Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs

Nikola Kovachki[*][†]   NKOVACHKI@NVIDIA.COM
Nvidia

Zongyi Li[*]   ZONGYILI@CALTECH.EDU
Caltech

Burigede Liu   BL377@CAM.AC.UK
Cambridge University

Kamyar Azizzadenesheli   KAMYARA@NVIDIA.COM
Nvidia

Kaushik Bhattacharya   BHATTA@CALTECH.EDU
Caltech

Andrew Stuart   ASTUART@CALTECH.EDU
Caltech

Anima Anandkumar   ANIMA@CALTECH.EDU
Caltech

Abstract

The classical development of neural networks has primarily focused on learning mappings between finite dimensional Euclidean spaces or finite sets. We propose a generalization of neural networks to learn operators, termed *neural operators*, that map between infinite dimensional function spaces. We formulate the neural operator as a composition of linear integral operators and nonlinear activation functions. We prove a universal approximation theorem for our proposed neural operator, showing that it can approximate any given nonlinear continuous operator. The proposed neural operators are also discretization-invariant, i.e., they share the same model parameters among different discretization of the underlying function spaces. Furthermore, we introduce four classes of efficient parameterization, viz., graph neural operators, multi-pole graph neural operators, low-rank neural operators, and Fourier neural operators. An important application for neural operators is learning surrogate maps for the solution operators of partial differential equations (PDEs). We consider standard PDEs such as the Burgers, Darcy subsurface flow, and the Navier-Stokes equations, and show that the proposed neural operators have superior performance compared to existing machine learning based methodologies, while being several orders of magnitude faster than conventional PDE solvers.

Keywords: Deep Learning, Operator Learning, Discretization-Invariance, Partial Differential Equations, Navier-Stokes Equation.

---

[*]. Equal contribution.

[†]. Majority of the work was completed while the author was at Caltech.

1. Introduction

Learning mappings between function spaces has widespread applications in science and engineering. For instance, for solving differential equations, the input is a coefficient function and the output is a solution function. A straightforward solution to this problem is to simply discretize the infinite-dimensional input and output function spaces into finite-dimensional grids, and apply standard learning models such as neural networks. However, this limits applicability since the learned neural network model may not generalize well to different discretizations, beyond the discretization grid of the training data.

To overcome these limitations of standard neural networks, we formulate a new deep-learning framework for learning operators, called *neural operators*, which directly map between function spaces on bounded domains. Since our neural operator is designed on function spaces, they can be discretized by a variety of different methods, and at different levels of resolution, without the need for re-training. In contrast, standard neural network architectures depend heavily on the discretization of training data: new architectures with new parameters may be needed to achieve the same error for data with varying discretization. We also propose the notion of *discretization-invariant* models and prove that our neural operators satisfy this property, while standard neural networks do not.

1.1 Our Approach

Discretization-Invariant Models. We formulate a precise mathematical notion of discretization invariance. We require any discretization-invariant model with a fixed number of parameters to satisfy the following:

1. acts on any discretization of the input function, i.e. accepts any set of points in the input domain,

2. can be evaluated at any point of the output domain,

3. converges to a continuum operator as the discretization is refined.

The first two requirements of accepting any input and output points in the domain is a natural requirement for discretization invariance, while the last one ensures consistency in the limit as the discretization is refined. For example, families of graph neural networks (Scarselli et al., 2008) and transformer models (Vaswani et al., 2017) are resolution invariant, i.e., they can receive inputs at any resolution, but they fail to converge to a continuum operator as discretization is refined. Moreover, we require the models to have a fixed number of parameters; otherwise, the number of parameters becomes unbounded in the limit as the discretization is refined, as shown in Figure 1. Thus the notion of discretization invariance allows us to define neural operator models that are consistent in function spaces and can be applied to data given at any resolution and on any mesh. We also establish that standard neural network models are not discretization invariant.

Neural Operators. We introduce the concept of neural operators for learning operators that are mappings between infinite-dimensional function spaces. We propose neural operator architectures to be multi-layers where layers are themselves operators composed with non-linear activations. This ensures that that the overall end-to-end composition is an operator, and thus satisfies the discretization invariance property. The key design choice for neural operator is the operator layers. To keep it simple, we limit ourselves to layers that are linear operators. Since these layers are composed with
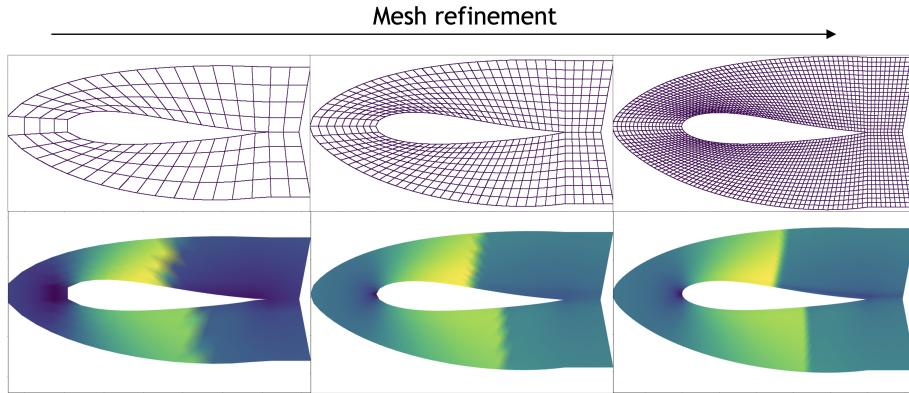
Mesh refinement

Figure 1: Discretization Invariance
An discretization-invariant operator has convergent predictions on a mesh refinement.

non-linear activations, we obtain neural operator models that are expressive and able to capture any continuous operator. The latter property is known as universal approximation.

The above line of reasoning for neural operator design follows closely the design of standard neural networks, where linear layers (e.g. matrix multiplication, convolution) are composed with non-linear activations, and we have universal approximation of continuous functions defined on compact domains (Hornik et al., 1989). Neural operators replace finite-dimensional linear layers in neural networks with linear operators in function spaces.

We formally establish that neural operator models with a fixed number of parameters satisfy discretization invariance. We further show that neural operators models are universal approximators of continuous operators acting between Banach spaces, and can uniformly approximate any continuous operator defined on a compact set of a Banach space. **Neural operators are the only known class of models that guarantee both discretization-invariance and universal approximation.** See Table 1 for a comparison among the deep learning models. Previous deep learning models are mostly defined on a fixed grid, and removing, adding, or moving grid points generally makes these models no longer applicable. Thus, they are not discretization invariant.

We propose several design choices for the linear operator layers in neural operator such as a parameterized integral operator or through multiplication in the spectral domain as shown in Figure 2. Specifically, we propose four practical methods for implementing the neural operator framework: graph-based operators, low-rank operators, multipole graph-based operators, and Fourier operators. Specifically, for graph-based operators, we develop a Nyström extension to connect the integral operator formulation of the neural operator to families of graph neural networks (GNNs) on arbitrary grids. For Fourier operators, we consider the spectral domain formulation of the neural operator which leads to efficient algorithms in settings where fast transform methods are applicable.

We include an exhaustive numerical study of the four formulations of neural operators. Numerically, we show that the proposed methodology consistently outperforms all existing deep learning methods even on the resolutions for which the standard neural networks were designed. For the two-dimensional Navier-Stokes equation, when learning the entire flow map, the method achieves $< 1\%$ error for a Reynolds number of 20 and 8% error for a Reynolds number of 200.

The proposed Fourier neural operator (FNO) has an inference time that is three orders of magnitude faster than the pseudo-spectral method used to generate the data for the Navier-Stokes problem
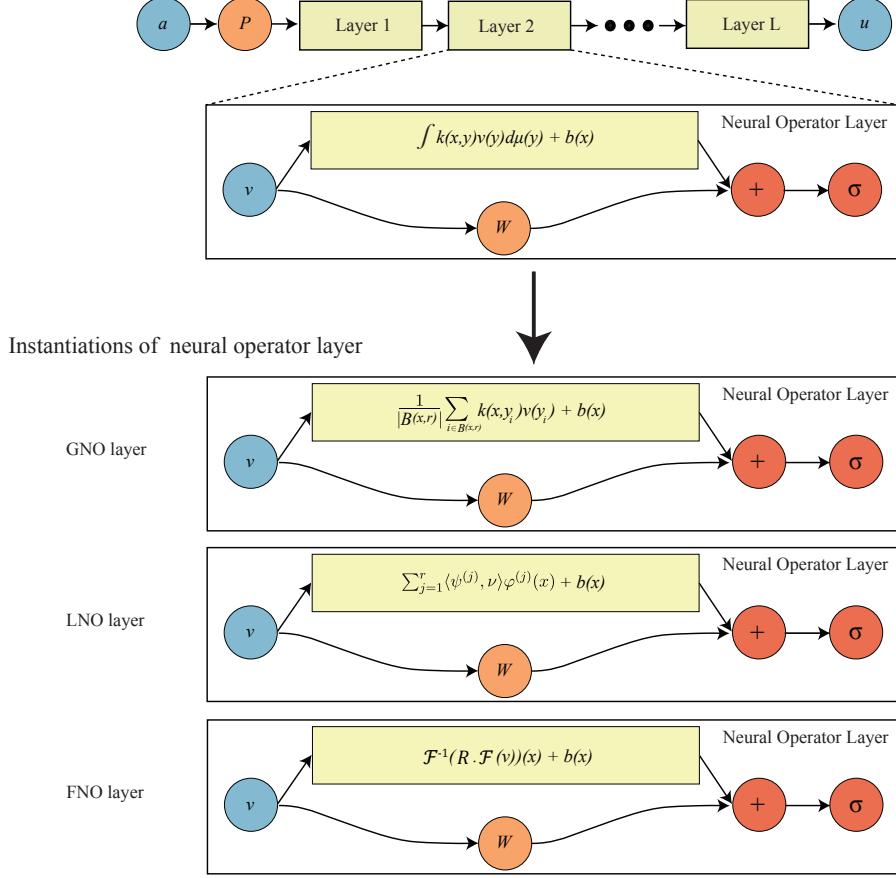
$$\int k(x,y)v(y)d\mu(y) + b(x)$$

Neural Operator Layer

Instantiations of neural operator layer

GNO layer

$$\frac{1}{|B(x,r)|}\sum_{i\in B(x,r)} k(x,y_i)v(y_i) + b(x)$$

Neural Operator Layer

LNO layer

$$\sum_{j=1}^{r}\langle\psi^{(j)},\nu\rangle\varphi^{(j)}(x) + b(x)$$

Neural Operator Layer

FNO layer

$$\mathcal{F}^{-1}(R\cdot\mathcal{F}(v))(x) + b(x)$$

Neural Operator Layer

Figure 2: Neural operator architecture schematic
The input function a is passed to a pointwise lifting operator P that is followed by T layers of integral operators and pointwise non-linearity operations σ. In the end, the pointwise projection operator Q outputs the function u. Three instantiation of neural operator layers, GNO, LNO, and FNO are provided.

(Chandler and Kerswell, 2013) – $0.005s$ compared to the $2.2s$ on a 256×256 uniform spatial grid. Despite its tremendous speed advantage, the method does not suffer from accuracy degradation when used in downstream applications such as solving Bayesian inverse problems. Furthermore, we demonstrate that FNO is robust to noise on the testing problems we consider here.

1.2 Background and Context

Data-driven approaches for solving PDEs. Over the past decades, significant progress has been made in formulating (Gurtin, 1982) and solving (Johnson, 2012) the governing PDEs in many scientific fields from micro-scale problems (e.g., quantum and molecular dynamics) to macro-scale applications (e.g., civil and marine engineering). Despite the success in the application of PDEs to solve real-world problems, two significant challenges remain: (1) identifying the governing model for complex systems; (2) efficiently solving large-scale nonlinear systems of equations.

Model Property	NNs	DeepONets	Interpolation	Neural Operators
Discretization Invariance	✗	✗	✓	✓
Is the output a function?	✗	✓	✓	✓
Can query the output at any point?	✗	✓	✓	✓
Can take the input at any point?	✗	✗	✓	✓
Universal Approximation	✗	✓	✗	✓

Table 1: Comparison of deep learning models. The first row indicates whether the model is discretization invariant. The second and third rows indicate whether the output and input are a functions. The fourth row indicates whether the model class is a universal approximator of operators. Neural Operators are discretization invariant deep learning methods that output functions and can approximate any operator.

Identifying and formulating the underlying PDEs appropriate for modeling a specific problem usually requires extensive prior knowledge in the corresponding field which is then combined with universal conservation laws to design a predictive model. For example, modeling the deformation and failure of solid structures requires detailed knowledge of the relationship between stress and strain in the constituent material. For complicated systems such as living cells, acquiring such knowledge is often elusive and formulating the governing PDE for these systems remains prohibitive, or the models proposed are too simplistic to be informative. The possibility of acquiring such knowledge from data can revolutionize these fields. Second, solving complicated nonlinear PDE systems (such as those arising in turbulence and plasticity) is computationally demanding and can often make realistic simulations intractable. Again the possibility of using instances of data to design fast approximate solvers holds great potential for accelerating numerous problems.

Learning PDE Solution Operators. In PDE applications, the governing differential equations are by definition local, whilst the solution operator exhibits non-local properties. Such non-local effects can be described by integral operators explicitly in the spatial domain, or by means of spectral domain multiplication; convolution is an archetypal example. For integral equations, the graph approximations of Nyström type (Belongie et al., 2002) provide a consistent way of connecting different grid or data structures arising in computational methods and understanding their continuum limits (Von Luxburg et al., 2008; Trillos and Slepčev, 2018; Trillos et al., 2020). For spectral domain calculations, there are well-developed tools that exist for approximating the continuum (Boyd, 2001; Trefethen, 2000). However, these approaches for approximating integral operators are not data-driven. Neural networks present a natural approach for learning-based integral operator approximations since they can incorporate non-locality. However, standard neural networks are limited to the discretization of training data and hence, offer a poor approximation to the integral operator. We tackle this issue here by proposing the framework of neural operators.

Properties of existing deep-learning models. Previous deep learning models are mostly defined on a fixed grid, and removing, adding, or moving grid points generally makes these models no longer applicable, as seen in Table 1. Thus, they are not discretization invariant. In general, standard neural networks (NN) (such as Multilayer perceptron (MLP), convolution neural networks (CNN), Resnet, and Vision Transformers (ViT)) that take the input grid and output grid as finite-dimensional vectors

are not discretization-invariant since their input and output have to be at the fixed grid with fixed location. On the other hand, the pointwise neural networks used in PINNs (Raissi et al., 2019) that take each coordinate as input are discretization-invariant since it can be applied at each location in parallel. However PINNs only represent the solution function of one instance and it does not learn the map from the input functions to the output solution functions. A special class of neural networks is convolution neural networks (CNNs). CNNs also do not converge with grid refinement since their respective fields change with different input grids. On the other hand, if normalized by the grid size, CNNs can be applied to uniform grids with different resolutions, which converge to differential operators, in a similar fashion to the finite difference method. Interpolation is a baseline approach to achieve discretization-invariance. While NNs+Interpolation (or in general any finite-dimensional neural networks+Interpolation) are resolution invariant and their outputs can be queried at any point, they are not universal approximators of operators since the dimension of input and output of the internal CNN model is defined to a bounded number. DeepONets (Lu et al., 2019) are a class of operators that have the universal approximation property. DeepONets consist of a branch net and a trunk net. The trunk net allows queries at any point, but the branch net constrains the input to fixed locations; however it is possible to modify the branch net to make the methodology discretization invariant, for example by using the PCA-based approach as used in (De Hoop et al., 2022).

Furthermore, we show transformers (Vaswani et al., 2017) are special cases of neural operators with structured kernels that can be used with varying grids to represent the input function. However, the commonly used vision-based extensions of transformers, e.g., ViT (Dosovitskiy et al., 2020), use convolutions on patches to generate tokens, and therefore, they are not discretization-invariant models.

We also show that when our proposed neural operators are applied only on fixed grids, the resulting architectures coincide with neural networks and other operator learning frameworks. In such reductions, point evaluations of the input functions are available on the grid points. In particular, we show that the recent work of DeepONets (Lu et al., 2019), which are maps from finite-dimensional spaces to infinite dimensional spaces are special cases of neural operators architecture when neural operators are limited only to fixed input grids. Moreover, by introducing an adjustment to the DeepONet architecture, we propose the DeepONet-Operator model that fits into the full operator learning framework of maps between function spaces.

2. Learning Operators

In subsection 2.1, we describe the generic setting of PDEs to make the discussions in the following setting concrete. In subsection 2.2, we outline the general problem of operator learning as well as our approach to solving it. In subsection 2.3, we discuss the functional data that is available and how we work with it numerically.

2.1 Generic Parametric PDEs

We consider the generic family of PDEs of the following form,

$$
\begin{aligned}
(\mathsf{L}_a u)(x) &= f(x), && x \in D, \\
u(x) &= 0, && x \in \partial D,
\end{aligned}
\tag{1}
$$

for some $a \in \mathcal{A}$, $f \in \mathcal{U}^*$ and $D \subset \mathbb{R}^d$ a bounded domain. We assume that the solution $u : D \to \mathbb{R}$ lives in the Banach space $\mathcal{U}$ and $\mathsf{L}_a : \mathcal{A} \to \mathcal{L}(\mathcal{U}; \mathcal{U}^*)$ is a mapping from the parameter Banach space $\mathcal{A}$ to the space of (possibly unbounded) linear operators mapping $\mathcal{U}$ to its dual $\mathcal{U}^*$. A natural operator which arises from this PDE is $\mathcal{G}^\dagger := \mathsf{L}_a^{-1} f : \mathcal{A} \to \mathcal{U}$ defined to map the parameter to the solution $a \mapsto u$. A simple example that we study further in Section 6.2 is when L_a is the weak form of the second-order elliptic operator $-\nabla \cdot (a\nabla)$ subject to homogeneous Dirichlet boundary conditions. In this setting, $\mathcal{A} = L^\infty(D; \mathbb{R}_+)$, $\mathcal{U} = H_0^1(D; \mathbb{R})$, and $\mathcal{U}^* = H^{-1}(D; \mathbb{R})$. When needed, we will assume that the domain D is discretized into $K \in \mathbb{N}$ points and that we observe $N \in \mathbb{N}$ pairs of coefficient functions and (approximate) solution functions $\{a^{(i)}, u^{(i)}\}_{i=1}^N$ that are used to train the model (see Section 2.2). We assume that $a^{(i)}$ are i.i.d. samples from a probability measure μ supported on $\mathcal{A}$ and $u^{(i)}$ are the pushforwards under $\mathcal{G}^\dagger$.

2.2 Problem Setting

Our goal is to learn a mapping between two infinite dimensional spaces by using a finite collection of observations of input-output pairs from this mapping. We make this problem concrete in the following setting. Let $\mathcal{A}$ and $\mathcal{U}$ be Banach spaces of functions defined on bounded domains $D \subset \mathbb{R}^d$, $D' \subset \mathbb{R}^{d'}$ respectively and $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ be a (typically) non-linear map. Suppose we have observations $\{a^{(i)}, u^{(i)}\}_{i=1}^N$ where $a^{(i)} \sim \mu$ are i.i.d. samples drawn from some probability measure μ supported on $\mathcal{A}$ and $u^{(i)} = \mathcal{G}^\dagger(a^{(i)})$ is possibly corrupted with noise. We aim to build an approximation of $\mathcal{G}^\dagger$ by constructing a parametric map

$$\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}, \quad \theta \in \mathbb{R}^p \tag{2}$$

with parameters from the finite-dimensional space $\mathbb{R}^p$ and then choosing $\theta^\dagger \in \mathbb{R}^p$ so that $\mathcal{G}_{\theta^\dagger} \approx \mathcal{G}^\dagger$.

We will be interested in controlling the error of the approximation on average with respect to μ. In particular, assuming $\mathcal{G}^\dagger$ is μ-measurable, we will aim to control the $L_\mu^2(\mathcal{A}; \mathcal{U})$ Bochner norm of the approximation

$$\|\mathcal{G}^\dagger - \mathcal{G}_\theta\|_{L_\mu^2(\mathcal{A}; \mathcal{U})}^2 = \mathbb{E}_{a \sim \mu} \|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|_{\mathcal{U}}^2 = \int_{\mathcal{A}} \|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|_{\mathcal{U}}^2 \, d\mu(a). \tag{3}$$

This is a natural framework for learning in infinite-dimensions as one could seek to solve the associated empirical-risk minimization problem

$$\min_{\theta \in \mathbb{R}^p} \mathbb{E}_{a \sim \mu} \|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|_{\mathcal{U}}^2 \approx \min_{\theta \in \mathbb{R}^p} \frac{1}{N} \sum_{i=1}^N \|u^{(i)} - \mathcal{G}_\theta(a^{(i)})\|_{\mathcal{U}}^2 \tag{4}$$

which directly parallels the classical finite-dimensional setting (Vapnik, 1998). As well as using error measured in the Bochner norm, we will also consider the setting where error is measured uniformly over compact sets of $\mathcal{A}$. In particular, given any $K \subset \mathcal{A}$ compact, we consider

$$\sup_{a \in K} \|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|_{\mathcal{U}} \tag{5}$$

which is a more standard error metric in the approximation theory literature. Indeed, the classic approximation theory of neural networks in formulated analogously to equation (5) (Hornik et al., 1989).

In Section 9 we show that, for the architecture we propose and given any desired error tolerance, there exists $p \in \mathbb{N}$ and an associated parameter $\theta^\dagger \in \mathbb{R}^p$, so that the loss (3) or (5) is less than the specified tolerance. However, we do not address the challenging open problems of characterizing the error with respect to either (a) a fixed parameter dimension p or (b) a fixed number of training samples N. Instead, we approach this in the empirical test-train setting where we minimize (4) based on a fixed training set and approximate (3) from new samples that were not seen during training. Because we conceptualize our methodology in the infinite-dimensional setting, all finite-dimensional approximations can share a common set of network parameters which are defined in the (approximation-free) infinite-dimensional setting. In particular, our architecture does not depend on the way the functions $a^{(i)}, u^{(i)}$ are discretized. . The notation used through out this paper, along with a useful summary table, may be found in Appendix A.

