. We present this result in a schematic manner, meaning without some indication of the error magnitude, in order to provide a sense of the deformation of the MNIST pixels under the final displacement.

F.7.4 Shallow Water Equations

The modeling of the currents in Earth science is often modelled by the Shallow Water equations, which describes the flow below a pressure surface when the horizontal length-scales are much larger than the vertical ones. The system of equations is defined as:

$$
\begin{aligned}
&\frac{\partial \rho}{\partial t} + \frac{\partial(\rho v_1)}{\partial x_1} + \frac{\partial(\rho v_2)}{\partial x_2} = 0, \\
&\frac{\partial(\rho v_1)}{\partial t} + \frac{\partial}{\partial x_1}(\rho v_1^2 + \frac{1}{2} g \rho^2) + \frac{\partial(\rho v_1 v_2)}{\partial x_2} = 0, \qquad t \in (0,1], x \in (0,1)^2 \\
&\frac{\partial(\rho v_2)}{\partial t} + \frac{\partial(\rho v_1 v_2)}{\partial x_1} + \frac{\partial}{\partial x_2}(\rho v_2^2 + \frac{1}{2} g \rho^2) = 0,
\end{aligned}
$$

where ρ is the total fluid column height, v_1 the velocity in the x_1-direction, v_2 the velocity in the x_2-direction, averaged across the vertical column, ρ the fluid density and g the acceleration due to gravity. The above equation can be also written in conservation form:

$$
\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x_1} + \frac{\partial \vec{G}}{\partial x_2} = 0,
$$

where,

$$
\vec{U} = \begin{pmatrix} \rho \\ \rho v_1 \\ \rho v_2 \end{pmatrix}, \quad \vec{F} = \begin{pmatrix} \rho v_1 \\ \rho v_1^2 + \frac{1}{2} g \rho^2 \\ \rho v_1 v_2 \end{pmatrix}, \quad \vec{G} = \begin{pmatrix} \rho v_2 \\ \rho v_1 v_2 \\ \rho v_2^2 + \frac{1}{2} g \rho^2 \end{pmatrix}.
$$

For a given set of initial conditions, the solution operator $\mathcal{G}$ of F.7.4 maps the initial fluid column height and velocity fields to the fluid column height and velocity fields at later times. Again in this problem the input and the output function domains coincide, therefore $\mathcal{X} = \mathcal{Y}$ with $d_x = d_y = 3$ and $d_u = d_s = 3$. The goal is to learn the operator $\mathcal{G} : \mathcal{C}(\mathcal{X}, \mathbb{R}^3) \to \mathcal{C}(\mathcal{X}, \mathbb{R}^3)$.

We set the boundary conditions by considering a solid, impermeable wall with reflective boundaries:

$$v_1 \cdot n_{x_1} + v_2 \cdot n_{x_2} = 0,$$

where $\hat{n} = n_{x_1}\hat{i} + n_{x_2}\hat{j}$ is the unit outward normal of the boundary. We sample the initial conditions by considering a falling droplet of random width, falling from a random height to a random spatial location and zero initial velocities:

$$\begin{aligned} \rho &= 1 + h \exp\left(-((x_1 - \xi)^2 + (x_2 - \zeta)^2)/w\right) \\ v_1 &= v_2 = 0, \end{aligned}$$

where ρ corresponds to the altitude that the droplet falls from, w the width of the droplet, and ξ and ζ the coordinates that the droplet falls in time $t = 0s$. Because the velocities v_1, v_2 are equal to zero in the initial time $t_0 = 0s$ for all realizations, we choose time $t_0 = dt = 0.002s$ as the initial time to make the problem more interesting. Therefore, the input functions become

$$\begin{aligned} \rho &= 1 + h \exp\left(-((x_1 - \xi)^2 + (x_2 - \zeta)^2)/w\right), \\ v_1 &= v_1(dt, y_1, y_2), \\ v_2 &= v_2(dt, y_1, y_2). \end{aligned}$$