2.3 Discretization

Since our data $a^{(i)}$ and $u^{(i)}$ are, in general, functions, to work with them numerically, we assume access only to their point-wise evaluations. To illustrate this, we will continue with the example of the preceding paragraph. For simplicity, assume $D = D'$ and suppose that the input and output functions are both real-valued. Let $D^{(i)} = \{x_\ell^{(i)}\}_{\ell=1}^L \subset D$ be a L-point discretization of the domain D and assume we have observations $a^{(i)}|_{D^{(i)}}, u^{(i)}|_{D^{(i)}} \in \mathbb{R}^L$, for a finite collection of input-output pairs indexed by j. In the next section, we propose a kernel inspired graph neural network architecture which, while trained on the discretized data, can produce the solution $u(x)$ for any $x \in D$ given an input $a \sim \mu$. In particular, our discretized architecture maps into the space $\mathcal{U}$ and not into a discretization thereof. Furthermore our parametric operator class is consistent, in that, given a fixed set of parameters, refinement of the input discretization converges to the true functions space operator. We make this notion precise in what follows and refer to architectures that possess it as function space architectures, mesh-invariant architectures, or *discretization-invariant architectures*. *

Definition 1 *We call a discrete refinement of the domain $D \subset \mathbb{R}^d$ any sequence of nested sets $D_1 \subset D_2 \subset \cdots \subset D$ with $|D_L| = L$ for any $L \in \mathbb{N}$ such that, for any $\epsilon > 0$, there exists a number $L = L(\epsilon) \in \mathbb{N}$ such that*

$$D \subseteq \bigcup_{x \in D_L} \{y \in \mathbb{R}^d : \|y - x\|_2 < \epsilon\}.$$

Definition 2 *Given a discrete refinement $(D_L)_{L=1}^\infty$ of the domain $D \subset \mathbb{R}^d$, any member D_L is called a discretization of D.*

Since $a : D \subset \mathbb{R}^d \to \mathbb{R}^m$, pointwise evaluation of the function (discretization) at a set of L points gives rise to the data set $\{(x_\ell, a(x_\ell))\}_{\ell=1}^L$. Note that this may be viewed as a vector in $\mathbb{R}^{Ld} \times \mathbb{R}^{Lm}$. An example of the mesh refinement is given in Figure 1.

Definition 3 *Suppose $\mathcal{A}$ is a Banach space of $\mathbb{R}^m$-valued functions on the domain $D \subset \mathbb{R}^d$. Let $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ be an operator, D_L be an L-point discretization of D, and $\hat{\mathcal{G}} : \mathbb{R}^{Ld} \times \mathbb{R}^{Lm} \to \mathcal{U}$ some map. For any $K \subset \mathcal{A}$ compact, we define the discretized uniform risk as*

$$R_K(\mathcal{G}, \hat{\mathcal{G}}, D_L) = \sup_{a \in K} \|\hat{\mathcal{G}}(D_L, a|_{D_L}) - \mathcal{G}(a)\|_{\mathcal{U}}.$$

∗. Note that the meaning of the indexing of sets $D_\bullet$ in the following definition differs that used earlier in this paragraph.

Definition 4 *Let $\Theta \subseteq \mathbb{R}^p$ be a finite dimensional parameter space and $\mathcal{G} : \mathcal{A} \times \Theta \to \mathcal{U}$ a map representing a parametric class of operators with parameters $\theta \in \Theta$. Given a discrete refinement $(D_n)_{n=1}^{\infty}$ of the domain $D \subset \mathbb{R}^d$, we say $\mathcal{G}$ is discretization-invariant if there exists a sequence of maps $\hat{\mathcal{G}}_1, \hat{\mathcal{G}}_2, \ldots$ where $\hat{\mathcal{G}}_L : \mathbb{R}^{Ld} \times \mathbb{R}^{Lm} \times \Theta \to \mathcal{U}$ such that, for any $\theta \in \Theta$ and any compact set $K \subset \mathcal{A}$,*

$$\lim_{L \to \infty} R_K(\mathcal{G}(\cdot, \theta), \hat{\mathcal{G}}_L(\cdot, \cdot, \theta), D_L) = 0.$$

We prove that the architectures proposed in Section 3 are discretization-invariant. We further verify this claim numerically by showing that the approximation error is approximately constant as we refine the discretization. Such a property is highly desirable as it allows a transfer of solutions between different grid geometries and discretization sizes with a single architecture that has a fixed number of parameters.

We note that, while the application of our methodology is based on having point-wise evaluations of the function, it is not limited by it. One may, for example, represent a function numerically as a finite set of truncated basis coefficients. Invariance of the representation would then be with respect to the size of this set. Our methodology can, in principle, be modified to accommodate this scenario through a suitably chosen architecture. We do not pursue this direction in the current work. From the construction of neural operators, when the input and output functions are evaluated on fixed grids, the architecture of neural operators on these fixed grids coincide with the class of neural networks.

3. Neural Operators

In this section, we outline the neural operator framework. We assume that the input functions $a \in \mathcal{A}$ are $\mathbb{R}^{d_a}$-valued and defined on the bounded domain $D \subset \mathbb{R}^d$ while the output functions $u \in \mathcal{U}$ are $\mathbb{R}^{d_u}$-valued and defined on the bounded domain $D' \subset \mathbb{R}^{d'}$. The proposed architecture $\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}$ has the following overall structure:

1. **Lifting**: Using a pointwise function $\mathbb{R}^{d_a} \to \mathbb{R}^{d_{v_0}}$, map the input $\{a : D \to \mathbb{R}^{d_a}\} \mapsto \{v_0 : D \to \mathbb{R}^{d_{v_0}}\}$ to its first hidden representation. Usually, we choose $d_{v_0} > d_a$ and hence this is a lifting operation performed by a fully local operator.

2. **Iterative Kernel Integration**: For $t = 0, \ldots, T - 1$, map each hidden representation to the next $\{v_t : D_t \to \mathbb{R}^{d_{v_t}}\} \mapsto \{v_{t+1} : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}\}$ via the action of the sum of a local linear operator, a non-local integral kernel operator, and a bias function, composing the sum with a fixed, pointwise nonlinearity. Here we set $D_0 = D$ and $D_T = D'$ and impose that $D_t \subset \mathbb{R}^{d_t}$ is a bounded domain.[†]

3. **Projection**: Using a pointwise function $\mathbb{R}^{d_{v_T}} \to \mathbb{R}^{d_u}$, map the last hidden representation $\{v_T : D' \to \mathbb{R}^{d_{v_T}}\} \mapsto \{u : D' \to \mathbb{R}^{d_u}\}$ to the output function. Analogously to the first step, we usually pick $d_{v_T} > d_u$ and hence this is a projection step performed by a fully local operator.

---

†. The indexing of sets $D_\bullet$ here differs from the two previous indexings used in Subsection 2.3. The index t is not the physical time, but the iteration (layer) in the model architecture.

The outlined structure mimics that of a finite dimensional neural network where hidden representations are successively mapped to produce the final output. In particular, we have

$$\mathcal{G}_\theta := \mathcal{Q} \circ \sigma_T(W_{T-1} + \mathcal{K}_{T-1} + b_{T-1}) \circ \cdots \circ \sigma_1(W_0 + \mathcal{K}_0 + b_0) \circ \mathcal{P} \tag{6}$$

where $\mathcal{P} : \mathbb{R}^{d_a} \to \mathbb{R}^{d_{v_0}}$, $\mathcal{Q} : \mathbb{R}^{d_{v_T}} \to \mathbb{R}^{d_u}$ are the local lifting and projection mappings respectively, $W_t \in \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}}$ are local linear operators (matrices), $\mathcal{K}_t : \{v_t : D_t \to \mathbb{R}^{d_{v_t}}\} \to \{v_{t+1} : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}\}$ are integral kernel operators, $b_t : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}$ are bias functions, and σ_t are fixed activation functions acting locally as maps $\mathbb{R}^{v_{t+1}} \to \mathbb{R}^{v_{t+1}}$ in each layer. The output dimensions $d_{v_0}, \ldots, d_{v_T}$ as well as the input dimensions $d_1, \ldots, d_{T-1}$ and domains of definition $D_1, \ldots, D_{T-1}$ are hyperparameters of the architecture. By local maps, we mean that the action is pointwise, in particular, for the lifting and projection maps, we have $(\mathcal{P}(a))(x) = \mathcal{P}(a(x))$ for any $x \in D$ and $(\mathcal{Q}(v_T))(x) = \mathcal{Q}(v_T(x))$ for any $x \in D'$ and similarly, for the activation, $(\sigma(v_{t+1}))(x) = \sigma(v_{t+1}(x))$ for any $x \in D_{t+1}$. The maps, $\mathcal{P}$, $\mathcal{Q}$, and σ_t can thus be thought of as defining Nemitskiy operators (Dudley and Norvaisa, 2011, Chapters 6,7) when each of their components are assumed to be Borel measurable. This interpretation allows us to define the general neural operator architecture when pointwise evaluation is not well-defined in the spaces $\mathcal{A}$ or $\mathcal{U}$ e.g. when they are Lebesgue, Sobolev, or Besov spaces.

The crucial difference between the proposed architecture (6) and a standard feed-forward neural network is that all operations are directly defined in function space (noting that the activation funtions, $\mathcal{P}$ and $\mathcal{Q}$ are all interpreted through their extension to Nemitskiy operators) and therefore do not depend on any discretization of the data. Intuitively, the lifting step locally maps the data to a space where the non-local part of $\mathcal{G}^\dagger$ is easier to capture. We confirm this intuition numerically in Section 7; however, we note that for the theory presented in Section 9 it suffices that $\mathcal{P}$ is the identity map. The non-local part of $\mathcal{G}^\dagger$ is then learned by successively approximating using integral kernel operators composed with a local nonlinearity. Each integral kernel operator is the function space analog of the weight matrix in a standard feed-forward network since they are infinite-dimensional linear operators mapping one function space to another. We turn the biases, which are normally vectors, to functions and, using intuition from the ResNet architecture (He et al., 2016), we further add a local linear operator acting on the output of the previous layer before applying the nonlinearity. The final projection step simply gets us back to the space of our output function. We concatenate in $\theta \in \mathbb{R}^p$ the parameters of $\mathcal{P}$, $\mathcal{Q}$, $\{b_t\}$ which are usually themselves shallow neural networks, the parameters of the kernels representing $\{\mathcal{K}_t\}$ which are again usually shallow neural networks, and the matrices $\{W_t\}$. We note, however, that our framework is general and other parameterizations such as polynomials may also be employed.

Integral Kernel Operators We define three version of the integral kernel operator $\mathcal{K}_t$ used in (6). For the first, let $\kappa^{(t)} \in C(D_{t+1} \times D_t; \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}})$ and let ν_t be a Borel measure on D_t. Then we define $\mathcal{K}_t$ by

$$(\mathcal{K}_t(v_t))(x) = \int_{D_t} \kappa^{(t)}(x, y) v_t(y) \, \mathrm{d}\nu_t(y) \qquad \forall x \in D_{t+1}. \tag{7}$$

Normally, we take ν_t to simply be the Lebesgue measure on $\mathbb{R}^{d_t}$ but, as discussed in Section 4, other choices can be used to speed up computation or aid the learning process by building in *a priori* information. The choice of integral kernel operator in (7) defines the basic form of the neural operator and is the one we analyze in Section 9 and study most in the numerical experiments of Section 7.

10

For the second, let $\kappa^{(t)} \in C(D_{t+1} \times D_t \times \mathbb{R}^{d_a} \times \mathbb{R}^{d_a}; \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}})$. Then we define $\mathcal{K}_t$ by

$$(\mathcal{K}_t(v_t))(x) = \int_{D_t} \kappa^{(t)}(x, y, a(\Pi^D_{t+1}(x)), a(\Pi^D_t(y))) v_t(y) \, \mathrm{d}\nu_t(y) \qquad \forall x \in D_{t+1}. \qquad (8)$$

where $\Pi^D_t : D_t \to D$ are fixed mappings. We have found numerically that, for certain PDE problems, the form (8) outperforms (7) due to the strong dependence of the solution u on the parameters a, for example, the Darcy flow problem considered in subsection 7.2.1. Indeed, if we think of (6) as a discrete time dynamical system, then the input $a \in \mathcal{A}$ only enters through the initial condition hence its influence diminishes with more layers. By directly building in a-dependence into the kernel, we ensure that it influences the entire architecture.

Lastly, let $\kappa^{(t)} \in C(D_{t+1} \times D_t \times \mathbb{R}^{d_{v_t}} \times \mathbb{R}^{d_{v_t}}; \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}})$. Then we define $\mathcal{K}_t$ by

$$(\mathcal{K}_t(v_t))(x) = \int_{D_t} \kappa^{(t)}(x, y, v_t(\Pi_t(x)), v_t(y)) v_t(y) \, \mathrm{d}\nu_t(y) \qquad \forall x \in D_{t+1}. \qquad (9)$$

where $\Pi_t : D_{t+1} \to D_t$ are fixed mappings. Note that, in contrast to (7) and (8), the integral operator (9) is nonlinear since the kernel can depend on the input function v_t. With this definition and a particular choice of kernel κ_t and measure ν_t, we show in Section 5.2 that neural operators are a continuous input/output space generalization of the popular transformer architecture (Vaswani et al., 2017).

Single Hidden Layer Construction Having defined possible choices for the integral kernel operator, we are now in a position to explicitly write down a full layer of the architecture defined by (6). For simplicity, we choose the integral kernel operator given by (7), but note that the other definitions (8), (9) work analogously. We then have that a single hidden layer update is given by

$$v_{t+1}(x) = \sigma_{t+1}\left(W_t v_t(\Pi_t(x)) + \int_{D_t} \kappa^{(t)}(x, y) v_t(y) \, \mathrm{d}\nu_t(y) + b_t(x)\right) \qquad \forall x \in D_{t+1} \qquad (10)$$

where $\Pi_t : D_{t+1} \to D_t$ are fixed mappings. We remark that, since we often consider functions on the same domain, we usually take Π_t to be the identity.

We will now give an example of a full single hidden layer architecture i.e. when $T = 2$. We choose $D_1 = D$, take σ_2 as the identity, and denote σ_1 by σ, assuming it is any activation function. Furthermore, for simplicity, we set $W_1 = 0$, $b_1 = 0$, and assume that $\nu_0 = \nu_1$ is the Lebesgue measure on $\mathbb{R}^d$. Then (6) becomes

$$(\mathcal{G}_\theta(a))(x) = \mathcal{Q}\left(\int_D \kappa^{(1)}(x, y) \sigma\left(W_0 \mathcal{P}(a(y)) + \int_D \kappa^{(0)}(y, z) \mathcal{P}(a(z)) \, \mathrm{d}z + b_0(y)\right) \mathrm{d}y\right) \qquad (11)$$

for any $x \in D'$. In this example, $\mathcal{P} \in C(\mathbb{R}^{d_a}; \mathbb{R}^{d_{v_0}})$, $\kappa^{(0)} \in C(D \times D; \mathbb{R}^{d_{v_1} \times d_{v_0}})$, $b_0 \in C(D; \mathbb{R}^{d_{v_1}})$, $W_0 \in \mathbb{R}^{d_{v_1} \times d_{v_0}}$, $\kappa^{(1)} \in C(D' \times D; \mathbb{R}^{d_{v_2} \times d_{v_1}})$, and $\mathcal{Q} \in C(\mathbb{R}^{d_{v_2}}; \mathbb{R}^{d_u})$. One can then parametrize the continuous functions $\mathcal{P}, \mathcal{Q}, \kappa^{(0)}, \kappa^{(1)}, b_0$ by standard feed-forward neural networks (or by any other means) and the matrix W_0 simply by its entries. The parameter vector $\theta \in \mathbb{R}^p$ then becomes the concatenation of the parameters of $\mathcal{P}, \mathcal{Q}, \kappa^{(0)}, \kappa^{(1)}, b_0$ along with the entries of W_0. One can then optimize these parameters by minimizing with respect to θ using standard gradient based minimization techniques. To implement this minimization, the functions entering the loss need to be discretized; but the learned parameters may then be used with other discretizations. In Section 4, we discuss various choices for parametrizing the kernels, picking the integration measure, and how those choices affect the computational complexity of the architecture.

Preprocessing It is often beneficial to manually include features into the input functions a to help facilitate the learning process. For example, instead of considering the $\mathbb{R}^{d_a}$-valued vector field a as input, we use the $\mathbb{R}^{d+d_a}$-valued vector field $(x, a(x))$. By including the identity element, information about the geometry of the spatial domain D is directly incorporated into the architecture. This allows the neural networks direct access to information that is already known in the problem and therefore eases learning. We use this idea in all of our numerical experiments in Section 7. Similarly, when learning a smoothing operator, it may be beneficial to include a smoothed version of the inputs a_ϵ using, for example, Gaussian convolution. Derivative information may also be of interest and thus, as input, one may consider, for example, the $\mathbb{R}^{d+2d_a+dd_a}$-valued vector field $(x, a(x), a_\epsilon(x), \nabla_x a_\epsilon(x))$. Many other possibilities may be considered on a problem-specific basis.

Discretization Invariance and Approximation In light of discretization invariance Theorem 8 and universal approximation Theorems 11 12, 13, 14 whose formal statements are given in Section 9, we may obtain a decomposition of the total error made by a neural operator as a sum of the discretization error and the approximation error. In particular, given a finite dimensional instantiation of a neural operator $\hat{\mathcal{G}}_\theta : \mathbb{R}^{Ld} \times \mathbb{R}^{Ld_a} \to \mathcal{U}$, for some L-point discretization of the input, we have

$$\|\hat{\mathcal{G}}_\theta(D_L, a|_{D_L}) - \mathcal{G}^\dagger(a)\|_{\mathcal{U}} \leq \underbrace{\|\hat{\mathcal{G}}_\theta(D_L, a|_{D_L}) - \mathcal{G}_\theta(a)\|_{\mathcal{U}}}_{\text{discretization error}} + \underbrace{\|\mathcal{G}_\theta(a) - \mathcal{G}^\dagger(a)\|_{\mathcal{U}}}_{\text{approximation error}}.$$

Our approximation theoretic Theorems imply that we can find parameters θ so that the approximation error is arbitrarily small while the discretization invariance Theorem states that we can find a fine enough discretization (large enough L) so that the discretization error is arbitrarily small. Therefore, with a fixed set of parameters independent of the input discretization, a neural operator that is able to be implemented on a computer can approximate operators to arbitrary accuracy.

4. Parameterization and Computation

In this section, we discuss various ways of parameterizing the infinite dimensional architecture (6), Figure 2. The goal is to find an intrinsic infinite dimensional parameterization that achieves small error (say ϵ), and then rely on numerical approximation to ensure that this parameterization delivers an error of the same magnitude (say 2ϵ), for all data discretizations fine enough. In this way the number of parameters used to achieve $\mathcal{O}(\epsilon)$ error is independent of the data discretization. In many applications we have in mind the data discretization is something we can control, for example when generating input/output pairs from solution of partial differential equations via numerical simulation. The proposed approach allows us to train a neural operator approximation using data from different discretizations, and to predict with discretizations different from those used in the data, all by relating everything to the underlying infinite dimensional problem.

We also discuss the computational complexity of the proposed parameterizations and suggest algorithms which yield efficient numerical methods for approximation. Subsections 4.1-4.4 delineate each of the proposed methods.

To simplify notation, we will only consider a single layer of (6) i.e. (10) and choose the input and output domains to be the same. Furthermore, we will drop the layer index t and write the single layer update as

$$u(x) = \sigma \left(Wv(x) + \int_D \kappa(x, y)v(y) \, \mathrm{d}\nu(y) + b(x) \right) \qquad \forall x \in D \tag{12}$$

where $D \subset \mathbb{R}^d$ is a bounded domain, $v : D \to \mathbb{R}^n$ is the input function and $u : D \to \mathbb{R}^m$ is the output function. When the domain domains D of v and u are different, we will usually extend them to be on a larger domain. We will consider σ to be fixed, and, for the time being, take $d\nu(y) = dy$ to be the Lebesgue measure on $\mathbb{R}^d$. Equation (12) then leaves three objects which can be parameterized: W, κ, and b. Since W is linear and acts only locally on v, we will always parametrize it by the values of its associated $m \times n$ matrix; hence $W \in \mathbb{R}^{m \times n}$ yielding mn parameters. We have found empirically that letting $b : D \to \mathbb{R}^m$ be a constant function over any domain D works at least as well as allowing it to be an arbitrary neural network. Perusal of the proof of Theorem 11 shows that we do not lose any approximation power by doing this, and we reduce the total number of parameters in the architecutre. Therefore we will always parametrize b by the entries of a fixed m-dimensional vector; in particular, $b \in \mathbb{R}^m$ yielding m parameters. Notice that both parameterizations are independent of any discretization of v.

The rest of this section will be dedicated to choosing the kernel function $\kappa : D \times D \to \mathbb{R}^{m \times n}$ and the computation of the associated integral kernel operator. For clarity of exposition, we consider only the simplest proposed version of this operator (7) but note that similar ideas may also be applied to (8) and (9). Furthermore, in order to focus on learning the kernel κ, here we drop σ, W, and b from (12) and simply consider the linear update

$$u(x) = \int_D \kappa(x, y)v(y)\, d\nu(y) \qquad \forall x \in D. \tag{13}$$

To demonstrate the computational challenges associated with (13), let $\{x_1, \ldots, x_J\} \subset D$ be a uniformly-sampled J-point discretization of D. Recall that we assumed $d\nu(y) = dy$ and, for simplicity, suppose that $\nu(D) = 1$, then the Monte Carlo approximation of (13) is

$$u(x_j) = \frac{1}{J} \sum_{l=1}^{J} \kappa(x_j, x_l)v(x_l), \qquad j = 1, \ldots, J \tag{14}$$

The integral in (13) can be approximated using any other integral approximation methods, including the celebrated Riemann sum for which $u(x_j) = \sum_{l=1}^{J} \kappa(x_j, x_l)v(x_l)\Delta x_l$ and Δx_l is the Riemann sum coefficient associated with ν at x_l. For the approximation methods, to compute u on the entire grid requires $\mathcal{O}(J^2)$ matrix-vector multiplications. Each of these matrix-vector multiplications requires $\mathcal{O}(mn)$ operations; for the rest of the discussion, we treat $mn = \mathcal{O}(1)$ as constant and consider only the cost with respect to J the discretization parameter since m and n are fixed by the architecture choice whereas J varies depending on required discretization accuracy and hence may be arbitrarily large. This cost is not specific to the Monte Carlo approximation but is generic for quadrature rules which use the entirety of the data. Therefore, when J is large, computing (13) becomes intractable and new ideas are needed in order to alleviate this. Subsections 4.1-4.4 propose different approaches to the solution to this problem, inspired by classical methods in numerical analysis. We finally remark that, in contrast, computations with W, b, and σ only require $\mathcal{O}(J)$ operations which justifies our focus on computation with the kernel integral operator.