We set the random variables h, w, ξ, and ζ to be distributed according to the uniform distributions

$$\begin{aligned} h &= \mathcal{U}(1.5, 2.5), \\ w &= \mathcal{U}(0.002, 0.008), \\ \xi &= \mathcal{U}(0.4, 0.6), \\ \zeta &= \mathcal{U}(0.4, 0.6). \end{aligned}$$

The data set is constructed by sampling the initial conditions along a 32 × 32 grid and solving the forward problem using a Lax-Friedrichs scheme. This provides us with a solution for a 32 × 32 grid which we can use for different operator learning models. Sub-sampling the solution to create the training data set allows us to predict the solution using only a percentage of the available spatial data.

In Figures 15, 16, 17, 18, 19 we provide a visual comparison of the outputs of the trained model for 5 time steps, $t = [0.11, 0.16, 0.21, 0.26, 0.31]s$, for the worst case scenario prediction in the testing data set, along with the point-wise absolute error plot. We see that our model provides favorable solutions for all time steps for an initial condition not in the train data set.

F.7.5 Climate Modeling

For this example, our aim is to approximate the map between the surface air temperature and surface air pressure. In contrast to the previous examples, here we do not consider a relation between these two fields, for example a partial differential equation or a constitutive law. Therefore, we aim to learn a black-box operator which we then use for making predictions of the pressure using the temperature as an input. Therefore, we consider the map

$$T(x) \mapsto P(y),$$

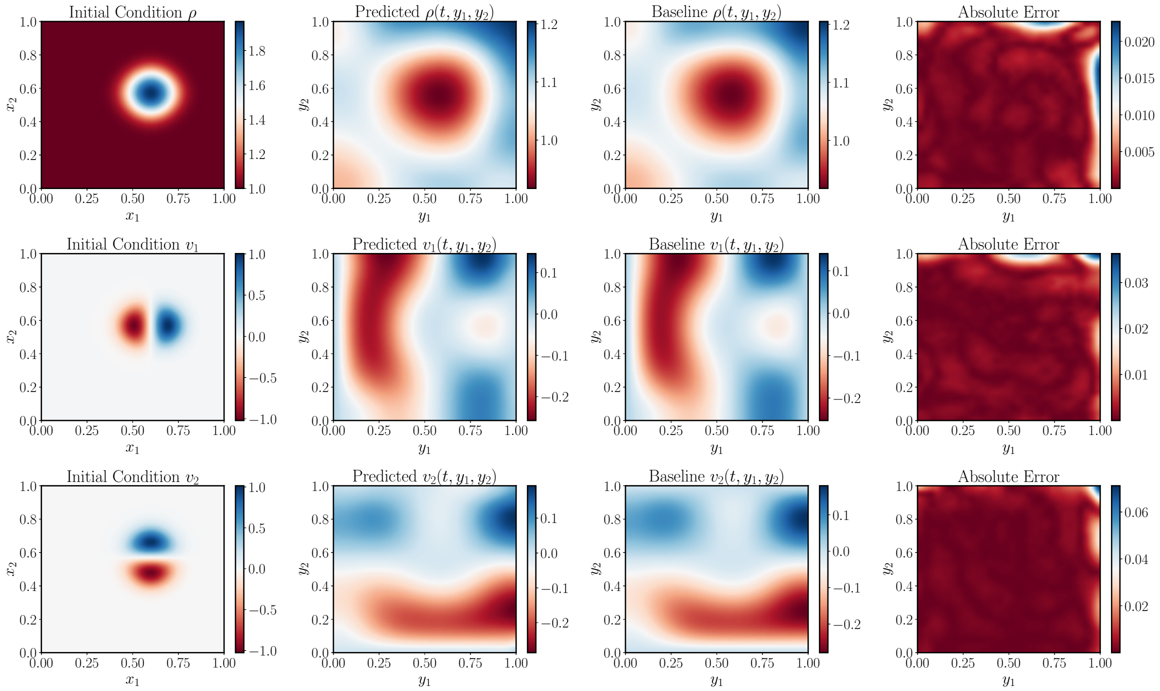


Figure 15: Comparison between the predicted and ground truth solution for the worst case prediction of the Shallow Water Equations benchmark: We present the inputs to the model (initial conditions), the ground truth and the predicted parameters, as well as the absolute error for time instance $t = 0.11s$.