Kernel Matrix. It will often times be useful to consider the kernel matrix associated to κ for the discrete points $\{x_1, \ldots, x_J\} \subset D$. We define the kernel matrix $K \in \mathbb{R}^{mJ \times nJ}$ to be the $J \times J$ block matrix with each block given by the value of the kernel i.e.

$$K_{jl} = \kappa(x_j, x_l) \in \mathbb{R}^{m \times n}, \qquad j, l = 1, \ldots, J$$

13

where we use (j, l) to index an individual block rather than a matrix element. Various numerical algorithms for the efficient computation of (13) can be derived based on assumptions made about the structure of this matrix, for example, bounds on its rank or sparsity.

4.1 Graph Neural Operator (GNO)

We first outline the Graph Neural Operator (GNO) which approximates (13) by combining a Nyström approximation with domain truncation and is implemented with the standard framework of graph neural networks. This construction was originally proposed in Li et al. (2020c).

Nyström Approximation. A simple yet effective method to alleviate the cost of computing (13) is employing a Nyström approximation. This amounts to sampling uniformly at random the points over which we compute the output function u. In particular, let $x_{k_1}, \ldots, x_{k_{J'}} \subset \{x_1, \ldots, x_J\}$ be $J' \ll J$ randomly selected points and, assuming $\nu(D) = 1$, approximate (13) by

$$u(x_{k_j}) \approx \frac{1}{J'} \sum_{l=1}^{J'} \kappa(x_{k_j}, x_{k_l}) v(x_{k_l}), \qquad j = 1, \ldots, J'.$$

We can view this as a low-rank approximation to the kernel matrix K, in particular,

$$K \approx K_{JJ'} K_{J'J'} K_{J'J} \tag{15}$$

where $K_{J'J'}$ is a $J' \times J'$ block matrix and $K_{JJ'}$, $K_{J'J}$ are interpolation matrices, for example, linearly extending the function to the whole domain from the random nodal points. The complexity of this computation is $\mathcal{O}(J'^2)$ hence it remains quadratic but only in the number of subsampled points J' which we assume is much less than the number of points J in the original discretization.

Truncation. Another simple method to alleviate the cost of computing (13) is to truncate the integral to a sub-domain of D which depends on the point of evaluation $x \in D$. Let $s : D \to \mathcal{B}(D)$ be a mapping of the points of D to the Lebesgue measurable subsets of D denoted $\mathcal{B}(D)$. Define $\mathrm{d}\nu(x, y) = \mathbb{1}_{s(x)} \mathrm{d}y$ then (13) becomes

$$u(x) = \int_{s(x)} \kappa(x, y) v(y) \, \mathrm{d}y \qquad \forall x \in D. \tag{16}$$

If the size of each set $s(x)$ is smaller than D then the cost of computing (16) is $\mathcal{O}(c_s J^2)$ where $c_s < 1$ is a constant depending on s. While the cost remains quadratic in J, the constant c_s can have a significant effect in practical computations, as we demonstrate in Section 7. For simplicity and ease of implementation, we only consider $s(x) = B(x, r) \cap D$ where $B(x, r) = \{y \in \mathbb{R}^d : \|y - x\|_{\mathbb{R}^d} < r\}$ for some fixed $r > 0$. With this choice of s and assuming that $D = [0, 1]^d$, we can explicitly calculate that $c_s \approx r^d$.

Furthermore notice that we do not lose any expressive power when we make this approximation so long as we combine it with composition. To see this, consider the example of the previous paragraph where, if we let $r = \sqrt{2}$, then (16) reverts to (13). Pick $r < 1$ and let $L \in \mathbb{N}$ with $L \geq 2$ be the smallest integer such that $2^{L-1}r \geq 1$. Suppose that $u(x)$ is computed by composing the right hand side of (16) L times with a different kernel every time. The domain of influence of $u(x)$ is then $B(x, 2^{L-1}r) \cap D = D$ hence it is easy to see that there exist L kernels such that

computing this composition is equivalent to computing (13) for any given kernel with appropriate regularity. Furthermore the cost of this computation is $\mathcal{O}(Lr^d J^2)$ and therefore the truncation is beneficial if $r^d(\log_2 1/r + 1) < 1$ which holds for any $r < 1/2$ when $d = 1$ and any $r < 1$ when $d \geq 2$. Therefore we have shown that we can always reduce the cost of computing (13) by truncation and composition. From the perspective of the kernel matrix, truncation enforces a sparse, block diagonally-dominant structure at each layer. We further explore the hierarchical nature of this computation using the multipole method in subsection 4.3.

Besides being a useful computational tool, truncation can also be interpreted as explicitly building local structure into the kernel κ. For problems where such structure exists, explicitly enforcing it makes learning more efficient, usually requiring less data to achieve the same generalization error. Many physical systems such as interacting particles in an electric potential exhibit strong local behavior that quickly decays, making truncation a natural approximation technique.

Graph Neural Networks.　We utilize the standard architecture of message passing graph networks employing edge features as introduced in Gilmer et al. (2017) to efficiently implement (13) on arbitrary discretizations of the domain D. To do so, we treat a discretization $\{x_1, \ldots, x_J\} \subset D$ as the nodes of a weighted, directed graph and assign edges to each node using the function $s : D \to \mathcal{B}(D)$ which, recall from the section on truncation, assigns to each point a domain of integration. In particular, for $j = 1, \ldots, J$, we assign the node x_j the value $v(x_j)$ and emanate from it edges to the nodes $s(x_j) \cap \{x_1, \ldots, x_J\} = \mathcal{N}(x_j)$ which we call the neighborhood of x_j. If $s(x) = D$ then the graph is fully-connected. Generally, the sparsity structure of the graph determines the sparsity of the kernel matrix K, indeed, the adjacency matrix of the graph and the block kernel matrix have the same zero entries. The weights of each edge are assigned as the arguments of the kernel. In particular, for the case of (13), the weight of the edge between nodes x_j and x_k is simply the concatenation $(x_j, x_k) \in \mathbb{R}^{2d}$. More complicated weighting functions are considered for the implementation of the integral kernel operators (8) or (9).

With the above definition the message passing algorithm of Gilmer et al. (2017), with averaging aggregation, updates the value $v(x_j)$ of the node x_j to the value $u(x_j)$ as

$$u(x_j) = \frac{1}{|\mathcal{N}(x_j)|} \sum_{y \in \mathcal{N}(x_j)} \kappa(x_j, y) v(y), \qquad j = 1, \ldots, J$$

which corresponds to the Monte-Carlo approximation of the integral (16). More sophisticated quadrature rules and adaptive meshes can also be implemented using the general framework of message passing on graphs, see, for example, Pfaff et al. (2020). We further utilize this framework in subsection 4.3.

Convolutional Neural Networks.　Lastly, we compare and contrast the GNO framework to standard convolutional neural networks (CNNs). In computer vision, the success of CNNs has largely been attributed to their ability to capture local features such as edges that can be used to distinguish different objects in a natural image. This property is obtained by enforcing the convolution kernel to have local support, an idea similar to our truncation approximation. Furthermore by directly using a translation invariant kernel, a CNN architecture becomes translation equivariant; this is a desirable feature for many vision models e.g. ones that perform segmentation. We will show that similar ideas can be applied to the neural operator framework to obtain an architecture with built-in local properties and translational symmetries that, unlike CNNs, remain consistent in function space.

To that end, let $\kappa(x, y) = \kappa(x - y)$ and suppose that $\kappa : \mathbb{R}^d \to \mathbb{R}^{m \times n}$ is supported on $B(0, r)$. Let $r^* > 0$ be the smallest radius such that $D \subseteq B(x^*, r^*)$ where $x^* \in \mathbb{R}^d$ denotes the center of mass of D and suppose $r \ll r^*$. Then (13) becomes the convolution

$$u(x) = (\kappa * v)(x) = \int_{B(x,r) \cap D} \kappa(x - y)v(y) \, \mathrm{d}y \qquad \forall x \in D. \tag{17}$$

Notice that (17) is precisely (16) when $s(x) = B(x, r) \cap D$ and $\kappa(x, y) = \kappa(x - y)$. When the kernel is parameterized by e.g. a standard neural network and the radius r is chosen independently of the data discretization, (17) becomes a layer of a convolution neural network that is consistent in function space. Indeed the parameters of (17) do not depend on any discretization of v. The choice $\kappa(x, y) = \kappa(x - y)$ enforces translational equivariance in the output while picking r small enforces locality in the kernel; hence we obtain the distinguishing features of a CNN model.

We will now show that, by picking a parameterization that is *inconsistent* in function space and applying a Monte Carlo approximation to the integral, (17) becomes a standard CNN. This is most easily demonstrated when $D = [0, 1]$ and the discretization $\{x_1, \ldots, x_J\}$ is equispaced i.e. $|x_{j+1} - x_j| = h$ for any $j = 1, \ldots, J - 1$. Let $k \in \mathbb{N}$ be an odd filter size and let $z_1, \ldots, z_k \in \mathbb{R}$ be the points $z_j = (j - 1 - (k - 1)/2)h$ for $j = 1, \ldots, k$. It is easy to see that $\{z_1, \ldots, z_k\} \subset \bar{B}(0, (k - 1)h/2)$ which we choose as the support of κ. Furthermore, we parameterize κ directly by its pointwise values which are $m \times n$ matrices at the locations $z_1, \ldots, z_k$ thus yielding kmn parameters. Then (17) becomes

$$u(x_j)_p \approx \frac{1}{k} \sum_{l=1}^{k} \sum_{q=1}^{n} \kappa(z_l)_{pq} v(x_j - z_l)_q, \qquad j = 1, \ldots, J, \ p = 1, \ldots, m$$

where we define $v(x) = 0$ if $x \notin \{x_1, \ldots, x_J\}$. Up to the constant factor $1/k$ which can be re-absorbed into the parameterization of κ, this is precisely the update of a stride 1 CNN with n input channels, m output channels, and zero-padding so that the input and output signals have the same length. This example can easily be generalized to higher dimensions and different CNN structures, we made the current choices for simplicity of exposition. Notice that if we double the amount of discretization points for v i.e. $J \mapsto 2J$ and $h \mapsto h/2$, the support of κ becomes $\bar{B}(0, (k - 1)h/4)$ hence the model changes due to the discretization of the data. Indeed, if we take the limit to the continuum $J \to \infty$, we find $\bar{B}(0, (k - 1)h/2) \to \{0\}$ hence the model becomes completely local. To fix this, we may try to increase the filter size k (or equivalently add more layers) simultaneously with J, but then the number of parameters in the model goes to infinity as $J \to \infty$ since, as we previously noted, there are kmn parameters in this layer. Therefore standard CNNs are not consistent models in function space. We demonstrate their inability to generalize to different resolutions in Section 7.

4.2 Low-rank Neural Operator (LNO)

By directly imposing that the kernel κ is of a tensor product form, we obtain a layer with $\mathcal{O}(J)$ computational complexity. We term this construction the Low-rank Neural Operator (LNO) due to its equivalence to directly parameterizing a finite-rank operator. We start by assuming $\kappa : D \times D \to \mathbb{R}$ is scalar valued and later generalize to the vector valued setting. We express the kernel as

$$\kappa(x, y) = \sum_{j=1}^{r} \varphi^{(j)}(x)\psi^{(j)}(y) \qquad \forall x, y \in D$$

for some functions $\varphi^{(1)}, \psi^{(1)}, \ldots, \varphi^{(r)}, \psi^{(r)} : D \to \mathbb{R}$ that are normally given as the components of two neural networks $\varphi, \psi : D \to \mathbb{R}^r$ or a single neural network $\Xi : D \to \mathbb{R}^{2r}$ which couples all functions through its parameters. With this definition, and supposing that $n = m = 1$, we have that (13) becomes

$$
\begin{aligned}
u(x) &= \int_D \sum_{j=1}^r \varphi^{(j)}(x)\psi^{(j)}(y)v(y) \, \mathrm{d}y \\
&= \sum_{j=1}^r \int_D \psi^{(j)}(y)v(y) \, \mathrm{d}y \, \varphi^{(j)}(x) \\
&= \sum_{j=1}^r \langle \psi^{(j)}, v \rangle \varphi^{(j)}(x)
\end{aligned}
$$

where $\langle \cdot, \cdot \rangle$ denotes the $L^2(D; \mathbb{R})$ inner product. Notice that the inner products can be evaluated independently of the evaluation point $x \in D$ hence the computational complexity of this method is $\mathcal{O}(rJ)$ which is linear in the discretization.

We may also interpret this choice of kernel as directly parameterizing a rank $r \in \mathbb{N}$ operator on $L^2(D; \mathbb{R})$. Indeed, we have

$$
u = \sum_{j=1}^r (\varphi^{(j)} \otimes \psi^{(j)})v \tag{18}
$$

which corresponds preceisely to applying the SVD of a rank r operator to the function v. Equation (18) makes natural the vector valued generalization. Assume $m, n \geq 1$ and $\varphi^{(j)} : D \to \mathbb{R}^m$ and $\psi^{(j)} : D \to \mathbb{R}^n$ for $j = 1, \ldots, r$ then, (18) defines an operator mapping $L^2(D; \mathbb{R}^n) \to L^2(D; \mathbb{R}^m)$ that can be evaluated as

$$
u(x) = \sum_{j=1}^r \langle \psi^{(j)}, v \rangle_{L^2(D; \mathbb{R}^n)} \varphi^{(j)}(x) \qquad \forall x \in D.
$$

We again note the linear computational complexity of this parameterization. Finally, we observe that this method can be interpreted as directly imposing a rank r structure on the kernel matrix. Indeed,

$$
K = K_{Jr}K_{rJ}
$$

where K_{Jr}, K_{rJ} are $J \times r$ and $r \times J$ block matricies respectively. This construction is similar to the DeepONet construction of Lu et al. (2019) discussed in Section 5.1, but parameterized to be consistent in function space.

4.3 Multipole Graph Neural Operator (MGNO)

A natural extension to directly working with kernels in a tensor product form as in Section 4.2 is to instead consider kernels that can be well approximated by such a form. This assumption gives rise to the fast multipole method (FMM) which employs a multi-scale decomposition of the kernel in order to achieve linear complexity in computing (13); for a detailed discussion see e.g. (E, 2011, Section 3.2). FMM can be viewed as a systematic approach to combine the sparse and low-rank approximations to the kernel matrix. Indeed, the kernel matrix is decomposed into different ranges

and a hierarchy of low-rank structures is imposed on the long-range components. We employ this idea to construct hierarchical, multi-scale graphs, without being constrained to particular forms of the kernel. We will elucidate the workings of the FMM through matrix factorization. This approach was first outlined in Li et al. (2020b) and is referred as the Multipole Graph Neural Operator (MGNO).

The key to the fast multipole method's linear complexity lies in the subdivision of the kernel matrix according to the range of interaction, as shown in Figure 3:

$$K = K_1 + K_2 + \ldots + K_L, \tag{19}$$

where K_ℓ with $\ell = 1$ corresponds to the shortest-range interaction, and $\ell = L$ corresponds to the longest-range interaction; more generally index ℓ is ordered by the range of interaction. While the uniform grids depicted in Figure 3 produce an orthogonal decomposition of the kernel, the decomposition may be generalized to arbitrary discretizations by allowing slight overlap of the ranges.
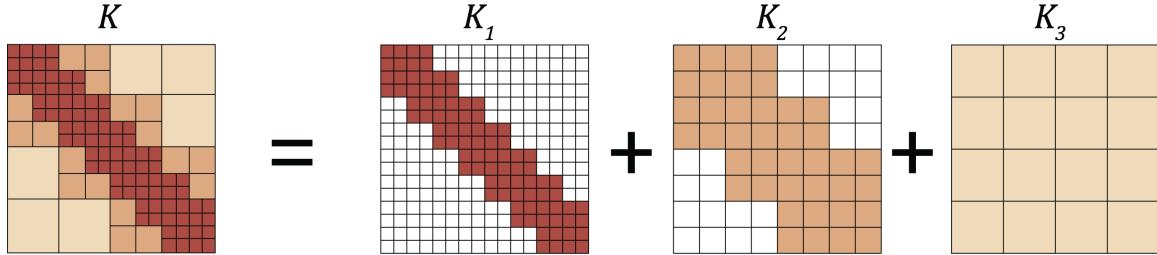

Figure 3: Hierarchical matrix decomposition
The kernel matrix K is decomposed with respect to its interaction ranges. K_1 corresponds to short-range interaction; it is sparse but full-rank. K_3 corresponds to long-range interaction; it is dense but low-rank.

Multi-scale Discretization. We produce a hierarchy of L discretizations with a decreasing number of nodes $J_1 \geq \ldots \geq J_L$ and increasing kernel integration radius $r_1 \leq \ldots \leq r_L$. Therefore, the shortest-range interaction K_1 has a fine resolution but is truncated locally, while the longest-range interaction K_L has a coarse resolution, but covers the entire domain. This is shown pictorially in Figure 3. The number of nodes $J_1 \geq \ldots \geq J_L$, and the integration radii $r_1 \leq \ldots \leq r_L$ are hyperparameter choices and can be picked so that the total computational complexity is linear in J.

A special case of this construction is when the grid is uniform. Then our formulation reduces to the standard fast multipole algorithm and the kernels K_l form an orthogonal decomposition of the full kernel matrix K. Assuming the underlying discretization $\{x_1, \ldots, x_J\} \subset D$ is a uniform grid with resolution s such that $s^d = J$, the L multi-level discretizations will be grids with resolution $s_l = s/2^{l-1}$, and consequentially $J_l = s_l^d = (s/2^{l-1})^d$. In this case r_l can be chosen as $1/s$ for $l = 1, \ldots, L$. To ensure orthogonality of the discretizations, the fast multipole algorithm sets the integration domains to be $B(x, r_l) \setminus B(x, r_{l-1})$ for each level $l = 2, \ldots, L$, so that the discretization on level l does not overlap with the one on level $l - 1$. Details of this algorithm can be found in e.g. Greengard and Rokhlin (1997).

Recursive Low-rank Decomposition. The coarse discretization representation can be understood as recursively applying an inducing points approximation (Quiñonero Candela and Rasmussen,
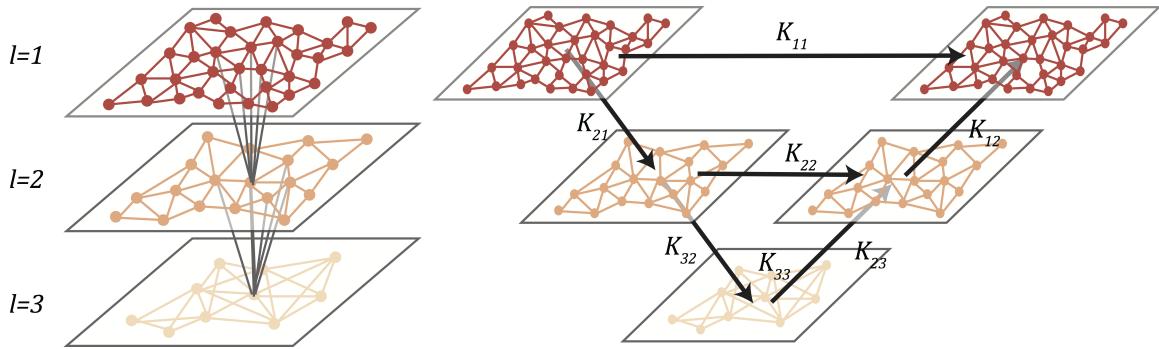
Figure 4: V-cycle

Left: the multi-level discretization. **Right**: one V-cycle iteration for the multipole neural operator.

2005): starting from a discretization with $J_1 = J$ nodes, we impose inducing points of size $J_2, J_3, \ldots, J_L$ which all admit a low-rank kernel matrix decomposition of the form (15). The original $J \times J$ kernel matrix K_l is represented by a much smaller $J_l \times J_l$ kernel matrix, denoted by $K_{l,l}$. As shown in Figure 3, K_1 is full-rank but very sparse while K_L is dense but low-rank. Such structure can be achieved by applying equation (15) recursively to equation (19), leading to the multi-resolution matrix factorization (Kondor et al., 2014):

$$K \approx K_{1,1} + K_{1,2}K_{2,2}K_{2,1} + K_{1,2}K_{2,3}K_{3,3}K_{3,2}K_{2,1} + \cdots \tag{20}$$

where $K_{1,1} = K_1$ represents the shortest range, $K_{1,2}K_{2,2}K_{2,1} \approx K_2$, represents the second shortest range, etc. The center matrix $K_{l,l}$ is a $J_l \times J_l$ kernel matrix corresponding to the l-level of the discretization described above. The matrices $K_{l+1,l}, K_{l,l+1}$ are $J_{l+1} \times J_l$ and $J_l \times J_{l+1}$ wide and long respectively block transition matrices. Denote $v_l \in R^{J_l \times n}$ for the representation of the input v at each level of the discretization for $l = 1, \ldots, L$, and $u_l \in R^{J_l \times n}$ for the output (assuming the inputs and outputs has the same dimension). We define the matrices $K_{l+1,l}, K_{l,l+1}$ as moving the representation v_l between different levels of the discretization via an integral kernel that we learn. Combining with the truncation idea introduced in subsection 4.1, we define the transition matrices as discretizations of the following integral kernel operators:

$$K_{l,l} : v_l \mapsto u_l = \int_{B(x,r_{l,l})} \kappa_{l,l}(x,y)v_l(y) \, \mathrm{d}y \tag{21}$$

$$K_{l+1,l} : v_l \mapsto u_{l+1} = \int_{B(x,r_{l+1,l})} \kappa_{l+1,l}(x,y)v_l(y) \, \mathrm{d}y \tag{22}$$

$$K_{l,l+1} : v_{l+1} \mapsto u_l = \int_{B(x,r_{l,l+1})} \kappa_{l,l+1}(x,y)v_{l+1}(y) \, \mathrm{d}y \tag{23}$$

where each kernel $\kappa_{l,l'} : D \times D \to \mathbb{R}^{n \times n}$ is parameterized as a neural network and learned.

V-cycle Algorithm   We present a V-cycle algorithm, see Figure 4, for efficiently computing (20). It consists of two steps: the *downward pass* and the *upward pass*. Denote the representation in downward pass and upward pass by $\check{v}$ and $\hat{v}$ respectively. In the downward step, the algorithm starts from the fine discretization representation $\check{v}_1$ and updates it by applying a downward transition

$\check{v}_{l+1} = K_{l+1,l}\check{v}_l$. In the upward step, the algorithm starts from the coarse presentation $\hat{v}_L$ and updates it by applying an upward transition and the center kernel matrix $\hat{v}_l = K_{l,l-1}\hat{v}_{l-1} + K_{l,l}\check{v}_l$. Notice that applying one level downward and upward exactly computes $K_{1,1} + K_{1,2}K_{2,2}K_{2,1}$, and a full L-level V-cycle leads to the multi-resolution decomposition (20).

Employing (21)-(23), we use L neural networks $\kappa_{1,1}, \ldots, \kappa_{L,L}$ to approximate the kernel operators associated to $K_{l,l}$, and $2(L-1)$ neural networks $\kappa_{1,2}, \kappa_{2,1}, \ldots$ to approximate the transitions $K_{l+1,l}, K_{l,l+1}$. Following the iterative architecture (6), we introduce the linear operator $W \in \mathbb{R}^{n \times n}$ (denoting it by W_l for each corresponding resolution) to help regularize the iteration, as well as the nonlinear activation function σ to increase the expensiveness. Since W acts pointwise (requiring J remains the same for input and output), we employ it only along with the kernel $K_{l,l}$ and not the transitions. At each layer $t = 0, \ldots, T-1$, we perform a full V-cycle as:

- Downward Pass

$$\text{For } l = 1, \ldots, L: \qquad \check{v}_{l+1}^{(t+1)} = \sigma(\hat{v}_{l+1}^{(t)} + K_{l+1,l}\check{v}_l^{(t+1)}) \qquad (24)$$

- Upward Pass

$$\text{For } l = L, \ldots, 1: \qquad \hat{v}_l^{(t+1)} = \sigma((W_l + K_{l,l})\check{v}_l^{(t+1)} + K_{l,l-1}\hat{v}_{l-1}^{(t+1)}). \qquad (25)$$

Notice that one full pass of the V-cycle algorithm defines a mapping $v \mapsto u$.

Multi-level Graphs. We emphasize that we view the discretization $\{x_1, \ldots, x_J\} \subset D$ as a graph in order to facilitate an efficient implementation through the message passing graph neural network architecture. Since the V-cycle algorithm works at different levels of the discretization, we build multi-level graphs to represent the coarser and finer discretizations. We present and utilize two constructions of multi-level graphs, the orthogonal multipole graph and the generalized random graph. The orthogonal multipole graph is the standard grid construction used in the fast multiple method which is adapted to a uniform grid, see e.g. (Greengard and Rokhlin, 1997). In this construction, the decomposition in (19) is orthogonal in that the finest graph only captures the closest range interaction, the second finest graph captures the second closest interaction minus the part already captured in the previous graph and so on, recursively. In particular, the ranges of interaction for each kernel do not overlap. While this construction is usually efficient, it is limited to uniform grids which may be a bottleneck for certain applications. Our second construction is the generalized random graph as shown in Figure 3 where the ranges of the kernels are allowed to overlap. The generalized random graph is very flexible as it can be applied on any domain geometry and discretization. Further it can also be combined with random sampling methods to work on problems where J is very large or combined with an active learning method to adaptively choose the regions where a finer discretization is needed.

Linear Complexity. Each term in the decomposition (19) is represented by the kernel matrix $K_{l,l}$ for $l = 1, \ldots, L$, and $K_{l+1,l}, K_{l,l+1}$ for $l = 1, \ldots, L-1$ corresponding to the appropriate sub-discretization. Therefore the complexity of the multipole method is $\sum_{l=1}^{L} \mathcal{O}(J_l^2 r_l^d) + \sum_{l=1}^{L-1} \mathcal{O}(J_l J_{l+1} r_l^d) = \sum_{l=1}^{L} \mathcal{O}(J_l^2 r_l^d)$. By designing the sub-discretization so that $\mathcal{O}(J_l^2 r_l^d) \leq \mathcal{O}(J)$, we can obtain complexity linear in J. For example, when $d = 2$, pick $r_l = 1/\sqrt{J_l}$ and $J_l = \mathcal{O}(2^{-l}J)$ such that r_L is large enough so that there exists a ball of radius r_L containing D. Then clearly $\sum_{l=1}^{L} \mathcal{O}(J_l^2 r_l^d) = \mathcal{O}(J)$. By combining with a Nyström approximation, we can obtain $\mathcal{O}(J')$ complexity for some $J' \ll J$.

4.4 Fourier Neural Operator (FNO)

Instead of working with a kernel directly on the domain D, we may consider its representation in Fourier space and directly parameterize it there. This allows us to utilize Fast Fourier Transform (FFT) methods in order to compute the action of the kernel integral operator (13) with almost linear complexity. A similar idea was used in (Nelsen and Stuart, 2021) to construct random features in function space The method we outline was first described in Li et al. (2020a) and is termed the Fourier Neural Operator (FNO). We note that the theory of Section 4 is designed for general kernels and does not apply to the FNO formulation; however, similar universal approximation results were developed for it in (Kovachki et al., 2021) when the input and output spaces are Hilbert space. For simplicity, we will assume that $D = \mathbb{T}^d$ is the unit torus and all functions are complex-valued. Let $\mathcal{F} : L^2(D; \mathbb{C}^n) \to \ell^2(\mathbb{Z}^d; \mathbb{C}^n)$ denote the Fourier transform of a function $v : D \to \mathbb{C}^n$ and $\mathcal{F}^{-1}$ its inverse. For $v \in L^2(D; \mathbb{C}^n)$ and $w \in \ell^2(\mathbb{Z}^d; \mathbb{C}^n)$, we have

$$(\mathcal{F}v)_j(k) = \langle v_j, \psi_k \rangle_{L^2(D;\mathbb{C})}, \qquad j \in \{1, \ldots, n\}, \quad k \in \mathbb{Z}^d,$$
$$(\mathcal{F}^{-1}w)_j(x) = \sum_{k \in \mathbb{Z}^d} w_j(k)\psi_k(x), \qquad j \in \{1, \ldots, n\}, \quad x \in D$$

where, for each $k \in \mathbb{Z}^d$, we define

$$\psi_k(x) = e^{2\pi i k_1 x_1} \cdots e^{2\pi i k_d x_d}, \qquad x \in D$$

with $i = \sqrt{-1}$ the imaginary unit. By letting $\kappa(x, y) = \kappa(x - y)$ for some $\kappa : D \to \mathbb{C}^{m \times n}$ in (13) and applying the convolution theorem, we find that

$$u(x) = \mathcal{F}^{-1}\big(\mathcal{F}(\kappa) \cdot \mathcal{F}(v)\big)(x) \qquad \forall x \in D.$$

We therefore propose to directly parameterize κ by its Fourier coefficients. We write

$$u(x) = \mathcal{F}^{-1}\big(R_\phi \cdot \mathcal{F}(v)\big)(x) \qquad \forall x \in D \tag{26}$$

where R_ϕ is the Fourier transform of a periodic function $\kappa : D \to \mathbb{C}^{m \times n}$ parameterized by some $\phi \in \mathbb{R}^p$.

For frequency mode $k \in \mathbb{Z}^d$, we have $(\mathcal{F}v)(k) \in \mathbb{C}^n$ and $R_\phi(k) \in \mathbb{C}^{m \times n}$. We pick a finite-dimensional parameterization by truncating the Fourier series at a maximal number of modes $k_{\max} = |Z_{k_{\max}}| = |\{k \in \mathbb{Z}^d : |k_j| \leq k_{\max,j}, \text{ for } j = 1, \ldots, d\}|$. This choice improves the empirical performance and sensitivity of the resulting model with respect to the choices of discretization. We thus parameterize R_ϕ directly as complex-valued $(k_{\max} \times m \times n)$-tensor comprising a collection of truncated Fourier modes and therefore drop ϕ from our notation. In the case where we have real-valued v and we want u to also be real-valued, we impose that κ is real-valued by enforcing conjugate symmetry in the parameterization i.e.

$$R(-k)_{j,l} = R^*(k)_{j,l} \qquad \forall k \in Z_{k_{\max}}, \quad j = 1, \ldots, m, \;\; l = 1, \ldots, n.$$

We note that the set $Z_{k_{\max}}$ is not the canonical choice for the low frequency modes of v_t. Indeed, the low frequency modes are usually defined by placing an upper-bound on the ℓ_1-norm of $k \in \mathbb{Z}^d$. We choose $Z_{k_{\max}}$ as above since it allows for an efficient implementation. Figure 5 gives a pictorial representation of an entire Neural Operator architecture employing Fourier layers.
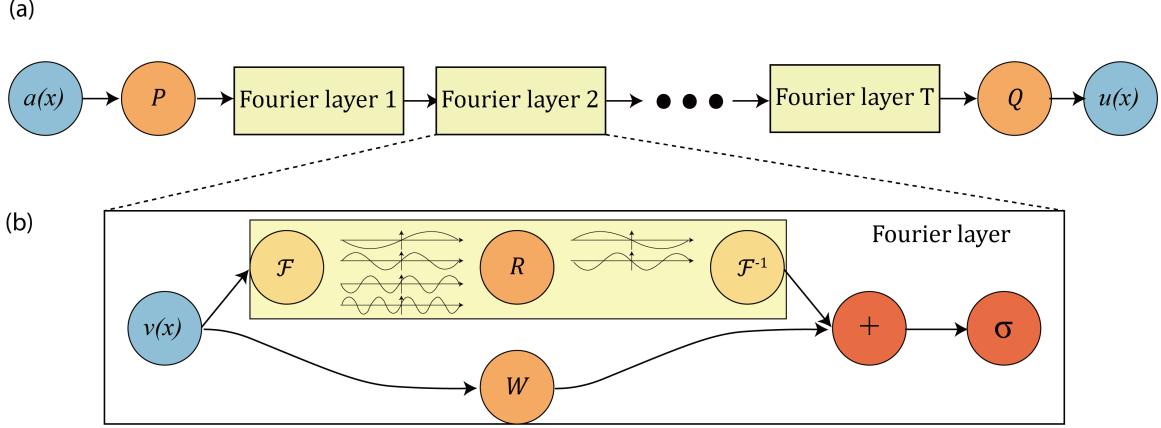
21

(a)

(b)

Figure 5: **top:** The architecture of the neural operators; **bottom:** Fourier layer.
(a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network $\mathcal{P}$. 2. Apply T (typically $T = 4$) layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. **(b) Fourier layers**: Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes which also filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

The Discrete Case and the FFT. Assuming the domain D is discretized with $J \in \mathbb{N}$ points, we can treat $v \in \mathbb{C}^{J \times n}$ and $\mathcal{F}(v) \in \mathbb{C}^{J \times n}$. Since we convolve v with a function which only has $k_{\max}$ Fourier modes, we may simply truncate the higher modes to obtain $\mathcal{F}(v) \in \mathbb{C}^{k_{\max} \times n}$. Multiplication by the weight tensor $R \in \mathbb{C}^{k_{\max} \times m \times n}$ is then

$$\left(R \cdot (\mathcal{F}v_t)\right)_{k,l} = \sum_{j=1}^{n} R_{k,l,j} (\mathcal{F}v)_{k,j}, \qquad k = 1, \ldots, k_{\max}, \quad l = 1, \ldots, m. \qquad (27)$$

When the discretization is uniform with resolution $s_1 \times \cdots \times s_d = J$, $\mathcal{F}$ can be replaced by the Fast Fourier Transform. For $v \in \mathbb{C}^{J \times n}$, $k = (k_1, \ldots, k_d) \in \mathbb{Z}_{s_1} \times \cdots \times \mathbb{Z}_{s_d}$, and $x = (x_1, \ldots, x_d) \in D$, the FFT $\hat{\mathcal{F}}$ and its inverse $\hat{\mathcal{F}}^{-1}$ are defined as

$$(\hat{\mathcal{F}}v)_l(k) = \sum_{x_1=0}^{s_1-1} \cdots \sum_{x_d=0}^{s_d-1} v_l(x_1, \ldots, x_d) e^{-2i\pi \sum_{j=1}^{d} \frac{x_j k_j}{s_j}},$$

$$(\hat{\mathcal{F}}^{-1}v)_l(x) = \sum_{k_1=0}^{s_1-1} \cdots \sum_{k_d=0}^{s_d-1} v_l(k_1, \ldots, k_d) e^{2i\pi \sum_{j=1}^{d} \frac{x_j k_j}{s_j}}$$

for $l = 1, \ldots, n$. In this case, the set of truncated modes becomes

$$Z_{k_{\max}} = \{(k_1, \ldots, k_d) \in \mathbb{Z}_{s_1} \times \cdots \times \mathbb{Z}_{s_d} \mid k_j \le k_{\max,j} \text{ or } s_j - k_j \le k_{\max,j}, \text{ for } j = 1, \ldots, d\}.$$

When implemented, R is treated as a $(s_1 \times \cdots \times s_d \times m \times n)$-tensor and the above definition of $Z_{k_{\max}}$ corresponds to the "corners" of R, which allows for a straight-forward parallel implementation of (27) via matrix-vector multiplication. In practice, we have found the choice $k_{\max,j}$ roughly around $\frac{1}{3}$ to $\frac{2}{3}$ of the maximum number of Fourier modes in the Fast Fourier Transform of the grid valuation of the input function provides desirable performance. In our empirical studies, we set $k_{\max,j} = 12$ which yields $k_{\max} = 12^d$ parameters per channel, to be sufficient for all the tasks that we consider.

Choices for R. In general, R can be defined to depend on $(\mathcal{F}a)$, the Fourier transform of the input $a \in \mathcal{A}$ to parallel our construction (8). Indeed, we can define $R_\phi : \mathbb{Z}^d \times \mathbb{C}^{d_a} \to \mathbb{C}^{m \times n}$ as a parametric function that maps $\big(k, (\mathcal{F}a)(k)\big)$ to the values of the appropriate Fourier modes. We have experimented with the following parameterizations of R_ϕ:

- *Direct.* Define the parameters $\phi_k \in \mathbb{C}^{m \times n}$ for each wave number k:

$$R_\phi\big(k, (\mathcal{F}a)(k)\big) := \phi_k.$$

- *Linear.* Define the parameters $\phi_{k_1} \in \mathbb{C}^{m \times n \times d_a}$, $\phi_{k_2} \in \mathbb{C}^{m \times n}$ for each wave number k:

$$R_\phi\big(k, (\mathcal{F}a)(k)\big) := \phi_{k_1}(\mathcal{F}a)(k) + \phi_{k_2}.$$

- *Feed-forward neural network.* Let $\Phi_\phi : \mathbb{Z}^d \times \mathbb{C}^{d_a} \to \mathbb{C}^{m \times n}$ be a neural network with parameters ϕ:

$$R_\phi\big(k, (\mathcal{F}a)(k)\big) := \Phi_\phi(k, (\mathcal{F}a)(k)).$$

We find that the *linear* parameterization has a similar performance to the *direct* parameterization above, however, it is not as efficient both in terms of computational complexity and the number of parameters required. On the other hand, we find that the *feed-forward neural network* parameterization has a worse performance. This is likely due to the discrete structure of the space $\mathbb{Z}^d$; numerical evidence suggests neural networks are not adept at handling this structure. Our experiments in this work focus on the direct parameterization presented above.

Invariance to Discretization. The Fourier layers are discretization-invariant because they can learn from and evaluate functions which are discretized in an arbitrary way. Since parameters are learned directly in Fourier space, resolving the functions in physical space simply amounts to projecting on the basis elements $e^{2\pi i \langle x, k \rangle}$; these are well-defined everywhere on $\mathbb{C}^d$.

Quasi-linear Complexity. The weight tensor R contains $k_{\max} < J$ modes, so the inner multiplication has complexity $\mathcal{O}(k_{\max})$. Therefore, the majority of the computational cost lies in computing the Fourier transform $\mathcal{F}(v)$ and its inverse. General Fourier transforms have complexity $\mathcal{O}(J^2)$, however, since we truncate the series the complexity is in fact $\mathcal{O}(J k_{\max})$, while the FFT has complexity $\mathcal{O}(J \log J)$. Generally, we have found using FFTs to be very efficient, however, a uniform discretization is required.

Non-uniform and Non-periodic Geometry. The Fourier neural operator model is defined based on Fourier transform operations accompanied by local residual operations and potentially additive bias function terms. These operations are mainly defined on general geometries, function spaces, and choices of discretization. They are not limited to rectangular domains, periodic functions, or uniform grids. In this paper, we instantiate these operations on uniform grids and periodic functions in order to develop fast implementations that enjoy spectral convergence and utilize methods such as fast Fourier transform. In order to maintain a fast and memory-efficient method, our implementation of the Fourier neural operator relies on the fast Fourier transform which is only defined on uniform mesh discretizations of $D = \mathbb{T}^d$, or for functions on the square satisfying homogeneous Dirichlet (fast Fourier sine transform) or homogeneous Neumann (fast Fourier cosine transform) boundary conditions. However, the fast implementation of Fourier neural operator can be applied in more

general geometries via Fourier continuations. Given any compact manifold $D = \mathcal{M}$, we can always embed it into a periodic cube (torus),

$$i : \mathcal{M} \to \mathbb{T}^d$$

where the regular FFT can be applied. Conventionally, in numerical analysis applications, the embedding i is defined through a continuous extension by fitting polynomials (Bruno et al., 2007). However, in the Fourier neural operator, the idea can be applied simply by padding the input with zeros. The loss is computed only on the original space during training. The Fourier neural operator will automatically generate a smooth extension to the padded domain in the output space.

4.5 Summary

We summarize the main computational approaches presented in this section and their complexity:

- **GNO**: Subsample J' points from the J-point discretization and compute the truncated integral

$$u(x) = \int_{B(x,r)} \kappa(x,y)v(y) \, \mathrm{d}y \tag{28}$$

  at a $\mathcal{O}(JJ')$ complexity.

- **LNO**: Decompose the kernel function tensor product form and compute

$$u(x) = \sum_{j=1}^{r} \langle \psi^{(j)}, v \rangle \varphi^{(j)}(x) \tag{29}$$

  at a $\mathcal{O}(J)$ complexity.

- **MGNO**: Compute a multi-scale decomposition of the kernel

$$\begin{aligned} K &= K_{1,1} + K_{1,2}K_{2,2}K_{2,1} + K_{1,2}K_{2,3}K_{3,3}K_{3,2}K_{2,1} + \cdots \\ u(x) &= (Kv)(x) \end{aligned} \tag{30}$$

  at a $\mathcal{O}(J)$ complexity.

- **FNO**: Parameterize the kernel in the Fourier domain and compute the using the FFT

$$u(x) = \mathcal{F}^{-1}\big(R_\phi \cdot \mathcal{F}(v)\big)(x) \tag{31}$$

  at a $\mathcal{O}(J \log J)$ complexity.

5. Neural Operators and Other Deep Learning Models

In this section, we provide a discussion on the recent related methods, in particular, DeepONets, and demonstrate that their architectures are subsumed by generic neural operators when neural operators are parametrized inconsistently 5.1. When only applied and queried on fixed grids, we show neural operator architectures subsume neural networks and, furthermore, we show how transformers are special cases of neural operators 5.2.

5.1 DeepONets

We will now draw a parallel between the recently proposed DeepONet architecture in Lu et al. (2019), a map from finite-dimensional spaces to function spaces, and the neural operator framework. We will show that if we use a particular, point-wise parameterization of the first kernel in a NO and discretize the integral operator, we obtain a DeepONet. However, such a parameterization breaks the notion of discretization invariance because the number of parameters depends on the discretization of the input function. Therefore such a model cannot be applied to arbitrarily discretized functions and its number of parameters goes to infinity as we take the limit to the continuum. This phenomenon is similar to our discussion in subsection 4.1 where a NO parametrization which is inconsistent in function space and breaks discretization invariance yields a CNN. We propose a modification to the DeepONet architecture, based on the idea of the LNO, which addresses this issue and gives a discretization invariant neural operator.

Proposition 5 *A neural operator with a point-wise parameterized first kernel and discretized integral operators yields a DeepONet.*

Proof We work with (11) where we choose $W_0 = 0$ and denote b_0 by b. For simplicity, we will consider only real-valued functions i.e. $d_a = d_u = 1$ and set $d_{v_0} = d_{v_1} = n$ and $d_{v_2} = p$ for some $n, p \in \mathbb{N}$. Define $\mathcal{P} : \mathbb{R} \to \mathbb{R}^n$ by $\mathcal{P}(x) = (x, \dots, x)$ and $\mathcal{Q} : \mathbb{R}^p \to \mathbb{R}$ by $\mathcal{Q}(x) = x_1 + \dots + x_p$. Furthermore let $\kappa^{(1)} : D' \times D \to \mathbb{R}^{p \times n}$ be defined by some $\kappa^{(1)}_{jk} : D' \times D \to \mathbb{R}$ for $j = 1, \dots, p$ and $k = 1, \dots, n$. Similarly let $\kappa^{(0)} : D \times D \to \mathbb{R}^{n \times n}$ be given as $\kappa^{(0)}(x, y) = \text{diag}(\kappa^{(0)}_1(x, y), \dots, \kappa^{(0)}_n(x, y))$ for some $\kappa^{(0)}_1, \dots \kappa^{(0)}_n : D \times D \to \mathbb{R}$. Then (11) becomes

$$(\mathcal{G}_\theta(a))(x) = \sum_{k=1}^{p} \sum_{j=1}^{n} \int_D \kappa^{(1)}_{jk}(x, y) \sigma \left(\int_D \kappa^{(0)}_j(y, z) a(z) \, \mathrm{d}z + b_j(y) \right) \mathrm{d}y$$

where $b(y) = (b_1(y), \dots, b_n(y))$ for some $b_1, \dots, b_n : D \to \mathbb{R}$. Let $x_1, \dots, x_q \in D$ be the points at which the input function a is evaluated and denote by $\tilde{a} = (a(x_1), \dots, a(x_q)) \in \mathbb{R}^q$ the vector of evaluations. Choose $\kappa^{(0)}_j(y, z) = \mathbb{1}(y) w_j(z)$ for some $w_1, \dots, w_n : D \to \mathbb{R}$ where $\mathbb{1}$ denotes the constant function taking the value one. Let

$$w_j(x_l) = \frac{q}{|D|} \tilde{w}_{jl}$$

for $j = 1, \dots, n$ and $l = 1, \dots, q$ where $\tilde{w}_{jl} \in \mathbb{R}$ are some constants. Furthermore let $b_j(y) = \tilde{b}_j \mathbb{1}(y)$ for some constants $\tilde{b}_j \in \mathbb{R}$. Then the Monte Carlo approximation of the inner-integral yields

$$(\mathcal{G}_\theta(a))(x) = \sum_{k=1}^{p} \sum_{j=1}^{n} \int_D \kappa^{(1)}_{jk}(x, y) \sigma \left(\langle \tilde{w}_j, \tilde{a} \rangle_{\mathbb{R}^q} + \tilde{b}_j \right) \mathbb{1}(y) \, \mathrm{d}y$$

where $\tilde{w}_j = (\tilde{w}_{j1}, \dots, \tilde{w}_{jq})$. Choose $\kappa^{(1)}_{jk}(x, y) = (\tilde{c}_{jk}/|D|) \varphi_k(x) \mathbb{1}(y)$ for some constants $\tilde{c}_{jk} \in \mathbb{R}$ and functions $\varphi_1, \dots, \varphi_p : D' \to \mathbb{R}$. Then we obtain

$$(\mathcal{G}_\theta(a))(x) = \sum_{k=1}^{p} \left(\sum_{j=1}^{n} \tilde{c}_{jk} \sigma \left(\langle \tilde{w}_j, \tilde{a} \rangle_{\mathbb{R}^q} + \tilde{b}_j \right) \right) \varphi_k(x) = \sum_{k=1}^{p} G_k(\tilde{v}) \varphi_k(x) \qquad (32)$$

where $G_k : \mathbb{R}^q \to \mathbb{R}$ can be viewed as the components of a single hidden layer neural network $G : \mathbb{R}^q \to \mathbb{R}^p$ with parameters $\tilde{w}_{jl}, \tilde{b}_j, \tilde{c}_{jk}$. The set of maps $\varphi_1, \ldots, \varphi_p$ form the *trunk net* while G is the *branch net* of a DeepONet. Our construction above can clearly be generalized to yield arbitrary depth branch nets by adding more kernel integral layers, and, similarly, the trunk net can be chosen arbitrarily deep by parameterizing each φ_k as a deep neural network. ∎

Since the mappings $w_1, \ldots, w_n$ are point-wise parametrized based on the input values $\tilde{a}$, it is clear that the construction in the above proof is not discretization invariant. In order to make this model a discretization invariant neural operator, we propose DeepONet-Operator where, for each j, we replace the inner product in the finite dimensional space $\langle \tilde{w}_j, \tilde{a} \rangle_{\mathbb{R}^q}$ with an appropriate inner product in the function space $\langle w_j, a \rangle$.

$$(\mathcal{G}_\theta(a))(x) = \sum_{k=1}^p \left(\sum_{j=1}^n \tilde{c}_{jk} \sigma \left(\langle w_j, a \rangle + \tilde{b}_j \right) \right) \varphi_k(x) \tag{33}$$

This operation is a projection of function a onto w_j. Parametrizing w_j by neural networks makes DeepONet-Operator a discretization invariant model.