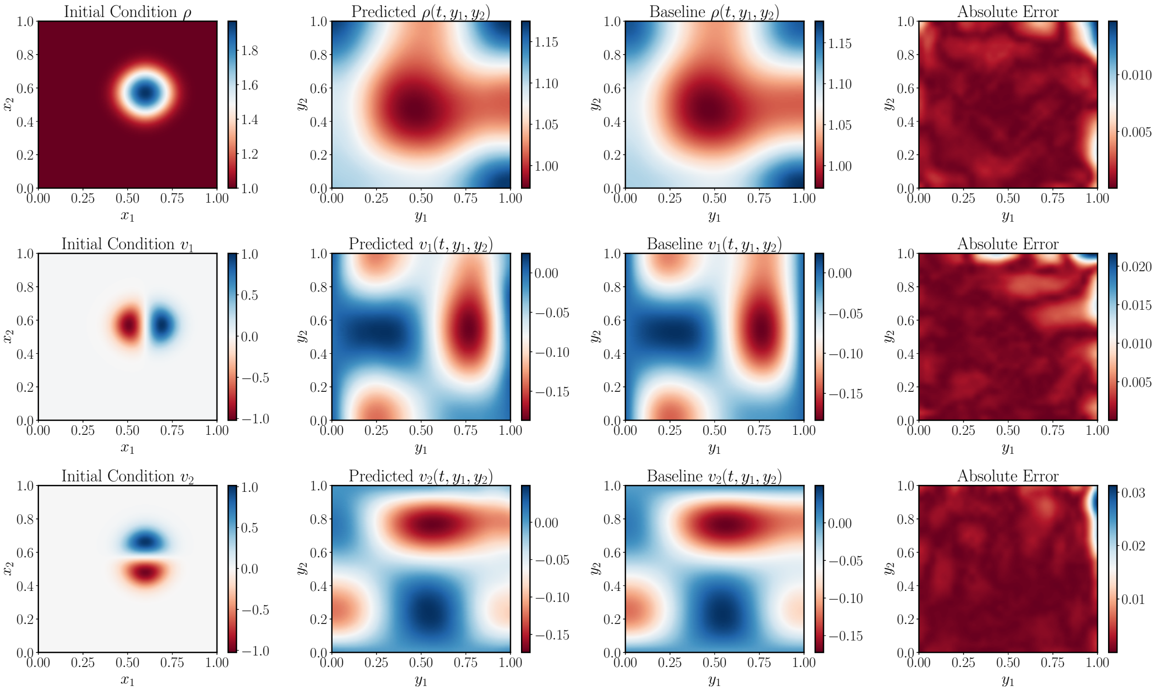


Figure 16: Comparison between the predicted and ground truth solution for the worst case prediction of the Shallow Water Equations benchmark: We present the inputs to the model (initial conditions), the ground truth and the predicted parameters, as well as the absolute error for time instance $t = 0.16s$.

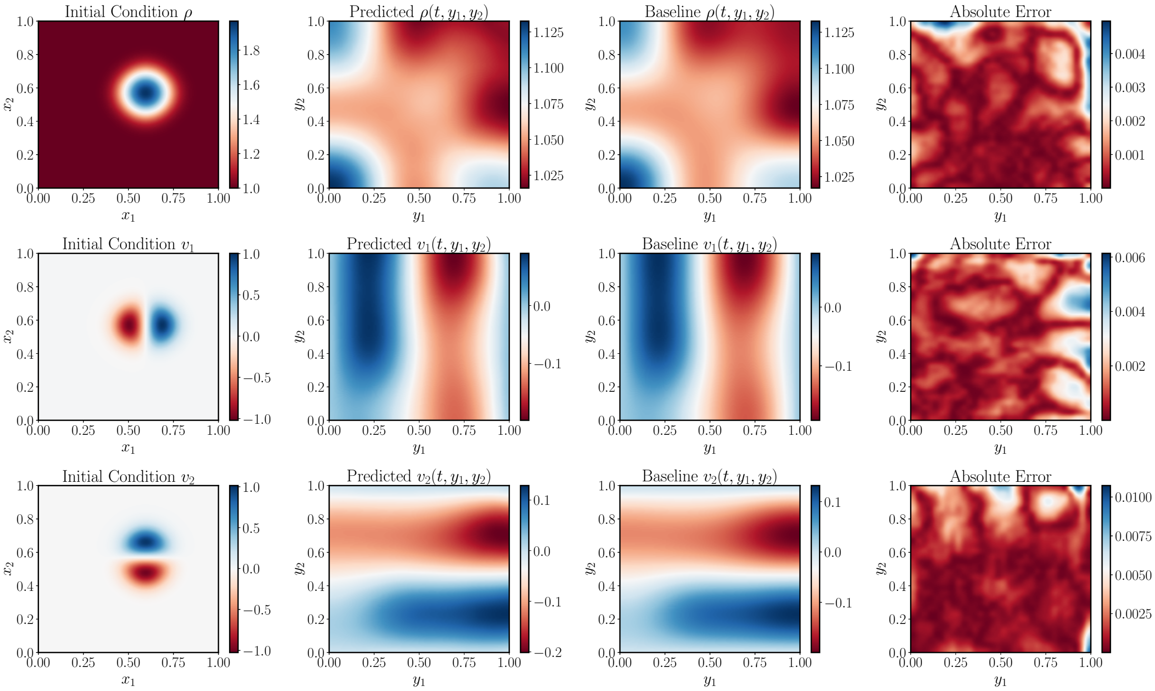


Figure 17: Comparison between the predicted and ground truth solution for the worst case prediction of the Shallow Water Equations benchmark: We present the inputs to the model (initial conditions), the ground truth and the predicted parameters, as well as the absolute error for time instance $t = 0.21s$.