There are other ways in which the issue can be resolved for DeepONets. For example, by fixing the set of points on which the input function is evaluated independently of its discretization, by taking local spatial averages as in (Lanthaler et al., 2021) or more generally by taking a set of linear functionals on $\mathcal{A}$ as input to a finite-dimensional branch neural network (a generalization to DeeONet-Operator) as in the specific PCA-based variant on DeepONet in (De Hoop et al., 2022). We demonstrate numerically in Section 7 that, when applied in the standard way, the error incurred by DeepONet(s) grows with the discretization of a while it remains constant for neural operators.

Linear Approximation and Nonlinear Approximation. We point out that parametrizations of the form (32) fall within the class of *linear* approximation methods since the nonlinear space $\mathcal{G}^\dagger(\mathcal{A})$ is approximated by the linear space $\mathrm{span}\{\varphi_1, \ldots, \varphi_p\}$ (DeVore, 1998). The quality of the best possible linear approximation to a nonlinear space is given by the Kolmogorov n-width where n is the dimension of the linear space used in the approximation (Pinkus, 1985). The rate of decay of the n-width as a function of n quantifies how well the linear space approximates the nonlinear one. It is well know that for some problems such as the flow maps of advection-dominated PDEs, the n-widths decay very slowly; hence a very large n is needed for a good approximation for such problems (Cohen and DeVore, 2015). This can be limiting in practice as more parameters are needed in order to describe more basis functions φ_j and therefore more data is needed to fit these parameters.

On the other hand, we point out that parametrizations of the form (6), and the particular case (11), constitute (in general) a form of *nonlinear* approximation. The benefits of nonlinear approximation are well understood in the setting of function approximation, see e.g. (DeVore, 1998), however the theory for the operator setting is still in its infancy (Bonito et al., 2020; Cohen et al., 2020). We observe numerically in Section 7 that nonlinear parametrizations such as (11) outperform linear ones such as DeepONets or the low-rank method introduced in Section 4.2 when implemented with similar numbers of parameters. We acknowledge, however, that the theory presented in Section 9 is based on the reduction to a linear approximation and therefore does not capture the benefits of the nonlinear approximation. Furthermore, in practice, we have found that deeper architectures than

(11) (usually four to five layers are used in the experiments of Section 7), perform better. The benefits of depth are again not captured in our analysis in Section 9 either. We leave further theoretical studies of approximation properties as an interesting avenue of investigation for future work.

Function Representation.   An important difference between neural operators, introduced here, PCA-based operator approximation, introduced in Bhattacharya et al. (2020) and DeepONets, introduced in Lu et al. (2019), is the manner in which the output function space is finite-dimensionalized. Neural operators as implemented in this paper typically use the same finite-dimensionalization in both the input and output function spaces; however different variants of the neural operator idea use different finite-dimensionalizations. As discussed in Section 4, the GNO and MGNO are finite-dimensionalized using pointwise values as the nodes of graphs; the FNO is finite-dimensionalized in Fourier space, requiring finite-dimensionalization on a uniform grid in real space; the Low-rank neural operator is finite-dimensionalized on a product space formed from the Barron space of neural networks. The PCA approach finite-dimensionalizes in the span of PCA modes. DeepONet, on the other hand, uses different input and output space finite-dimensionalizations; in its basic form it uses pointwise (grid) values on the input (branch net) whilst its output (trunk net) is represented as a function in Barron space. There also exist POD-DeepONet variants that finite-dimensionalize the output in the span of PCA modes Lu et al. (2021b), bringing them closer to the method introduced in Bhattacharya et al. (2020), but with a different finite-dimensionalization of the input space.

As is widely quoted, "all models are wrong, but some are useful" Box (1976). For operator approximation, each finite-dimensionalization has its own induced biases and limitations, and therefore works best on a subset of problems. Finite-dimensionalization introduces a trade-off between flexibility and representation power of the resulting approximate architecture. The Barron space representation (Low-rank operator and DeepONet) is usually the most generic and flexible as it is widely applicable. However this can lead to induced biases and reduced representation power on specific problems; in practice, DeepONet sometimes needs problem-specific feature engineering and architecture choices as studied in Lu et al. (2021b). We conjecture that these problem-specific features compensate for the induced bias and reduced representation power that the basic form of the method (Lu et al., 2019) sometimes exhibits. The PCA (PCA operator, POD-DeepONet) and graph-based (GNO, MGNO) discretizations are also generic, but more specific compared to the DeepONet representation; for this reason POD-DeepONet can outperform DeepONet on some problems (Lu et al., 2021b). On the other hand, the uniform grid-based representation FNO is the most specific of all those operator approximators considered in this paper: in its basic form it applies by discretizing the input functions, assumed to be specified on a periodic domain, on a uniform grid. As shown in Section 7 FNO usually works out of the box on such problems. But, as a trade-off, it requires substantial additional treatments to work well on non-uniform geometries, such as extension, interpolation (explored in Lu et al. (2021b)), and Fourier continuation (Bruno et al., 2007).

5.2 Transformers as a Special Case of Neural Operators

We will now show that our neural operator framework can be viewed as a continuum generalization to the popular transformer architecture (Vaswani et al., 2017) which has been extremely successful in natural language processing tasks (Devlin et al., 2018; Brown et al., 2020) and, more recently, is becoming a popular choice in computer vision tasks (Dosovitskiy et al., 2020). The parallel stems from the fact that we can view sequences of arbitrary length as arbitrary discretizations of functions. Indeed, in the context of natural language processing, we may think of a sentence as

a "word"-valued function on, for example, the domain $[0, 1]$. Assuming our function is linked to a sentence with a fixed semantic meaning, adding or removing words from the sentence simply corresponds to refining or coarsening the discretization of $[0, 1]$. We will now make this intuition precise in the proof of the following statement.

Proposition 6 *The attention mechanism in transformer models is a special case of a neural operator layer.*

Proof We will show that by making a particular choice of the nonlinear integral kernel operator (9) and discretizing the integral by a Monte-Carlo approximation, a neural operator layer reduces to a pre-normalized, single-headed attention, transformer block as originally proposed in (Vaswani et al., 2017). For simplicity, we assume $d_{v_t} = n \in \mathbb{N}$ and that $D_t = D$ for any $t = 0, \ldots, T$, the bias term is zero, and $W = I$ is the identity. Furthermore, to simplify notation, we will drop the layer index t from (10) and, employing (9), obtain

$$u(x) = \sigma \left(v(x) + \int_D \kappa_v(x, y, v(x), v(y)) v(y) \, \mathrm{d}y \right) \qquad \forall x \in D \tag{34}$$

a single layer of the neural operator where $v : D \to \mathbb{R}^n$ is the input function to the layer and we denote by $u : D \to \mathbb{R}^n$ the output function. We use the notation κ_v to indicate that the kernel depends on the entirety of the function v as well as on its pointwise values $v(x)$ and $v(y)$. While this is not explicitly done in (9), it is a straightforward generalization. We now pick a specific form for kernel, in particular, we assume $\kappa_v : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{n \times n}$ does not explicitly depend on the spatial variables (x, y) but only on the input pair $(v(x), v(y))$. Furthermore, we let

$$\kappa_v(v(x), v(y)) = g_v(v(x), v(y)) R$$

where $R \in \mathbb{R}^{n \times n}$ is a matrix of free parameters i.e. its entries are concatenated in θ so they are learned, and $g_v : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is defined as

$$g_v(v(x), v(y)) = \left(\int_D \exp \left(\frac{\langle Av(s), Bv(y) \rangle}{\sqrt{m}} \right) \mathrm{d}s \right)^{-1} \exp \left(\frac{\langle Av(x), Bv(y) \rangle}{\sqrt{m}} \right).$$

Here $A, B \in \mathbb{R}^{m \times n}$ are again matrices of free parameters, $m \in \mathbb{N}$ is a hyperparameter, and $\langle \cdot, \cdot \rangle$ is the Euclidean inner-product on $\mathbb{R}^m$. Putting this together, we find that (34) becomes

$$u(x) = \sigma \left(v(x) + \int_D \frac{\exp \left(\frac{\langle Av(x), Bv(y) \rangle}{\sqrt{m}} \right)}{\int_D \exp \left(\frac{\langle Av(s), Bv(y) \rangle}{\sqrt{m}} \right) \mathrm{d}s} Rv(y) \, \mathrm{d}y \right) \qquad \forall x \in D. \tag{35}$$

Equation (35) can be thought of as the continuum limit of a transformer block. To see this, we will discretize to obtain the usual transformer block.

To that end, let $\{x_1, \ldots, x_k\} \subset D$ be a uniformly-sampled, k-point discretization of D and denote $v_j = v(x_j) \in \mathbb{R}^n$ and $u_j = u(x_j) \in \mathbb{R}^n$ for $j = 1, \ldots, k$. Approximating the inner-integral in (35) by Monte-Carlo, we have

$$\int_D \exp \left(\frac{\langle Av(s), Bv(y) \rangle}{\sqrt{m}} \right) \mathrm{d}s \approx \frac{|D|}{k} \sum_{l=1}^k \exp \left(\frac{\langle Av_l, Bv(y) \rangle}{\sqrt{m}} \right).$$

Plugging this into (35) and using the same approximation for the outer integral yields

$$u_j = \sigma \left(v_j + \sum_{q=1}^{k} \frac{\exp\left(\frac{\langle Av_j, Bv_q \rangle}{\sqrt{m}}\right)}{\sum_{l=1}^{k} \exp\left(\frac{\langle Av_l, Bv_q \rangle}{\sqrt{m}}\right)} Rv_q \right), \qquad j = 1, \ldots, k. \tag{36}$$

Equation (36) can be viewed as a Nyström approximation of (35). Define the vectors $z_q \in \mathbb{R}^k$ by

$$z_q = \frac{1}{\sqrt{m}} (\langle Av_1, Bv_q \rangle, \ldots, \langle Av_k, Bv_q \rangle), \qquad q = 1, \ldots, k.$$

Define $S : \mathbb{R}^k \to \Delta_k$, where Δ_k denotes the k-dimensional probability simplex, as the softmax function

$$S(w) = \left(\frac{\exp(w_1)}{\sum_{j=1}^{k} \exp(w_j)}, \ldots, \frac{\exp(w_k)}{\sum_{j=1}^{k} \exp(w_j)} \right), \qquad \forall w \in \mathbb{R}^k.$$

Then we may re-write (36) as

$$u_j = \sigma \left(v_j + \sum_{q=1}^{k} S_j(z_q) Rv_q \right), \qquad j = 1, \ldots, k.$$

Furthermore, if we re-parametrize $R = R^{\text{out}} R^{\text{val}}$ where $R^{\text{out}} \in \mathbb{R}^{n \times m}$ and $R^{\text{val}} \in \mathbb{R}^{m \times n}$ are matrices of free parameters, we obtain

$$u_j = \sigma \left(v_j + R^{\text{out}} \sum_{q=1}^{k} S_j(z_q) R^{\text{val}} v_q \right), \qquad j = 1, \ldots, k$$

which is precisely the single-headed attention, transformer block with no layer normalization applied inside the activation function. In the language of transformers, the matrices A, B, and R^{val} correspond to the *queries*, *keys*, and *values* functions respectively. We note that tricks such as layer normalization (Ba et al., 2016) can be adapted in a straightforward manner to the continuum setting and incorporated into (35). Furthermore multi-headed self-attention can be realized by simply allowing κ_v to be a sum over multiple functions with form $g_v R$ all of which have separate trainable parameters. Including such generalizations yields the continuum limit of the transformer as implemented in practice. We do not pursue this here as our goal is simply to draw a parallel between the two methods. ∎

Even though transformers are special cases of neural operators, the standard attention mechanism is memory and computation intensive, as seen in Section 6, compared to neural operator architectures developed here (7)-(9). The high computational complexity of transformers is evident is (35) since we must evaluate a *nested* integral of v for each $x \in D$. Recently, efficient attention mechanisms have been explored, e.g. long-short Zhu et al. (2021) and adaptive FNO-based attention mechanisms (Guibas et al., 2021). However, many of the efficient vision transformer architectures (Choromanski et al., 2020; Dosovitskiy et al., 2020) like ViTs are *not* special cases of neural operators since they use CNN layers to generate tokens, which are not discretization invariant.

6. Test Problems

A central application of neural operators is learning solution operators defined by parametric partial differential equations. In this section, we define four test problems for which we numerically study the approximation properties of neural operators. To that end, let $(\mathcal{A}, \mathcal{U}, \mathcal{F})$ be a triplet of Banach spaces. The first two problem classes considered are derived from the following general class of PDEs:

$$\mathsf{L}_a u = f \tag{37}$$

where, for every $a \in \mathcal{A}$, $\mathsf{L}_a : \mathcal{U} \to \mathcal{F}$ is a, possibly nonlinear, partial differential operator, and $u \in \mathcal{U}$ corresponds to the solution of the PDE (37) when $f \in \mathcal{F}$ and appropriate boundary conditions are imposed. The second class will be evolution equations with initial condition $a \in \mathcal{A}$ and solution $u(t) \in \mathcal{U}$ at every time $t > 0$. We seek to learn the map from a to $u := u(\tau)$ for some fixed time $\tau > 0$; we will also study maps on paths (time-dependent solutions).

Our goal will be to learn the mappings

$$\mathcal{G}^\dagger : a \mapsto u \qquad \text{or} \qquad \mathcal{G}^\dagger : f \mapsto u;$$

we will study both cases, depending on the test problem considered. We will define a probability measure μ on $\mathcal{A}$ or $\mathcal{F}$ which will serve to define a model for likely input data. Furthermore, measure μ will define a topology on the space of mappings in which $\mathcal{G}^\dagger$ lives, using the Bochner norm (3). We will assume that each of the spaces $(\mathcal{A}, \mathcal{U}, \mathcal{F})$ are Banach spaces of functions defined on a bounded domain $D \subset \mathbb{R}^d$. All reported errors will be Monte-Carlo estimates of the relative error

$$\mathbb{E}_{a \sim \mu} \frac{\|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|_{L^2(D)}}{\|\mathcal{G}^\dagger(a)\|_{L^2(D)}}$$

or equivalently replacing a with f in the above display and with the assumption that $\mathcal{U} \subseteq L^2(D)$. The domain D will be discretized, usually uniformly, with $J \in \mathbb{N}$ points.

6.1 Poisson Equation

First we consider the one-dimensional Poisson equation with a zero boundary condition. In particular, (37) takes the form

$$-\frac{d^2}{dx^2} u(x) = f(x), \qquad x \in (0,1)$$
$$u(0) = u(1) = 0 \tag{38}$$

for some source function $f : (0,1) \to \mathbb{R}$. In particular, for $D(\mathsf{L}) := H_0^1((0,1); \mathbb{R}) \cap H^2((0,1); \mathbb{R})$, we have $\mathsf{L} : D(\mathsf{L}) \to L^2((0,1); \mathbb{R})$ defined as $-d^2/dx^2$, noting that that L has no dependence on any parameter $a \in \mathcal{A}$ in this case. We will consider the weak form of (38) with source function $f \in H^{-1}((0,1); \mathbb{R})$ and therefore the solution operator $\mathcal{G}^\dagger : H^{-1}((0,1); \mathbb{R}) \to H_0^1((0,1); \mathbb{R})$ defined as

$$\mathcal{G}^\dagger : f \mapsto u.$$

We define the probability measure $\mu = N(0, C)$ where

$$C = \left(\mathsf{L} + I\right)^{-2},$$

defined through the spectral theory of self-adjoint operators. Since μ charges a subset of $L^2((0,1);\mathbb{R})$, we will learn $\mathcal{G}^\dagger : L^2((0,1);\mathbb{R}) \to H_0^1((0,1);\mathbb{R})$ in the topology induced by (3).

In this setting, $\mathcal{G}^\dagger$ has a closed-form solution given as

$$\mathcal{G}^\dagger(f) = \int_0^1 G(\cdot, y) f(y) \, \mathrm{d}y$$

where

$$G(x,y) = \frac{1}{2}\left(x + y - |y - x|\right) - xy, \qquad \forall (x,y) \in [0,1]^2$$

is the Green's function. Note that while $\mathcal{G}^\dagger$ is a linear operator, the Green's function G is non-linear as a function of its arguments. We will consider only a single layer of (6) with $\sigma_1 = \mathrm{Id}$, $\mathcal{P} = \mathrm{Id}$, $\mathcal{Q} = \mathrm{Id}$, $W_0 = 0$, $b_0 = 0$, and

$$\mathcal{K}_0(f) = \int_0^1 \kappa_\theta(\cdot, y) f(y) \, \mathrm{d}y$$

where $\kappa_\theta : \mathbb{R}^2 \to \mathbb{R}$ will be parameterized as a standard neural network with parameters θ.

The purpose of the current example is two-fold. First we will test the efficacy of the neural operator framework in a simple setting where an exact solution is analytically available. Second we will show that by building in the right inductive bias, in particular, paralleling the form of the Green's function solution, we obtain a model that generalizes outside the distribution μ. That is, once trained, the model will generalize to any $f \in L^2((0,1);\mathbb{R})$ that may be outside the support of μ. For example, as defined, the random variable $f \sim \mu$ is a continuous function, however, if κ_θ approximates the Green's function well then the model $\mathcal{G}^\dagger$ will approximate the solution to (38) accurately even for discontinuous inputs.

To create the dataset used for training, solutions to (38) are obtained by numerical integration using the Green's function on a uniform grid with 85 collocation points. We use $N = 1000$ training examples.

6.2 Darcy Flow

We consider the steady state of Darcy Flow in two dimensions which is the second order elliptic equation

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x), &\quad x \in D \\ u(x) &= 0, &\quad x \in \partial D \end{aligned} \tag{39}$$

where $D = (0,1)^2$ is the unit square. In this setting $\mathcal{A} = L^\infty(D;\mathbb{R}_+)$, $\mathcal{U} = H_0^1(D;\mathbb{R})$, and $\mathcal{F} = H^{-1}(D;\mathbb{R})$. We fix $f \equiv 1$ and consider the weak form of (39) and therefore the solution operator $\mathcal{G}^\dagger : L^\infty(D;\mathbb{R}^+) \to H_0^1(D;\mathbb{R})$ defined as

$$\mathcal{G}^\dagger : a \mapsto u. \tag{40}$$

Note that while (39) is a linear PDE, the solution operator $\mathcal{G}^\dagger$ is nonlinear. We define the probability measure $\mu = T_\sharp N(0, C)$ as the pushforward of a Gaussian measure under the operator T where the covariance of the Gaussian is

$$C = (-\Delta + 9I)^{-2}$$

31

with $D(-\Delta)$ defined to impose zero Neumann boundary on the Laplacian. We define T to be a Nemytskii operator acting on functions, defined through the map $T : \mathbb{R} \to \mathbb{R}_+$ defined as

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

The random variable $a \sim \mu$ is a piecewise-constant function with random interfaces given by the underlying Gaussian random field. Such constructions are prototypical models for many physical systems such as permeability in sub-surface flows and (in a vector generalization) material microstructures in elasticity.

To create the dataset used for training, solutions to (39) are obtained using a second-order finite difference scheme on a uniform grid of size 421×421. All other resolutions are downsampled from this data set. We use $N = 1000$ training examples.

6.3 Burgers' Equation

We consider the one-dimensional viscous Burgers' equation

$$\frac{\partial}{\partial t}u(x,t) + \frac{1}{2}\frac{\partial}{\partial x}\big(u(x,t)\big)^2 = \nu\frac{\partial^2}{\partial x^2}u(x,t), \qquad x \in (0, 2\pi), t \in (0, \infty) \tag{41}$$
$$u(x,0) = u_0(x), \qquad x \in (0, 2\pi)$$

with periodic boundary conditions and a fixed viscosity $\nu = 10^{-1}$. Let $\Psi : L^2_{\text{per}}((0, 2\pi); \mathbb{R}) \times \mathbb{R}_+ \to H^s_{\text{per}}((0, 2\pi); \mathbb{R})$, for any $s > 0$, be the flow map associated to (41), in particular,

$$\Psi(u_0, t) = u(\cdot, t), \qquad t > 0.$$

We consider the solution operator defined by evaluating Ψ at a fixed time. Fix any $s \geq 0$. Then we may define $\mathcal{G}^\dagger : L^2_{\text{per}}((0, 2\pi); \mathbb{R}) \to H^s_{\text{per}}((0, 2\pi); \mathbb{R})$ by

$$\mathcal{G}^\dagger : u_0 \mapsto \Psi(u_0, 1). \tag{42}$$

We define the probability measure $\mu = N(0, C)$ where

$$C = 625\Big(-\frac{d^2}{dx^2} + 25I\Big)^{-2}$$

with domain of the Laplacian defined to impose periodic boundary conditions. We chose the initial condition for (41) by drawing $u_0 \sim \mu$, noting that μ charges a subset of $L^2_{\text{per}}((0, 2\pi); \mathbb{R})$.