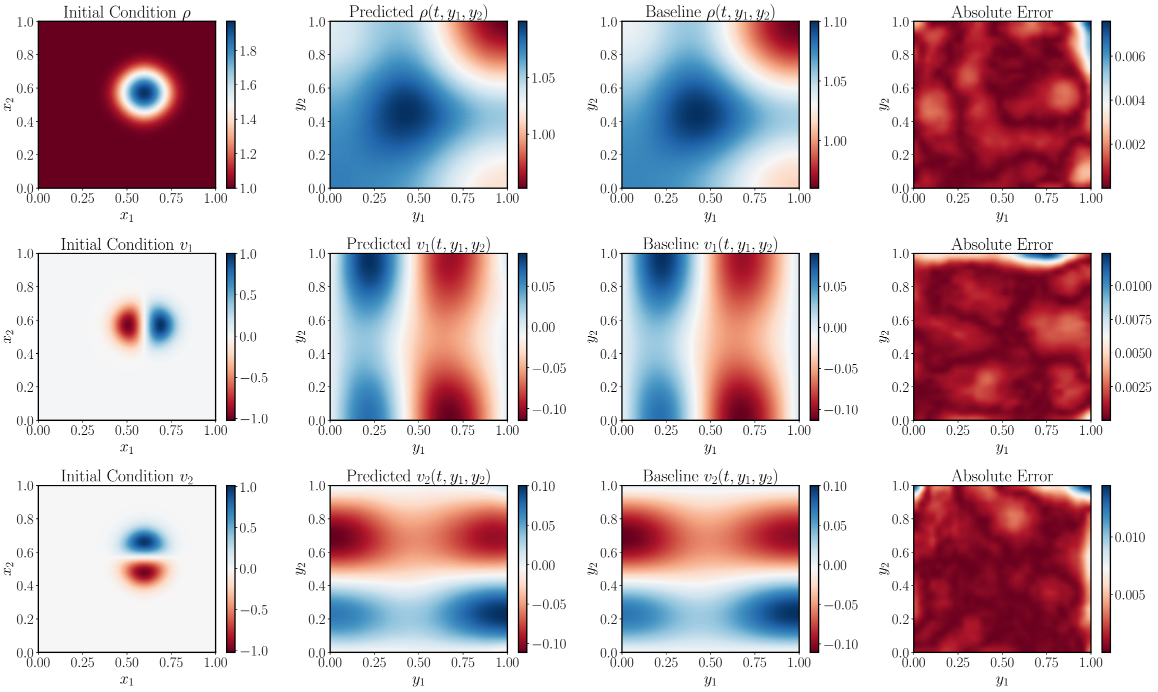


Figure 18: Comparison between the predicted and ground truth solution for the worst case prediction of the Shallow Water Equations benchmark: We present the inputs to the model (initial conditions), the ground truth and the predicted parameters, as well as the absolute error for time instance $t = 0.26s$.

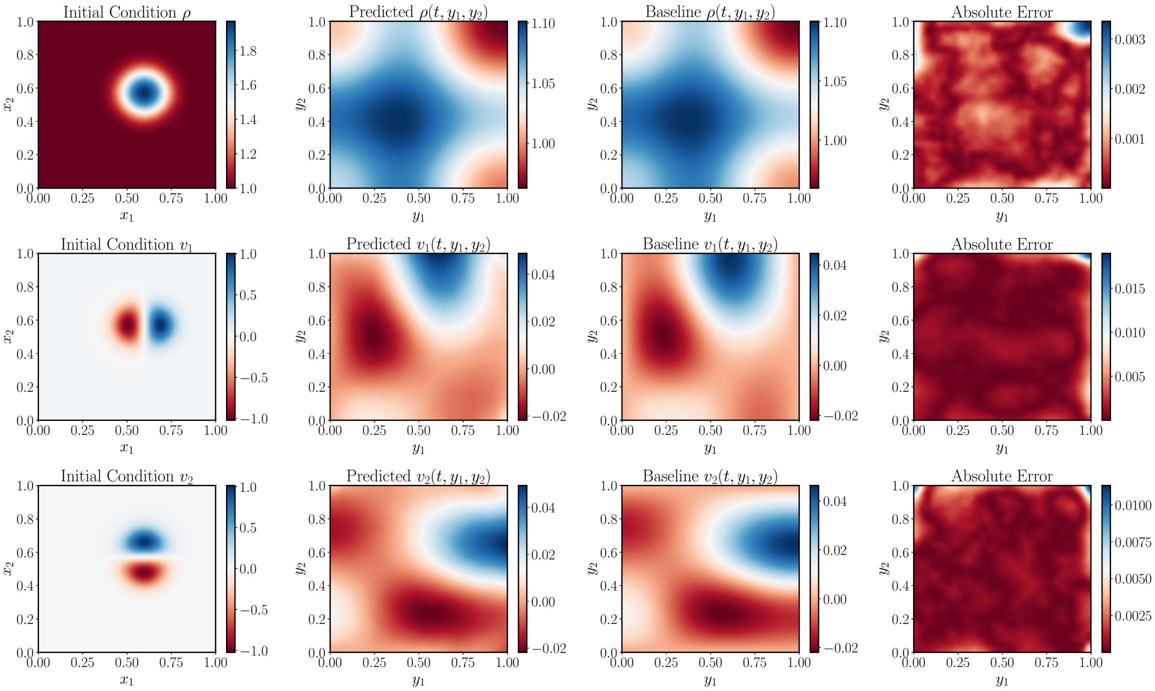


Figure 19: Comparison between the predicted and ground truth solution for the worst case prediction of the Shallow Water Equations benchmark: We present the inputs to the model (initial conditions), the ground truth and the predicted parameters, as well as the absolute error for time instance $t = 0.31s$.

where $x, y \in [-90, 90] \times [0, 360]$ for the latitude and longitude. For a given day of the year the solution operator maps the surface air temperature to the surface air pressure. For this set-up, the input and output function domains coincide which means $\mathcal{X} = \mathcal{Y}$ with $d_x = d_y = 2$ and $d_u = d_s = 1$ because we the input and output functions are scalar fields. We can write the map as $\mathcal{G} : \mathcal{C}(\mathcal{X}, \mathbb{R}) \to \mathcal{C}(\mathcal{X}, \mathbb{R})$.

For constructing the training data set, we consider the Physical Sciences Laboratory meteorological data (Kalnay et al., 1996)(`https://psl.noaa.gov/data/gridded/data.ncep.reanalysis.surface.html`) from the year 2000 to 2005. We consider the different model realizations to be the values of the daily Temperature and Pressure for these 5 years, meaning $N_{train} = 1825$ (excluding the days for leap years). We sub-sample the spatial coverage from 2.5 degree latitude × 2.5 degree longitude global grid (144×73) to 72×72 for creating a regular grid for both the quantities. We consider a test data set consisting of the daily surface air temperature and pressure data from the years 2005 to 2010, meaning $N_{test} = 1825$ (excluding leap years), on an 72×72 grid also.

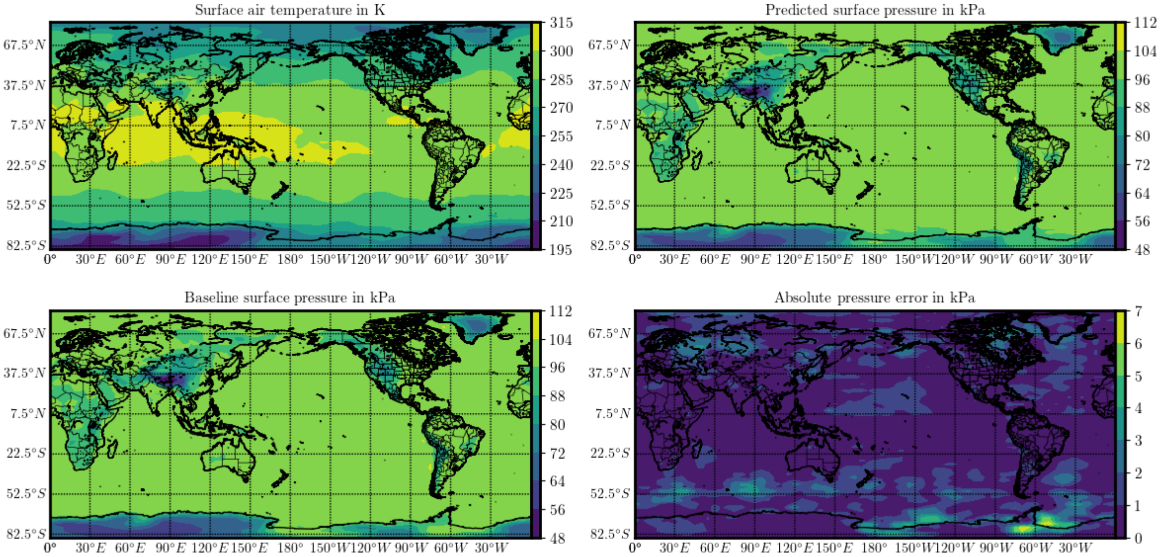


Figure 20: Comparison between the full resolution prediction and base line for the best prediction in the testing data set for the climate modeling benchmark: We present the input temperature field, the output prediction and ground truth, as well as the absolute error between our model's prediction and the ground truth solution.