To create the dataset used for training, solutions to (41) are obtained using a pseudo-spectral split step method where the heat equation part is solved exactly in Fourier space and then the nonlinear part is advanced using a forward Euler method with a very small time step. We use a uniform spatial grid with $2^{13} = 8192$ collocation points and subsample all other resolutions from this data set. We use $N = 1000$ training examples.

6.4 Navier-Stokes Equation

We consider the two-dimensional Navier-Stokes equation for a viscous, incompressible fluid

$$\partial_t u(x,t) + u(x,t) \cdot \nabla u(x,t) + \nabla p(x,t) = \nu \Delta u(x,t) + f(x), \qquad x \in \mathbb{T}^2, t \in (0, \infty)$$
$$\nabla \cdot u(x,t) = 0, \qquad x \in \mathbb{T}^2, t \in [0, \infty) \qquad (43)$$
$$u(x,0) = u_0(x), \qquad x \in \mathbb{T}^2$$

where $\mathbb{T}^2$ is the unit torus i.e. $[0,1]^2$ equipped with periodic boundary conditions, and $\nu \in \mathbb{R}_+$ is a fixed viscosity. Here $u : \mathbb{T}^2 \times \mathbb{R}_+ \to \mathbb{R}^2$ is the velocity field, $p : \mathbb{T}^2 \times \mathbb{R}_+ \to \mathbb{R}^2$ is the pressure field, and $f : \mathbb{T}^2 \to \mathbb{R}$ is a fixed forcing function.

Equivalently, we study the vorticity-streamfunction formulation of the equation

$$\partial_t w(x,t) + \nabla^\perp \psi \cdot \nabla w(x,t) = \nu \Delta w(x,t) + g(x), \qquad x \in \mathbb{T}^2, t \in (0, \infty), \qquad (44\text{a})$$
$$-\Delta \psi = \omega, \qquad x \in \mathbb{T}^2, t \in (0, \infty), \qquad (44\text{b})$$
$$w(x,0) = w_0(x), \qquad x \in \mathbb{T}^2, \qquad (44\text{c})$$

where w is the out-of-plane component of the vorticity field $\nabla \times u : \mathbb{T}^2 \times \mathbb{R}_+ \to \mathbb{R}^3$. Since, when viewed in three dimensions, $u = \big(u_1(x_1, x_2), u_2(x_1, x_2), 0\big)$, it follows that $\nabla \times u = (0, 0, \omega)$. The stream function ψ is related to the velocity by $u = \nabla^\perp \psi$, enforcing the divergence-free condition. Similar considerations as for the curl of u apply to the curl of f, showing that $\nabla \times f = (0, 0, g)$. Note that in (44b) ψ is undetermined up to an (irrelevant, since u is computed from ψ by taking a gradient) additive constant; uniqueness is restored by seeking ψ with spatial mean 0. We define the forcing term as

$$g(x_1, x_2) = 0.1(\sin(2\pi(x_1 + x_2)) + \cos(2\pi(x_1 + x_2))), \qquad \forall (x_1, x_2) \in \mathbb{T}^2.$$

The corresponding Reynolds number is estimated as $Re = \frac{\sqrt{0.1}}{\nu(2\pi)^{3/2}}$ (Chandler and Kerswell, 2013). Let $\Psi : L^2(\mathbb{T}^2; \mathbb{R}) \times \mathbb{R}_+ \to H^s(\mathbb{T}^2; \mathbb{R})$, for any $s > 0$, be the flow map associated to (44), in particular,

$$\Psi(w_0, t) = w(\cdot, t), \qquad t > 0.$$

Notice that this is well-defined for any $w_0 \in L^2(\mathbb{T}; \mathbb{R})$.

We will define two notions of the solution operator. In the first, we will proceed as in the previous examples, in particular, $\mathcal{G}^\dagger : L^2(\mathbb{T}^2; \mathbb{R}) \to H^s(\mathbb{T}^2; \mathbb{R})$ is defined as

$$\mathcal{G}^\dagger : w_0 \mapsto \Psi(w_0, T) \qquad (45)$$

for some fixed $T > 0$. In the second, we will map an initial part of the trajectory to a later part of the trajectory. In particular, we define $\mathcal{G}^\dagger : L^2(\mathbb{T}^2; \mathbb{R}) \times C\big((0, 10]; H^s(\mathbb{T}^2; \mathbb{R})\big) \to C\big((10, T]; H^s(\mathbb{T}^2; \mathbb{R})\big)$ by

$$\mathcal{G}^\dagger : \big(w_0, \Psi(w_0, t)|_{t \in (0, 10]}\big) \mapsto \Psi(w_0, t)|_{t \in (10, T]} \qquad (46)$$

for some fixed $T > 10$. We define the probability measure $\mu = N(0, C)$ where

$$C = 7^{3/2}(-\Delta + 49I)^{-2.5}$$

33

with periodic boundary conditions on the Laplacian. We model the initial vorticity $w_0 \sim \mu$ to (44) as μ charges a subset of $L^2(\mathbb{T}^2; \mathbb{R})$. Its pushforward onto $\Psi(w_0, t)|_{t \in (0,10]}$ is required to define the measure on input space in the second case defined by (46).

To create the dataset used for training, solutions to (44) are obtained using a pseudo-spectral split step method where the viscous terms are advanced using a Crank–Nicolson update and the nonlinear and forcing terms are advanced using Heun's method. Dealiasing is used with the 2/3 rule. For further details on this approach see (Chandler and Kerswell, 2013). Data is obtained on a uniform 256×256 grid and all other resolutions are subsampled from this data set. We experiment with different viscosities ν, final times T, and amounts of training data N.

6.4.1 BAYESIAN INVERSE PROBLEM

As an application of operator learning, we consider the inverse problem of recovering the initial vorticity in the Navier-Stokes equation (44) from partial, noisy observations of the vorticity at a later time. Consider the first solution operator defined in subsection 6.4, in particular, $\mathcal{G}^\dagger : L^2(\mathbb{T}^2; \mathbb{R}) \to H^s(\mathbb{T}^2; \mathbb{R})$ defined as

$$\mathcal{G}^\dagger : w_0 \mapsto \Psi(w_0, 50)$$

where Ψ is the flow map associated to (44). We then consider the inverse problem

$$y = O\big(\mathcal{G}^\dagger(w_0)\big) + \eta \tag{47}$$

of recovering $w_0 \in L^2(\mathbb{T}^2; \mathbb{R})$ where $O : H^s(\mathbb{T}^2; \mathbb{R}) \to \mathbb{R}^{49}$ is the evaluation operator on a uniform 7×7 interior grid, and $\eta \sim N(0, \Gamma)$ is observational noise with covariance $\Gamma = (1/\gamma^2)I$ and $\gamma = 0.1$. We view (47) as the Bayesian inverse problem mapping prior measure μ on w_0 to posterior measure π^y on w_0/y. In particular, π^y has density with respect to μ, given by the Randon-Nikodym derivative

$$\frac{\mathrm{d}\pi^y}{\mathrm{d}\mu}(w_0) \propto \exp\Big(-\frac{1}{2}\|y - O\big(\mathcal{G}^\dagger(w_0)\big)\|_\Gamma^2\Big)$$

where $\|\cdot\|_\Gamma = \|\Gamma^{-1/2} \cdot\|$ and $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^{49}$. For further details on Bayesian inversion for functions see (Cotter et al., 2009; Stuart, 2010), and see (Cotter et al., 2013) for MCMC methods adapted to the function-space setting.

We solve (47) by computing the posterior mean $\mathbb{E}_{w_0 \sim \pi^y}[w_0]$ using the pre-conditioned Crank–Nicolson (pCN) MCMC method described in Cotter et al. (2013) for this task. We employ pCN in two cases: (i) using $\mathcal{G}^\dagger$ evaluated with the pseudo-spectral method described in section 6.4; and (ii) using $\mathcal{G}_\theta$, the neural operator approximating $\mathcal{G}^\dagger$. After a 5,000 sample burn-in period, we generate 25,000 samples from the posterior using both approaches and use them to compute the posterior mean.

6.4.2 SPECTRA

Because of the constant-in-time forcing term the energy reaches a non-zero equilibrium in time which is statistically reproducible for different initial conditions. To compare the complexity of the solution to the Navier-Stokes problem outlined in subsection 6.4 we show, in Figure 6, the Fourier spectrum of the solution data at time $t = 50$ for three different choices of the viscosity ν. The figure demonstrates that, for a wide range of wavenumbers k, which grows as ν decreases, the rate of decay of the spectrum is $-5/3$, matching what is expected in the turbulent regime (Kraichnan, 1967). This is a statistically stationary property of the equation, sustained for all positive times.
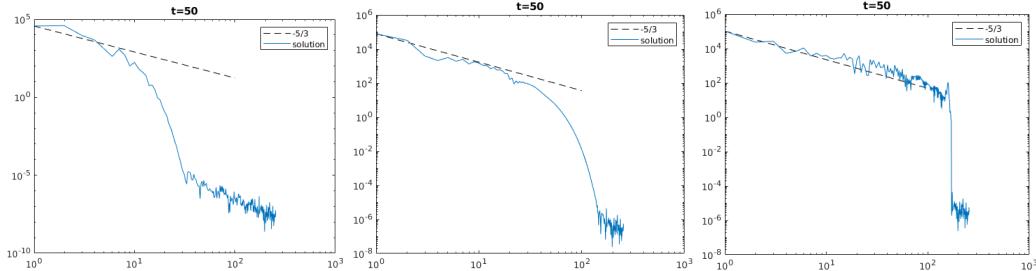
Figure 6: Spectral Decay of Navier-Stokes.
The spectral decay of the Navier-stokes equation data. The y-axis is represents the value of each mode; the x-axis is the wavenumber $|k| = k_1 + k_2$. From left to right, the solutions have viscosity $\nu = 10^{-3}$, 10^{-4}, 10^{-5} respectively.

6.5 Choice of Loss Criteria

In general, the model has the best performance when trained and tested using the same loss criteria. If one trains the model using one norm and tests with another norm, the model may overfit in the training norm. Furthermore, the choice of loss function plays a key role. In this work, we use the relative L_2 error to measure the performance in all our problems. Both the L_2 error and its square, the mean squared error (MSE), are common choices of the testing criteria in the numerical analysis and machine learning literature. We observed that using the relative error to train the model has a good normalization and regularization effect that prevents overfitting. In practice, training with the relative L_2 loss results in around half the testing error rate compared to training with the MSE loss.

7. Numerical Results

In this section, we compare the proposed neural operator with other supervised learning approaches, using the four test problems outlined in Section 6. In Subsection 7.1 we study the Poisson equation, and learning a Greens function; Subsection 7.2 considers the coefficient to solution map for steady Darcy flow, and the initial condition to solution at positive time map for Burgers equation. In subsection 7.3 we study the Navier-Stokes equation.

   We compare with a variety of architectures found by discretizing the data and applying finite-dimensional approaches, as well as with other operator-based approximation methods; further detailed comparison of other operator-based approximation methods may be found in De Hoop et al. (2022), where the issue of error versus cost (with cost defined in various ways such as evaluation time of the network, amount of data required) is studied. We do not compare against traditional solvers (FEM/FDM/Spectral), although our methods, once trained, enable evaluation of the input to output map orders of magnitude more quickly than by use of such traditional solvers on complex problems. We demonstrate the benefits of this speed-up in a prototypical application, Bayesian inversion, in Subsubection 7.3.4.

   All the computations are carried on a single Nvidia V100 GPU with 16GB memory. The code is available at `https://github.com/zongyi-li/graph-pde` and `https://github.com/zongyi-li/fourier_neural_operator`.

Setup of the Four Methods: We construct the neural operator by stacking four integral operator layers as specified in (6) with the ReLU activation. No batch normalization is needed. Unless

otherwise specified, we use $N = 1000$ training instances and 200 testing instances. We use the Adam optimizer to train for 500 epochs with an initial learning rate of 0.001 that is halved every 100 epochs. We set the channel dimensions $d_{v_0} = \cdots = d_{v_3} = 64$ for all one-dimensional problems and $d_{v_0} = \cdots = d_{v_3} = 32$ for all two-dimensional problems. The kernel networks $\kappa^{(0)}, \ldots, \kappa^{(3)}$ are standard feed-forward neural networks with three layers and widths of 256 units. We use the following abbreviations to denote the methods introduced in Section 4.

- **GNO:** The method introduced in subsection 4.1, truncating the integral to a ball with radius $r = 0.25$ and using the Nyström approximation with $J' = 300$ sub-sampled nodes.

- **LNO:** The low-rank method introduced in subsection 4.2 with rank $r = 4$.

- **MGNO:** The multipole method introduced in subsection 4.3. On the Darcy flow problem, we use the random construction with three graph levels, each sampling $J_1 = 400, J_2 = 100, J_3 = 25$ nodes nodes respectively. On the Burgers' equation problem, we use the orthogonal construction without sampling.

- **FNO:** The Fourier method introduced in subsection 4.4. We set $k_{\max,j} = 16$ for all one-dimensional problems and $k_{\max,j} = 12$ for all two-dimensional problems.

Remark on the Resolution. Traditional PDE solvers such as FEM and FDM approximate a single function and therefore their error to the continuum decreases as the resolution is increased. The figures we show here exhibit something different: the error is independent of resolution, once enough resolution is used, but is not zero. This reflects the fact that there is a residual approximation error, in the infinite dimensional limit, from the use of a finite-parametrized neural operator, trained on a finite amount of data. Invariance of the error with respect to (sufficiently fine) resolution is a desirable property that demonstrates that an intrinsic approximation of the operator has been learned, independent of any specific discretization; see Figure 8. Furthermore, resolution-invariant operators can do zero-shot super-resolution, as shown in Subsubection 7.3.1.

7.1 Poisson Equation

Recall the Poisson equation (38) introduced in subsection 6.1. We use a zero hidden layer neural operator construction without lifting the input dimension. In particular, we simply learn a kernel $\kappa_\theta : \mathbb{R}^2 \to \mathbb{R}$ parameterized as a standard feed-forward neural network with parameters θ. Using only $N = 1000$ training examples, we obtain a relative test error of 10^{-7}. The neural operator gives an almost perfect approximation to the true solution operator in the topology of (3).

To examine the quality of the approximation in the much stronger uniform topology, we check whether the kernel κ_θ approximates the Green's function for this problem. To see why this is enough, let $K \subset L^2([0,1];\mathbb{R})$ be a bounded set i.e.

$$\|f\|_{L^2([0,1];\mathbb{R})} \leq M, \qquad \forall f \in K$$

and suppose that

$$\sup_{(x,y)\in[0,1]^2} |\kappa_\theta(x,y) - G(x,y)| < \frac{\epsilon}{M}.$$

for some $\epsilon > 0$. Then it is easy to see that

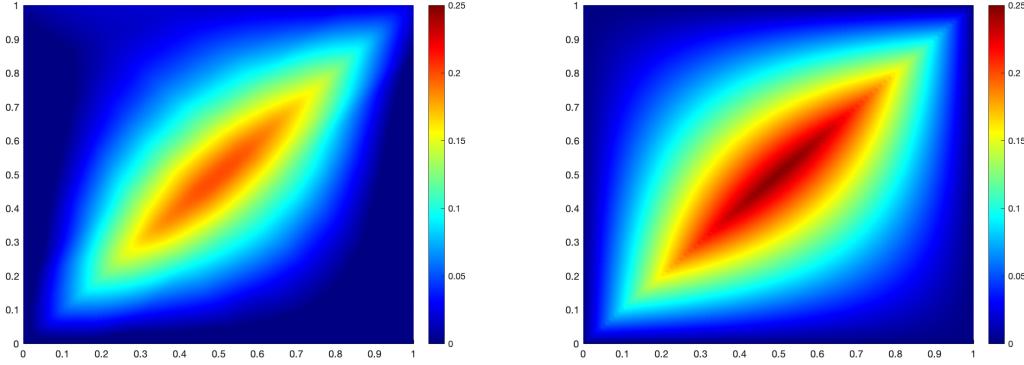$$\sup_{f\in K} \|\mathcal{G}^\dagger(f) - \mathcal{G}_\theta(f)\|_{L^2([0,1];\mathbb{R})} < \epsilon,$$

Figure 7: Kernel for one-dimensional Green's function, with the Nystrom approximation method
left: learned kernel function; **right:** the analytic Green's function.
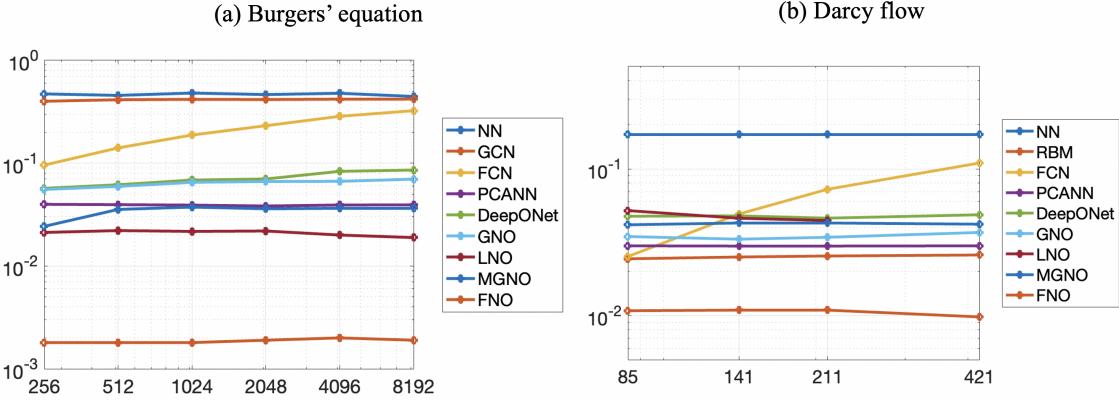This is a proof of concept of the graph kernel network on 1 dimensional Poisson equation and the
comparison of learned and truth kernel.

in particular, we obtain an approximation in the topology of uniform convergence over bounded sets, while having trained only in the topology of the Bochner norm (3). Figure 7 shows the results from which we can see that κ_θ does indeed approximate the Green's function well. This result implies that by constructing a suitable architecture, we can generalize to the entire space and data that is well outside the support of the training set.

7.2 Darcy and Burgers Equations

In the following section, we compare four methods presented in this paper, with different operator approximation benchmarks; we study the Darcy flow problem introduced in Subsection 6.2 and the Burgers' equation problem introduced in Subsection 6.3. The solution operators of interest are defined by (40) and (42). We use the following abbreviations for the methods against which we benchmark.

- **NN** is a standard point-wise feedforward neural network. It is mesh-free, but performs badly due to lack of neighbor information. We use standard fully connected neural networks with 8 layers and width 1000.

- **FCN** is the state of the art neural network method based on Fully Convolution Network (Zhu and Zabaras, 2018). It has a dominating performance for small grids $s = 61$. But fully convolution networks are mesh-dependent and therefore their error grows when moving to a larger grid.

- **PCA+NN** is an instantiation of the methodology proposed in Bhattacharya et al. (2020): using PCA as an autoencoder on both the input and output spaces and interpolating the latent spaces with a standard fully connected neural network with width 200. The method provably obtains mesh-independent error and can learn purely from data, however the solution can only be evaluated on the same mesh as the training data.

(a) benchmarks on Burgers equation; (b) benchmarks on Darcy Flow for different resolutions; Train and test on the same resolution. For acronyms, see Section 7; details in Tables 3, 2.

Figure 8: Benchmark on Burger's equation and Darcy Flow

- **RBM** is the classical Reduced Basis Method (using a PCA basis), which is widely used in applications and provably obtains mesh-independent error (DeVore, 2014). This method has good performance, but the solutions can only be evaluated on the same mesh as the training data and one needs knowledge of the PDE to employ it.

- **DeepONet** is the Deep Operator network (Lu et al., 2019) that comes equipped with an approximation theory (Lanthaler et al., 2021). We use the unstacked version with width 200 which is precisely defined in the original work (Lu et al., 2019). We use standard fully connected neural networks with 8 layers and width 200.

7.2.1 DARCY FLOW

The results of the experiments on Darcy flow are shown in Figure 8 and Table 2. All the methods, except for FCN, achieve invariance of the error with respect to the resolution s. In the experiment, we tune each model across of range of different widths and depth to obtain the choices used here; for DeepONet for example this leads to 8 layers and width 200 as reported above.

Within our hyperparameter search, the Fourier neural operator (FNO) obtains the lowest relative error. The Fourier based method likely sees this advantage because the output functions are smooth in these test problems. We also note that is also possible to obtain better results on each model using modified architectures and problem specific feature engineering. For example for DeepONet, using CNN on the branch net and PCA on the trunk net (the latter being similar to the method used in Bhattacharya et al. (2020)) can achieve 0.0232 relative L_2 error, as shown in Lu et al. (2021b), about half the size of the error we obtain here, but for a very coarse grid with $s = 29$. In the experiments the different approximation architectures are such their training cost are similar across all the methods considered, for given s. Noting this, and for example comparing the graph-based neural operator methods such as GNO and MGNO that use Nyström sampling in physical space with FNO, we see that FNO is more accurate.