We present the prediction and the ground truth together with the respective input and error Figure 20.The solution corresponds to the index for which the error vector takes the minimum value. The prediction, the ground truth solution and the absolute error are all presented on a 72×72 grid.

References

Robert J Adler. An introduction to continuity, extrema, and related topics for general gaussian processes. *Lecture Notes-Monograph Series*, 12:i–155, 1990.

Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

Mathieu Andreux, Tomás Angles, Georgios Exarchakis, Roberto Leonarduzzi, Gaspar Rochette, Louis Thiry, John Zarka, Stéphane Mallat, Joakim Andén, Eugene Belilovsky, et al. Kymatio: Scattering transforms in python. *J. Mach. Learn. Res.*, 21(60):1–6, 2020.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Jacob Bear. *Dynamics of fluids in porous media.* Courier Corporation, 2013.

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.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.

Shengze Cai, Zhicheng Wang, Lu Lu, Tamer A Zaki, and George Em Karniadakis. Deepm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *arXiv preprint arXiv:2009.12935*, 2020.

Shuhao Cao. Choose a Transformer: Fourier or Galerkin. *arXiv preprint arXiv:2105.14995*, 2021.

Andrea Caponnetto, Charles A Micchelli, Massimiliano Pontil, and Yiming Ying. Universal multi-task kernels. *The Journal of Machine Learning Research*, 9:1615–1646, 2008.

Kuang-Yu Chang and Chu-Song Chen. A learning framework for age rank estimation based on face images with scattering transform. *IEEE Transactions on Image Processing*, 24(3): 785–798, 2015.

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.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Andreas Christmann and Ingo Steinwart. Universal kernels on non-standard input spaces. In *Advances in neural information processing systems*, pages 406–414. Citeseer, 2010.

James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

I Daubechies. Orthogonal bases of compactly supported wavelets, communications on pure and applied, 1988.

P Clark Di Leoni, Lu Lu, Charles Meneveau, George Karniadakis, and Tamer A Zaki. Deeponet prediction of linear instability waves in high-speed boundary layers. *arXiv preprint arXiv:2105.08697*, 2021.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Xialiang Dou and Tengyuan Liang. Training neural networks as learning data-adaptive kernels: Provable representation and approximation benefits. *Journal of the American Statistical Association*, pages 1–14, 2020.

Alan Edelman, Tomás A Arias, and Steven T Smith. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications*, 20(2): 303–353, 1998.

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. *Scientific reports*, 11 (1):1–14, 2021.

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.

Yuan Gong, Yu-An Chung, and James Glass. Ast: Audio spectrogram Transformer. *arXiv preprint arXiv:2104.01778*, 2021.

Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. *Advances in Neural Information Processing Systems*, 34, 2021.

Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media, 2009.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

John D Hunter. Matplotlib: A 2D graphics environment. *IEEE Annals of the History of Computing*, 9(03):90–95, 2007.

Hachem Kadri, Emmanuel Duflos, Philippe Preux, Stéphane Canu, and Manuel Davy. Nonlinear functional regression: a functional RKHS approach. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 374–380. JMLR Workshop and Conference Proceedings, 2010.

Hachem Kadri, Emmanuel Duflos, Philippe Preux, Stéphane Canu, Alain Rakotomamonjy, and Julien Audiffren. Operator-valued kernels for learning from functional response data. *The Journal of Machine Learning Research*, 17(1):613–666, 2016.