Networks	$s = 85$	$s = 141$	$s = 211$	$s = 421$
NN	0.1716	0.1716	0.1716	0.1716
FCN	0.0253	0.0493	0.0727	0.1097
PCANN	0.0299	0.0298	0.0298	0.0299
RBM	0.0244	0.0251	0.0255	0.0259
DeepONet	0.0476	0.0479	0.0462	0.0487
GNO	0.0346	0.0332	0.0342	0.0369
LNO	0.0520	0.0461	0.0445	–
MGNO	0.0416	0.0428	0.0428	0.0420
FNO	**0.0108**	**0.0109**	**0.0109**	**0.0098**

Table 2: Relative error on 2-d Darcy Flow for different resolutions s.
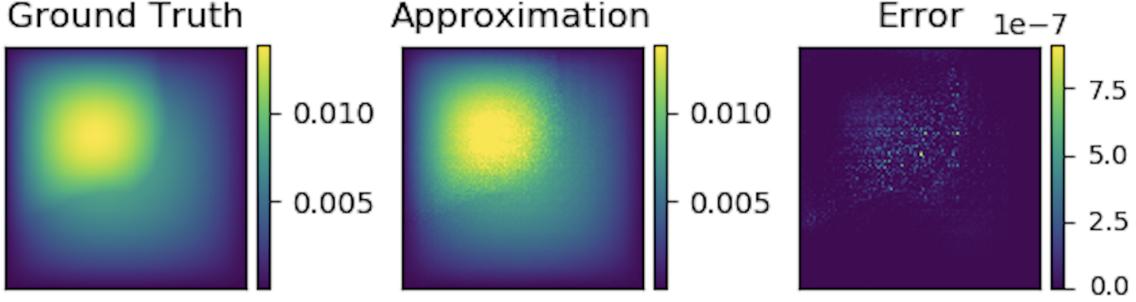
7.2.2 BURGERS' EQUATION

The results of the experiments on Burgers' equation are shown in Figure 8 and Table 3. As for the Darcy problem, our instantiation of the Fourier neural operator obtains nearly one order of magnitude lower relative error compared to any benchmarks. The Fourier neural operator has standard deviation 0.0010 and mean training error 0.0012. If one replaces the ReLU activation by GeLU, the test error of the FNO is further reduced from 0.0018 to 0.0007. We again observe the invariance of the error with respect to the resolution. It is possible to improve the performance on each model using modified architectures and problem specific feature engineering. Similarly, the PCA-enhanced DeepONet with a proper scaling can achieve 0.0194 relative L_2 error, as shown in Lu et al. (2021b), on a grid of resolution $s = 128$.

Networks	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
NN	0.4714	0.4561	0.4803	0.4645	0.4779	0.4452
GCN	0.3999	0.4138	0.4176	0.4157	0.4191	0.4198
FCN	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
DeepONet	0.0569	0.0617	0.0685	0.0702	0.0833	0.0857
GNO	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO	**0.0018**	**0.0018**	**0.0018**	**0.0019**	**0.0020**	**0.0019**

Table 3: Relative errors on 1-d Burgers' equation for different resolutions s.

7.2.3 ZERO-SHOT SUPER-RESOLUTION.

The neural operator is mesh-invariant, so it can be trained on a lower resolution and evaluated at a higher resolution, without seeing any higher resolution data (zero-shot super-resolution). Figure 9 shows an example of the Darcy Equation where we train the GNO model on 16×16 resolution data in the setting above and transfer to 256×256 resolution, demonstrating super-resolution in space.

Figure 9: Darcy, trained on 16×16, tested on 241×241
Graph kernel network for the solution of (6.2). It can be trained on a small resolution and will generalize to a large one. The Error is point-wise absolute squared error.

7.3 Navier-Stokes Equation

In the following section, we compare our four methods with different benchmarks on the Navier-Stokes equation introduced in subsection 6.4. The operator of interest is given by (46). We use the following abbreviations for the methods against which we benchmark.

- **ResNet:** 18 layers of 2-d convolution with residual connections He et al. (2016).

- **U-Net:** A popular choice for image-to-image regression tasks consisting of four blocks with 2-d convolutions and deconvolutions Ronneberger et al. (2015).

- **TF-Net:** A network designed for learning turbulent flows based on a combination of spatial and temporal convolutions Wang et al. (2020).

- **FNO-2d:** 2-d Fourier neural operator with an auto-regressive structure in time. We use the Fourier neural operator to model the local evolution from the previous 10 time steps to the next one time step, and iteratively apply the model to get the long-term trajectory. We set and $k_{\max,j} = 12, d_v = 32$.

- **FNO-3d:** 3-d Fourier neural operator that directly convolves in space-time. We use the Fourier neural operator to model the global evolution from the initial 10 time steps directly to the long-term trajectory. We set $k_{\max,j} = 12, d_v = 20$.

As shown in Table 4, the FNO-3D has the best performance when there is sufficient data ($\nu = 10^{-3}, N = 1000$ and $\nu = 10^{-4}, N = 10000$). For the configurations where the amount of data is insufficient ($\nu = 10^{-4}, N = 1000$ and $\nu = 10^{-5}, N = 1000$), all methods have $> 15\%$ error with FNO-2D achieving the lowest among our hyperparameter search. Note that we only present results for spatial resolution 64×64 since all benchmarks we compare against are designed for this resolution. Increasing the spatial resolution degrades their performance while FNO achieves the same errors.

Auto-regressive (2D) and Temporal Convolution (3D).   We investigate two standard formulation to model the time evolution: the auto-regressive model (2D) and the temporal convolution model (3D). **Auto-regressive models**: FNO-2D, U-Net, TF-Net, and ResNet all do 2D-convolution in the
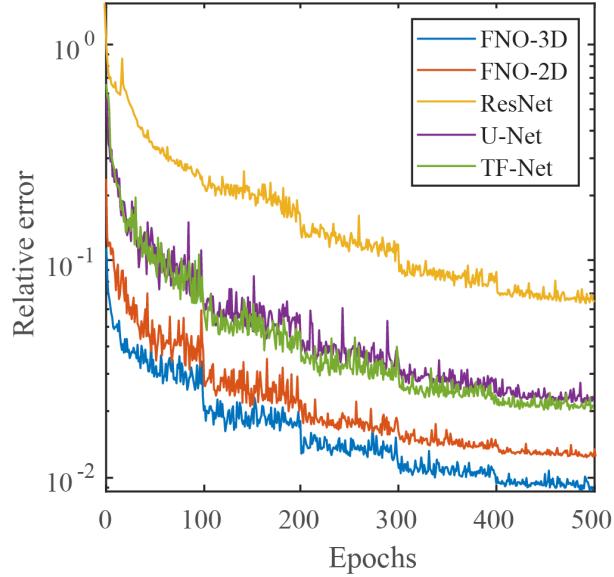
Figure 10: Benchmark on the Navier-Stokes
The learning curves on Navier-Stokes $\nu = 1e-3$ with different benchmarks. Train and test on the same resolution. For acronyms, see Section 7; details in Tables 4.

Config	Parameters	Time per epoch	$\nu = 10^{-3}$ $T = 50$ $N = 1000$	$\nu = 10^{-4}$ $T = 30$ $N = 1000$	$\nu = 10^{-4}$ $T = 30$ $N = 10000$	$\nu = 10^{-5}$ $T = 20$ $N = 1000$
FNO-3D	$6,558,537$	$38.99s$	**0.0086**	0.1918	**0.0820**	0.1893
FNO-2D	$414,517$	$127.80s$	0.0128	**0.1559**	0.0834	**0.1556**
U-Net	$24,950,491$	$48.67s$	0.0245	0.2051	0.1190	0.1982
TF-Net	$7,451,724$	$47.21s$	0.0225	0.2253	0.1168	0.2268
ResNet	$266,641$	$78.47s$	0.0701	0.2871	0.2311	0.2753

Table 4: Benchmarks on Navier Stokes (fixing resolution 64×64 for both training and testing).

spatial domain and recurrently propagate in the time domain (2D+RNN). The operator maps the solution at previous time steps to the next time step (2D functions to 2D functions). **Temporal convolution models**: on the other hand, FNO-3D performs convolution in space-time – it approximates the integral in time by a convolution. FNO-3D maps the initial time interval directly to the full trajectory (3D functions to 3D functions). The 2D+RNN structure can propagate the solution to any arbitrary time T in increments of a fixed interval length Δt, while the Conv3D structure is fixed to the interval $[0, T]$ but can transfer the solution to an arbitrary time-discretization. We find the 2D method work better for short time sequences while the 3D method more expressive and easier to train on longer sequences.

41

Networks	$s = 64$	$s = 128$	$s = 256$
FNO-3D	0.0098	0.0101	0.0106
FNO-2D	0.0129	0.0128	0.0126
U-Net	0.0253	0.0289	0.0344
TF-Net	0.0277	0.0278	0.0301

Table 5: Resolution study on Navier-stokes equation ($\nu = 10^{-3}$, $N = 200$, $T = 20$.)

7.3.1 ZERO-SHOT SUPER-RESOLUTION.

The neural operator is mesh-invariant, so it can be trained on a lower resolution and evaluated at a higher resolution, without seeing any higher resolution data (zero-shot super-resolution). Figure 11 shows an example where we train the FNO-3D model on $64 \times 64 \times 20$ resolution data in the setting above with ($\nu = 10^{-4}$, $N = 10000$) and transfer to $256 \times 256 \times 80$ resolution, demonstrating super-resolution in space-time. The Fourier neural operator is the only model among the benchmarks (FNO-2D, U-Net, TF-Net, and ResNet) that can do zero-shot super-resolution; the method works well not only on the spatial but also on the temporal domain.
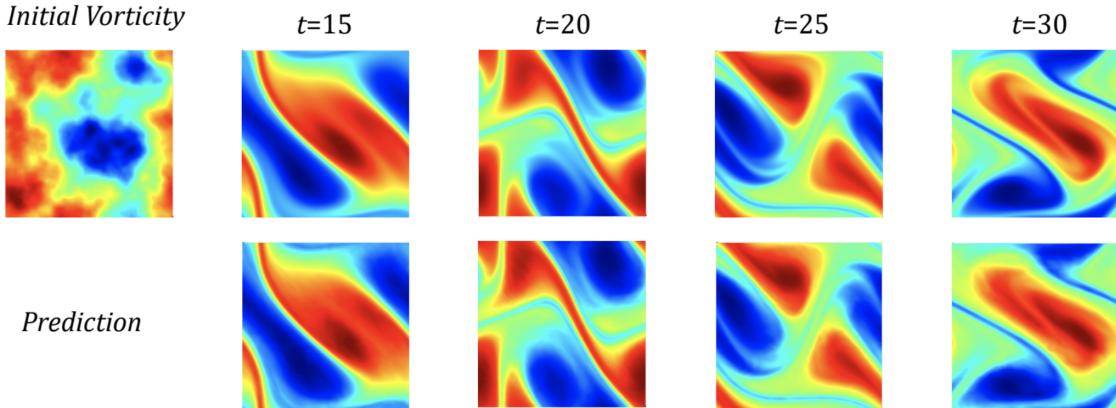

Figure 11: Zero-shot super-resolution
Vorticity field of the solution to the two-dimensional Navier-Stokes equation with viscosity $\nu = 10^4$ (Re$\approx$ 200); Ground truth on top and prediction on bottom. The model is trained on data that is discretized on a uniform 64×64 spatial grid and on a 20-point uniform temporal grid. The model is evaluated with a different initial condition that is discretized on a uniform 256×256 spatial grid and a 80-point uniform temporal grid.

7.3.2 SPECTRAL ANALYSIS

Figure 12 shows that all the methods are able to capture the spectral decay of the Navier-Stokes equation. Notice that, while the Fourier method truncates the higher frequency modes during the convolution, FNO can still recover the higher frequency components in the final prediction. Due to the way we parameterize R_ϕ, the function output by (26) has at most $k_{\max,j}$ Fourier modes per channel. This, however, does not mean that the Fourier neural operator can only approximate functions up to $k_{\max,j}$ modes. Indeed, the activation functions which occurs between integral operators
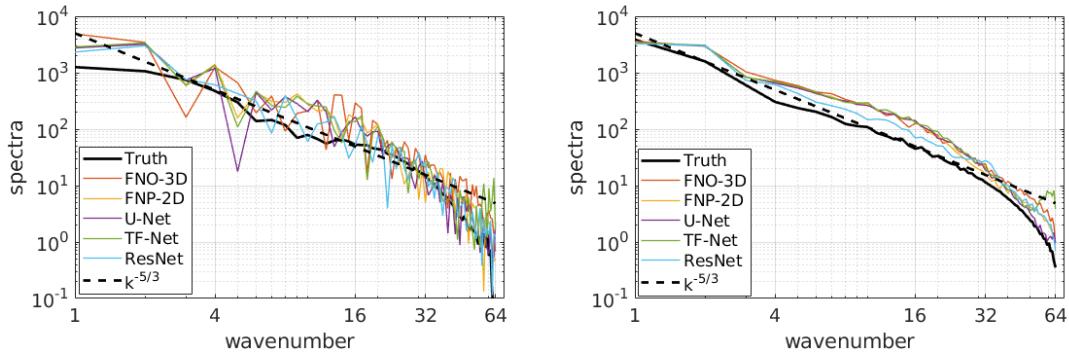
Figure 12: The spectral decay of the predictions of different methods

The spectral decay of the predictions of different models on the Navier-Stokes equation. The y-axis is the spectrum; the x-axis is the wavenumber. Left is the spectrum of one trajectory; right is the average of 40 trajectories.
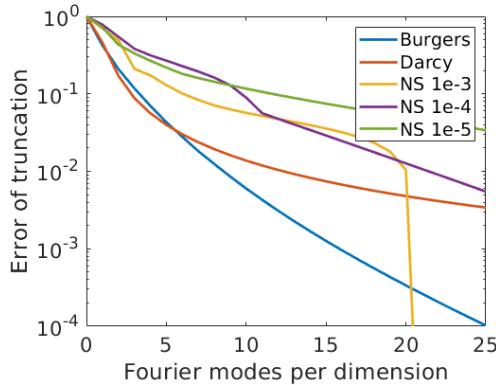

Figure 13: Spectral Decay in term of k_{max}

The error of truncation in one single Fourier layer without applying the linear transform R. The y-axis is the normalized truncation error; the x-axis is the truncation mode k_{max}.

and the final decoder network Q recover the high frequency modes. As an example, consider a solution to the Navier-Stokes equation with viscosity $\nu = 10^{-3}$. Truncating this function at 20 Fourier modes yields an error around 2% as shown in Figure 13, while the Fourier neural operator learns the parametric dependence and produces approximations to an error of $\leq 1\%$ with only $k_{max,j} = 12$ parameterized modes.

7.3.3 NON-PERIODIC BOUNDARY CONDITION.

Traditional Fourier methods work only with periodic boundary conditions. However, the Fourier neural operator does not have this limitation. This is due to the linear transform W (the bias term) which keeps the track of non-periodic boundary. As an example, the Darcy Flow and the time domain of Navier-Stokes have non-periodic boundary conditions, and the Fourier neural operator still learns the solution operator with excellent accuracy.

7.3.4 BAYESIAN INVERSE PROBLEM

As discussed in Section 6.4.1, we use the pCN method of Cotter et al. (2013) to draw samples from the posterior distribution of initial vorticities in the Navier-Stokes equation given sparse, noisy observations at time $T = 50$. We compare the Fourier neural operator acting as a surrogate model with the traditional solvers used to generate our train-test data (both run on GPU). We generate 25,000 samples from the posterior (with a 5,000 sample burn-in period), requiring 30,000 evaluations of the forward operator.

As shown in Figure 14, FNO and the traditional solver recover almost the same posterior mean which, when pushed forward, recovers well the later-time solution of the Navier-Stokes equation. In sharp contrast, FNO takes $0.005s$ to evaluate a single instance while the traditional solver, after being optimized to use the largest possible internal time-step which does not lead to blow-up, takes $2.2s$. This amounts to 2.5 minutes for the MCMC using FNO and over 18 hours for the traditional solver. Even if we account for data generation and training time (offline steps) which take 12 hours, using FNO is still faster. Once trained, FNO can be used to quickly perform multiple MCMC runs for different initial conditions and observations, while the traditional solver will take 18 hours for every instance. Furthermore, since FNO is differentiable, it can easily be applied to PDE-constrained optimization problems in which adjoint calculations are used as part of the solution procedure.
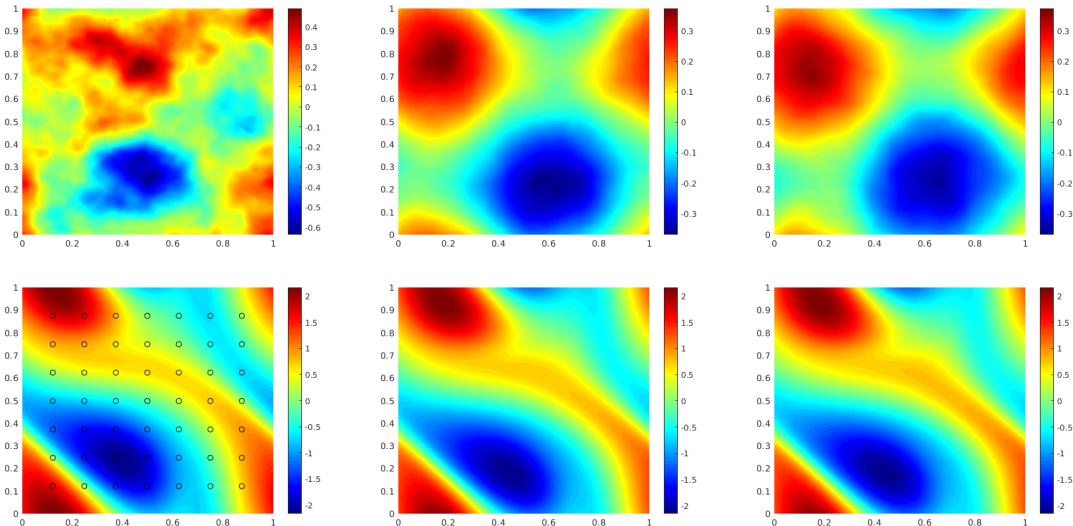

Figure 14: Results of the Bayesian inverse problem for the Navier-Stokes equation. The top left panel shows the true initial vorticity while bottom left panel shows the true observed vorticity at $T = 50$ with black dots indicating the locations of the observation points placed on a 7×7 grid. The top middle panel shows the posterior mean of the initial vorticity given the noisy observations estimated with MCMC using the traditional solver, while the top right panel shows the same thing but using FNO as a surrogate model. The bottom middle and right panels show the vorticity at $T = 50$ when the respective approximate posterior means are used as initial conditions.

7.4 Discussion and Comparison of the Four methods

In this section we will compare the four methods in term of expressiveness, complexity, refinability, and ingenuity.

7.4.1 INGENUITY

First we will discuss ingenuity, in other words, the design of the frameworks. The first method, GNO, relies on the Nyström approximation of the kernel, or the Monte Carlo approximation of the integration. It is the most simple and straightforward method. The second method, LNO, relies on the low-rank decomposition of the kernel operator. It is efficient when the kernel has a near low-rank structure. The third method, MGNO, is the combination of the first two. It has a hierarchical, multi-resolution decomposition of the kernel. The last one, FNO, is different from the first three; it restricts the integral kernel to induce a convolution.

GNO and MGNO are implemented using graph neural networks, which helps to define sampling and integration. The graph network library also allows sparse and distributed message passing. The LNO and FNO don't have sampling. They are faster without using the graph library.

	scheme	graph-based	kernel network
GNO	Nyström approximation	Yes	Yes
LNO	Low-rank approximation	No	Yes
MGNO	Multi-level graphs on GNO	Yes	Yes
FNO	Convolution theorem; Fourier features	No	No

Table 6: Ingenuity.

7.4.2 EXPRESSIVENESS

We measure the expressiveness by the training and testing error of the method. The full $O(J^2)$ integration always has the best results, but it is usually too expensive. As shown in the experiments 7.2.1 and 7.2.2, GNO usually has good accuracy, but its performance suffers from sampling. LNO works the best on the 1d problem (Burgers equation). It has difficulty on the 2d problem because it doesn't employ sampling to speed-up evaluation. MGNO has the multi-level structure, which gives it the benefit of the first two. Finally, FNO has overall the best performance. It is also the only method that can capture the challenging Navier-Stokes equation.

7.4.3 COMPLEXITY

The complexity of the four methods are listed in Table 7. GNO and MGNO have sampling. Their complexity depends on the number of nodes sampled J'. When using the full nodes. They are still quadratic. LNO has the lowest complexity $O(J)$. FNO, when using the fast Fourier transform, has complexity $O(J \log J)$.

In practice. FNO is faster then the other three methods because it doesn't have the kernel network κ. MGNO is relatively slower because of its multi-level graph structure.

	Complexity	Time per epochs in training
GNO	$O(J'^2 r^2)$	$4s$
LNO	$O(J)$	$20s$
MGNO	$\sum_l O(J_l^2 r_l^2) \sim O(J)$	$8s$
FNO	$(J \log J)$	$4s$

Table 7: Complexity (roundup to second on a single Nvidia V100 GPU)

7.4.4 REFINABILITY

Refineability measures the number of parameters used in the framework. Table 8 lists the accuracy of the relative error on Darcy Flow with respect to different number of parameters. Because GNO, LNO, and MGNO have the kernel networks, the slope of their error rates are flat: they can work with a very small number of parameters. On the other hand, FNO does not have the sub-network. It needs at a larger magnitude of parameters to obtain an acceptable error rate.

Number of parameters	10^3	10^4	10^5	10^6
GNO	0.075	0.065	0.060	0.035
LNO	0.080	0.070	0.060	0.040
MGNO	0.070	0.050	0.040	0.030
FNO	0.200	0.035	0.020	0.015

Table 8: Refinability.
The relative error on Darcy Flow with respect to different number of parameters. The errors above are approximated value roundup to 0.05. They are the lowest test error achieved by the model, given the model's number of parameters $|\theta|$ is bounded by $10^3, 10^4, 10^5, 10^6$ respectively.

7.4.5 ROBUSTNESS

We conclude with experiments investigating the robustness of Fourier neural operator to noise. We study: a) training on clean (noiseless) data and testing with clean and noisy data; b) training on clean (noiseless) data and testing with clean and noisy data. When creating noisy data we map a to noisy a' as follows: at every grid-point x we set

$$a(x)' = a(x) + 0.1 \cdot \|a\|_\infty \xi,$$

where $\xi \sim \mathcal{N}(0, 1)$ is drawn i.i.d. at every grid point; this is similar to the setting adopted in Lu et al. (2021b). We also study the 1d advection equation as an additional test case, following the setting in Lu et al. (2021b) in which the input data is a random square wave, defined by an $\mathbb{R}^3$-valued random variable.

As shown in the top half of Table 9 and Figure 15, we observe the Fourier neural operator is robust with respect to the (test) noise level on all four problems. In particular, on the advection problem, it has about 10% error with 10% noise. The Darcy and Navier-Stokes operators are smoothing, and the Fourier neural operator obtains lower than 10% error in all scenarios. However the FNO

Problems	Training error	Test (clean)	Test (noisy)
Burgers	0.002	0.002	0.018
Advection	0.002	0.002	0.094
Darcy	0.006	0.011	0.012
Navier-Stokes	0.024	0.024	0.039
Burgers (train with noise)	0.011	0.004	0.011
Advection (train with noise)	0.020	0.010	0.019
Darcy (train with noise)	0.007	0.012	0.012
Navier-Stokes (train with noise)	0.026	0.026	0.025

Table 9: Robustness.

is less robust on the advection equation, which is not smoothing, and on Burgers equation which, whilst smoothing also forms steep fronts.