Eugenia Kalnay, Masao Kanamitsu, Robert Kistler, William Collins, Dennis Deaven, Lev Gandin, Mark Iredell, Suranjana Saha, Glenn White, John Woollen, et al. The ncep/ncar 40-year reanalysis project. *Bulletin of the American meteorological Society*, 77(3):437–472, 1996.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22: Art–No, 2021a.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021b.

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.

Samuel Lanthaler, Siddhartha Mishra, and George E Karniadakis. Error estimates for deeponets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1):tnac001, 2022.

Emma Lejeune. Mechanical mnist: A benchmark dataset for mechanical metamodels. *Extreme Mechanics Letters*, 36:100659, 2020.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *arXiv preprint arXiv:2006.09535*, 2020b.

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

Chensen Lin, Zhen Li, Lu Lu, Shengze Cai, Martin Maxey, and George Em Karniadakis. Operator learning for predicting multiscale bubble growth dynamics. *The Journal of Chemical Physics*, 154(10):104118, 2021.

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.

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.

Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.

Charles A Micchelli and Massimiliano Pontil. Kernels for multi–task learning. In *NIPS*, volume 86, page 89. Citeseer, 2004.

Charles A Micchelli and Massimiliano Pontil. On learning vector-valued functions. *Neural computation*, 17(1):177–204, 2005.

Nicholas H Nelsen and Andrew M Stuart. The random feature model for input-output maps between banach spaces. *arXiv preprint arXiv:2005.10224*, 2020.

Houman Owhadi. Do ideas have shape? plato's theory of forms as the continuous limit of artificial neural networks. *arXiv preprint arXiv:2008.03920*, 2020.

Edouard Oyallon, Eugene Belilovsky, and Sergey Zagoruyko. Scaling the scattering transform: Deep hybrid networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5618–5627, 2017.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018.

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*, 32:8026–8037, 2019.

Vern I Paulsen and Mrinal Raghupathi. *An introduction to the theory of reproducing kernel Hilbert spaces*, volume 152. Cambridge university press, 2016.

Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *NIPS*, volume 3, page 5. Citeseer, 2007.

Alvin Rajkomar, Jeffrey Dean, and Isaac Kohane. Machine learning in medicine. *New England Journal of Medicine*, 380(14):1347–1358, 2019.

James O Ramsay. When the data are functions. *Psychometrika*, 47(4):379–396, 1982.

James O Ramsay and CJ Dalzell. Some tools for functional data analysis. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(3):539–561, 1991.

Venkataraman Santhanam, Vlad I Morariu, and Larry S Davis. Generalized deep image to image regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5609–5619, 2017.

John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis.* Cambridge university press, 2004.

Bharath K. Sriperumbudur, Kenji Fukumizu, and Gert R.G. Lanckriet. Universality, characteristic kernels and RKHS embedding of measures. *Journal of Machine Learning Research*, 12(70):2389–2410, 2011. URL `http://jmlr.org/papers/v12/sriperumbudur11a.html`.

Hiroyuki Takeda, Sina Farsiu, and Peyman Milanfar. Kernel regression for image processing and reconstruction. *IEEE Transactions on image processing*, 16(2):349–366, 2007.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Sifan Wang and Paris Perdikaris. Long-time integration of parametric evolution equations with physics-informed deeponets. *arXiv preprint arXiv:2106.05384*, 2021.

Sifan Wang, Mohamed Aziz Bhouri, and Paris Perdikaris. Fast PDE-constrained optimization via self-supervised operator learning. *arXiv preprint arXiv:2110.13297*, 2021a.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Improved architectures and training algorithms for deep operator networks. *arXiv preprint arXiv:2110.01654*, 2021b.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, 7(40):eabi8605, 2021c. doi: 10.1126/sciadv.abi8605.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Zelun Wang and Jyh-Charn Liu. Translating math formula images to LaTeX sequences using deep neural networks with sequence-level training, 2019.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A Nyström-based algorithm for approximating self-attention. *arXiv preprint arXiv:2102.03902*, 2021.