A straightforward approach to enhance the robustness is to train the model with noise. As shown in the bottom half of Table 9, the Fourier neural operator has no gap between the clean data and noisy data when training with noise. However, noise in training may degrade the performance on the clean data, as a trade-off. In general, augmenting the training data with noise leads to robustness. For example, in the auto-regressive modeling of dynamical systems, training the model with noise will reduce error accumulation in time, and thereby help the model to predict over longer time-horizons (Pfaff et al., 2020). We also observed that other regularization techniques such as early-stopping and weight decay improve robustness. Using a higher spatial resolution also helps.

The advection problem is a hard problem for the FNO since it has discontinuities; similar issues arise when using spectral methods for conservation laws. One can modify the architecture to address such discontinuities accordingly. For example, Wen et al. (2021) enhance the FNO by composing a CNN or UNet branch with the Fourier layer; the resulting composite model outperforms the basic FNO on multiphase flow with high contrast and sharp shocks. However the CNN and UNet take the method out of the realm of discretization-invariant methods; further work is required to design discretization-invariant image-processing tools, such as the identification of discontinuities.

8. Literature Review

We outline the major neural network-based approaches for the solution of PDEs.

Finite-dimensional Operators. An immediate approach to approximate $\mathcal{G}^\dagger$ is to parameterize it as a deep convolutional neural network (CNN) between the finite-dimensional Euclidean spaces on which the data is discretized i.e. $\mathcal{G} : \mathbb{R}^K \times \Theta \to \mathbb{R}^K$ (Guo et al., 2016; Zhu and Zabaras, 2018; Adler and Oktem, 2017; Bhatnagar et al., 2019; Kutyniok et al., 2022). Khoo et al. (2021) concerns a similar setting, but with output space $\mathbb{R}$. Such approaches are, by definition, not mesh independent and need modifications to the architecture for different resolution and discretization of D in order to achieve consistent error (if at all possible). We demonstrate this issue numerically in Section 7. Furthermore, these approaches are limited to the discretization size and geometry of the training data and hence it is not possible to query solutions at new points in the domain. In contrast for our method, we show in Section 7, both invariance of the error to grid resolution, and
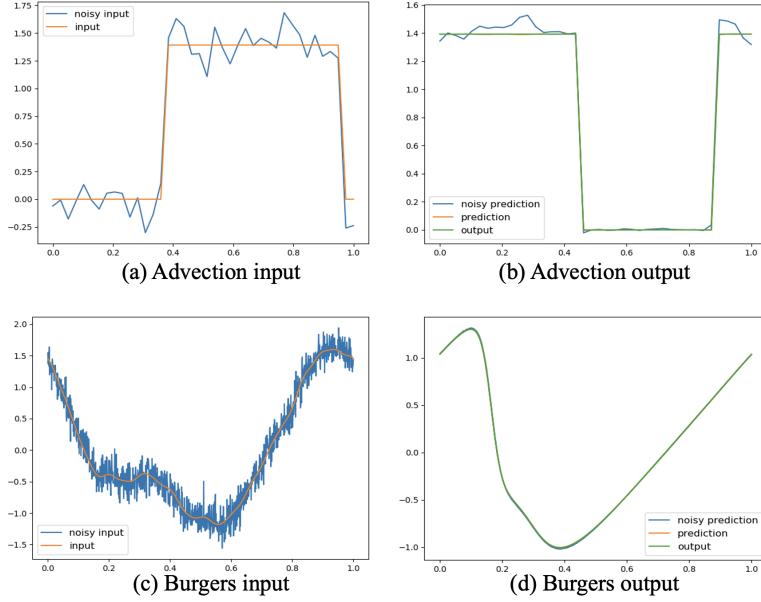
Figure 15: Robustness on Advection and Burgers equations
(a) The input of Advection equation ($s = 40$). The orange curve is the clean input; the blue curve is the noisy input. (b) The output of Advection equation. The green curve is the ground truth output; the orange curve is the prediction of FNO with clean input (overlapping with the ground truth); the blue curve is the prediction on the noisy input. Figure (c) and (d) are for Burgers' equation ($s = 1000$) correspondingly.

the ability to transfer the solution between meshes. The work Ummenhofer et al. (2020) proposed a continuous convolution network for fluid problems, where off-grid points are sampled and linearly interpolated. However the continuous convolution method is still constrained by the underlying grid which prevents generalization to higher resolutions. Similarly, to get finer resolution solution, Jiang et al. (2020) proposed learning super-resolution with a U-Net structure for fluid mechanics problems. However fine-resolution data is needed for training, while neural operators are capable of zero-shot super-resolution with no new data.

DeepONet A novel operator regression architecture, named DeepONet, was recently proposed by Lu et al. (2019, 2021a); it builds an iterated or deep structure on top of the shallow architecture proposed in Chen and Chen (1995). The architecture consists of two neural networks: a branch net applied on the input functions and a trunk net applied on the querying locations in the output space. The original work of Chen and Chen (1995) provides a universal approximation theorem, and more recently Lanthaler et al. (2021) developed an error estimate for DeepONet itself. The standard DeepONet structure is a linear approximation of the target operator, where the trunk net and branch net learn the coefficients and basis. On the other hand, the neural operator setting is heavily inspired by the advances in deep learning and is a non-linear approximation, which makes it constructively more expressive. A detailed discussion of DeepONet is provided in Section 5.1 and as well as a numerical comparison to DeepONet in Section 7.2.

Physics Informed Neural Networks (PINNs), Deep Ritz Method (DRM), and Deep Galerkin Method (DGM). A different approach is to directly parameterize the solution u as a neural net-

work $u : \bar{D} \times \Theta \rightarrow \mathbb{R}$ (E and Yu, 2018; Raissi et al., 2019; Sirignano and Spiliopoulos, 2018; Bar and Sochen, 2019; Smith et al., 2020; Pan and Duraisamy, 2020; Beck et al., 2021). This approach is designed to model one specific instance of the PDE, not the solution operator. It is mesh-independent, but for any given new parameter coefficient function $a \in \mathcal{A}$, one would need to train a new neural network u_a which is computationally costly and time consuming. Such an approach closely resembles classical methods such as finite elements, replacing the linear span of a finite set of local basis functions with the space of neural networks.

ML-based Hybrid Solvers Similarly, another line of work proposes to enhance existing numerical solvers with neural networks by building hybrid models (Pathak et al., 2020; Um et al., 2020a; Greenfeld et al., 2019) These approaches suffer from the same computational issue as classical methods: one needs to solve an optimization problem for every new parameter similarly to the PINNs setting. Furthermore, the approaches are limited to a setting in which the underlying PDE is known. Purely data-driven learning of a map between spaces of functions is not possible.

Reduced Basis Methods. Our methodology most closely resembles the classical reduced basis method (RBM) (DeVore, 2014) or the method of Cohen and DeVore (2015). The method introduced here, along with the contemporaneous work introduced in the papers (Bhattacharya et al., 2020; Nelsen and Stuart, 2021; Opschoor et al., 2020; Schwab and Zech, 2019; O'Leary-Roseberry et al., 2020; Lu et al., 2019; Fresca and Manzoni, 2022), are, to the best of our knowledge, amongst the first practical supervised learning methods designed to learn maps between infinite-dimensional spaces. Our methodology addresses the mesh-dependent nature of the approach in the papers (Guo et al., 2016; Zhu and Zabaras, 2018; Adler and Oktem, 2017; Bhatnagar et al., 2019) by producing a single set of network parameters that can be used with different discretizations. Furthermore, it has the ability to transfer solutions between meshes and indeed between different discretization methods. Moreover, it needs only to be trained once on the equation set $\{a_j, u_j\}_{j=1}^N$. Then, obtaining a solution for a new $a \sim \mu$ only requires a forward pass of the network, alleviating the major computational issues incurred in (E and Yu, 2018; Raissi et al., 2019; Herrmann et al., 2020; Bar and Sochen, 2019) where a different network would need to be trained for each input parameter. Lastly, our method requires no knowledge of the underlying PDE: it is purely data-driven and therefore non-intrusive. Indeed the true map can be treated as a black-box, perhaps to be learned from experimental data or from the output of a costly computer simulation, not necessarily from a PDE.

Continuous Neural Networks. Using continuity as a tool to design and interpret neural networks is gaining currency in the machine learning community, and the formulation of ResNet as a continuous time process over the depth parameter is a powerful example of this (Haber and Ruthotto, 2017; E, 2017). The concept of defining neural networks in infinite-dimensional spaces is a central problem that has long been studied (Williams, 1996; Neal, 1996; Roux and Bengio, 2007; Globerson and Livni, 2016; Guss, 2016). The general idea is to take the infinite-width limit which yields a non-parametric method and has connections to Gaussian Process Regression (Neal, 1996; Matthews et al., 2018; Garriga-Alonso et al., 2018), leading to the introduction of deep Gaussian processes (Damianou and Lawrence, 2013; Dunlop et al., 2018). Thus far, such methods have not yielded efficient numerical algorithms that can parallel the success of convolutional or recurrent neural networks for the problem of approximating mappings between finite dimensional spaces. Despite the superficial similarity with our proposed work, this body of work differs substantially from what we are proposing: in our work we are motivated by the continuous dependence of the data, in the

input or output spaces, in spatial or spatio-temporal variables; in contrast the work outlined in this paragraph uses continuity in an artificial algorithmic depth or width parameter to study the network architecture when the depth or width approaches infinity, but the input and output spaces remain of fixed finite dimension.

Nyström Approximation, GNNs, and Graph Neural Operators (GNOs). The graph neural operators (Section 4.1) has an underlying Nyström approximation formulation (Nyström, 1930) which links different grids to a single set of network parameters. This perspective relates our continuum approach to Graph Neural Networks (GNNs). GNNs are a recently developed class of neural networks that apply to graph-structured data; they have been used in a variety of applications. Graph networks incorporate an array of techniques from neural network design such as graph convolution, edge convolution, attention, and graph pooling (Kipf and Welling, 2016; Hamilton et al., 2017; Gilmer et al., 2017; Veličković et al., 2017; Murphy et al., 2018). GNNs have also been applied to the modeling of physical phenomena such as molecules (Chen et al., 2019) and rigid body systems (Battaglia et al., 2018) since these problems exhibit a natural graph interpretation: the particles are the nodes and the interactions are the edges. The work (Alet et al., 2019) performs an initial study that employs graph networks on the problem of learning solutions to Poisson's equation, among other physical applications. They propose an encoder-decoder setting, constructing graphs in the latent space, and utilizing message passing between the encoder and decoder. However, their model uses a nearest neighbor structure that is unable to capture non-local dependencies as the mesh size is increased. In contrast, we directly construct a graph in which the nodes are located on the spatial domain of the output function. Through message passing, we are then able to directly learn the kernel of the network which approximates the PDE solution. When querying a new location, we simply add a new node to our spatial graph and connect it to the existing nodes, avoiding interpolation error by leveraging the power of the Nyström extension for integral operators.

Low-rank Kernel Decomposition and Low-rank Neural Operators (LNOs). Low-rank decomposition is a popular method used in kernel methods and Gaussian process (Kulis et al., 2006; Bach, 2013; Lan et al., 2017; Gardner et al., 2018). We present the low-rank neural operator in Section 4.2 where we structure the kernel network as a product of two factor networks inspired by Fredholm theory. The low-rank method, while simple, is very efficient and easy to train especially when the target operator is close to linear. Khoo and Ying (2019) proposed a related neural network with low-rank structure to approximate the inverse of differential operators. The framework of two factor networks is also similar to the trunk and branch network used in DeepONet (Lu et al., 2019). But in our work, the factor networks are defined on the physical domain and non-local information is accumulated through integration with respect to the Lebesgue measure. In contrast, DeepONet(s) integrate against delta measures at a set of pre-defined nodal points that are usually taken to be the grid on which the data is given. See section 5.1 for further discussion.

Multipole, Multi-resolution Methods, and Multipole Graph Neural Operators (MGNOs). To efficiently capture long-range interaction, multi-scale methods such as the classical fast multipole methods (FMM) have been developed (Greengard and Rokhlin, 1997). Based on the assumption that long-range interactions decay quickly, FMM decomposes the kernel matrix into different ranges and hierarchically imposes low-rank structures on the long-range components (hierarchical matrices) (Börm et al., 2003). This decomposition can be viewed as a specific form of the multi-resolution matrix factorization of the kernel (Kondor et al., 2014; Börm et al., 2003). For example, the works of

Fan et al. (2019c,b); He and Xu (2019) propose a similar multipole expansion for solving parametric PDEs on structured grids. However, the classical FMM requires nested grids as well as the explicit form of the PDEs. In Section 4.3, we propose the multipole graph neural operator (MGNO) by generalizing this idea to arbitrary graphs in the data-driven setting, so that the corresponding graph neural networks can learn discretization-invariant solution operators which are fast and can work on complex geometries.

Fourier Transform, Spectral Methods, and Fourier Neural Operators (FNOs). The Fourier transform is frequently used in spectral methods for solving differential equations since differentiation is equivalent to multiplication in the Fourier domain. Fourier transforms have also played an important role in the development of deep learning. They are used in theoretical work, such as the proof of the neural network universal approximation theorem (Hornik et al., 1989) and related results for random feature methods (Rahimi and Recht, 2008); empirically, they have been used to speed up convolutional neural networks (Mathieu et al., 2013). Neural network architectures involving the Fourier transform or the use of sinusoidal activation functions have also been proposed and studied (Bengio et al., 2007; Mingo et al., 2004; Sitzmann et al., 2020). Recently, some spectral methods for PDEs have been extended to neural networks (Fan et al., 2019a,c; Kashinath et al., 2020). In Section 4.4, we build on these works by proposing the Fourier neural operator architecture defined directly in Fourier space with quasi-linear time complexity and state-of-the-art approximation capabilities.

Sources of Error In this paper we will study the error resulting from approximating an operator (mapping between Banach spaces) from within a class of finitely-parameterized operators. We show that the resulting error, expressed in terms of universal approximation of operators over a compact set or in terms of a resulting risk, can be driven to zero by increasing the number of parameters, and refining the approximations inherent in the neural operator architecture. In practice there will be two other sources of approximation error: firstly from the discretization of the data; and secondly from the use of empirical risk minimization over a finite data set to determine the parameters. Balancing all three sources of error is key to making algorithms efficient. However we do not study these other two sources of error in this work. Furthermore we do not study how the number of parameters in our approximation grows as the error tolerance is refined. Generally, this growth may be super-exponential as shown in (Kovachki et al., 2021). However, for certain classes of operators and related approximation methods, it is possible to beat the curse of dimensionality; we refer the reader to the works (Lanthaler et al., 2021; Kovachki et al., 2021) for detailed analyses demonstrating this. Finally we also emphasize that there is a potential source of error from the optimization procedure which attempts to minimize the empirical risk: it may not achieve the global minumum. Analysis of this error in the context of operator approximation has not been undertaken.

9. Approximation Theory

The paper by Chen and Chen (1995) provides the first universal approximation theorem for operator approximation via neural networks, and the paper by Bhattacharya et al. (2020) provides an alternative architecture and approximation result. The analysis of Chen and Chen (1995) was recently extended in significant ways in the paper by Lanthaler et al. (2021) where, for the first time, the curse of dimensionality is addressed, and resolved, for certain specific operator learning problems, using the DeepOnet generalization Lu et al. (2019, 2021a) of Chen and Chen (1995). The paper Lanthaler

et al. (2021) was generalized to study operator approximation, and the curse of dimensionality, for the FNO, in Kovachki et al. (2021).

Unlike the finite-dimensional setting, the choice of input and output spaces $\mathcal{A}$ and $\mathcal{U}$ for the mapping $\mathcal{G}^\dagger$ play a crucial role in the approximation theory due to the distinctiveness of the induced norm topologies. In this section, we prove universal approximation theorems for neural operators both with respect to the topology of uniform convergence over compact sets and with respect to the topology induced by the Bochner norm (3). We focus our attention on the Lebesgue, Sobolev, continuous, and continuously differentiable function classes as they have numerous applications in scientific computing and machine learning problems. Unlike the results of Bhattacharya et al. (2020); Kovachki et al. (2021) which rely on the Hilbertian structure of the input and output spaces or the results of Chen and Chen (1995); Lanthaler et al. (2021) which rely on the continuous functions, our results extend to more general Banach spaces as specified by Assumptions 9 and 10 (stated in Section 9.3) and are, to the best of our knowledge, the first of their kind to apply at this level of generality.

Our method of proof proceeds by making use of the following two observations. First we establish the Banach space approximation property Grothendieck (1955) for the input and output spaces of interest, which allows for a finite dimensionalization of the problem. In particular, we prove that the Banach space approximation property holds for various function spaces defined on Lipschitz domains; the precise result we need, while unsurprising, seems to be missing from the functional analysis literature and so we provide statement and proof. Details are given in Appendix A. Second, we establish that integral kernel operators with smooth kernels can be used to approximate linear functionals of various input spaces. In doing so, we establish a Riesz-type representation theorem for the continuously differentiable functions. Such a result is not surprising and mimics the well-known result for Sobolev spaces; however in the form we need it we could not find the result in the functional analysis literature and so we provide statement and proof. Details are given in Appendix B. With these two facts, we construct a neural operator which linearly maps any input function to a finite vector then non-linearly maps this vector to a new finite vector which is then used to form the coefficients of a basis expansion for the output function. We reemphasize that our approximation theory uses the fact that neural operators can be reduced to a linear method of approximation (as pointed out in Section 5.1) and does not capture any benefits of nonlinear approximation. However these benefits *are* present in the architecture and are exploited by the trained networks we find in practice. Exploiting their nonlinear nature to potentially obtain improved rates of approximation remains an interesting direction for future research.

The rest of this section is organized as follows. In Subsection 9.1, we define allowable activation functions and the set of neural operators used in our theory, noting that they constitute a subclass of the neural operators defined in Section 5. In Subsection 9.3, we state and prove our main universal approximation theorems.

9.1 Neural Operators

For any $n \in \mathbb{N}$ and $\sigma : \mathbb{R} \to \mathbb{R}$, we define the set of real-valued n-layer neural networks on $\mathbb{R}^d$ by

$$
\begin{aligned}
\mathsf{N}_n(\sigma; \mathbb{R}^d) := \{ f : \mathbb{R}^d \to \mathbb{R} : f(x) = W_n \sigma(\dots W_1 \sigma(W_0 x + b_0) + b_1 \dots) + b_n, \\
W_0 \in \mathbb{R}^{d_0 \times d}, W_1 \in \mathbb{R}^{d_1 \times d_0}, \dots, W_n \in \mathbb{R}^{1 \times d_{n-1}}, \\
b_0 \in \mathbb{R}^{d_0}, b_1 \in \mathbb{R}^{d_1}, \dots, b_n \in \mathbb{R}, \ d_0, d_1, \dots, d_{n-1} \in \mathbb{N} \}.
\end{aligned}
$$

We define the set of $\mathbb{R}^{d'}$-valued neural networks simply by stacking real-valued networks

$$\mathsf{N}_n(\sigma; \mathbb{R}^d, \mathbb{R}^{d'}) := \{f : \mathbb{R}^d \to \mathbb{R}^{d'} : f(x) = \big(f_1(x), \dots, f_{d'}(x)\big), \ f_1, \dots, f_{d'} \in \mathsf{N}_n(\sigma; \mathbb{R}^d)\}.$$

We remark that we could have defined $\mathsf{N}_n(\sigma; \mathbb{R}^d, \mathbb{R}^{d'})$ by letting $W_n \in \mathbb{R}^{d' \times d_n}$ and $b_n \in \mathbb{R}^{d'}$ in the definition of $\mathsf{N}_n(\sigma; \mathbb{R}^d)$ because we allow arbitrary width, making the two definitions equivalent; however the definition as presented is more convenient for our analysis. We also employ the preceding definition with $\mathbb{R}^d$ and $\mathbb{R}^{d'}$ replaced by spaces of matrices. For any $m \in \mathbb{N}_0$, we define the set of allowable activation functions as the continuous $\mathbb{R} \to \mathbb{R}$ maps which make neural networks dense in $C^m(\mathbb{R}^d)$ on compacta at any fixed depth,

$$\mathsf{A}_m := \{\sigma \in C(\mathbb{R}) : \exists n \in \mathbb{N} \text{ s.t. } \mathsf{N}_n(\sigma; \mathbb{R}^d) \text{ is dense in } C^m(K) \ \forall K \subset \mathbb{R}^d \text{ compact}\}.$$

It is shown in (Pinkus, 1999, Theorem 4.1) that $\{\sigma \in C^m(\mathbb{R}) : \sigma \text{ is not a polynomial}\} \subseteq \mathsf{A}_m$ with $n = 1$. Clearly $\mathsf{A}_{m+1} \subseteq \mathsf{A}_m$.

We define the set of linearly bounded activations as

$$\mathsf{A}_m^{\mathsf{L}} := \left\{\sigma \in \mathsf{A}_m : \sigma \text{ is Borel measurable} , \ \sup_{x \in \mathbb{R}} \frac{|\sigma(x)|}{1 + |x|} < \infty\right\},$$

noting that any globally Lipschitz, non-polynomial, C^m-function is contained in $\mathsf{A}_m^{\mathsf{L}}$. Most activation functions used in practice fall within this class, for example, $\mathrm{ReLU} \in \mathsf{A}_0^{\mathsf{L}}$, $\mathrm{ELU} \in \mathsf{A}_1^{\mathsf{L}}$ while $\tanh, \mathrm{sigmoid} \in \mathsf{A}_m^{\mathsf{L}}$ for any $m \in \mathbb{N}_0$.

For approximation in a Bochner norm, we will be interested in constructing globally bounded neural networks which can approximate the identity over compact sets as done in (Lanthaler et al., 2021; Bhattacharya et al., 2020). This allows us to control the potential unboundedness of the support of the input measure by exploiting the fact that the probability of an input must decay to zero in unbounded regions. Following (Lanthaler et al., 2021), we introduce the forthcoming definition which uses the notation of the diameter of a set. In particular, the diameter of any set $S \subseteq \mathbb{R}^d$ is defined as, for $| \cdot |_2$ the Euclidean norm on $\mathbb{R}^d$,

$$\mathrm{diam}_2(S) := \sup_{x,y \in S} |x - y|_2.$$

Definition 7 *We denote by* BA *the set of maps* $\sigma \in \mathsf{A}_0$ *such that, for any compact set* $K \subset \mathbb{R}^d$, $\epsilon > 0$, *and* $C \geq \mathrm{diam}_2(K)$, *there exists a number* $n \in \mathbb{N}$ *and a neural network* $f \in \mathsf{N}_n(\sigma; \mathbb{R}^d, \mathbb{R}^{d'})$ *such that*

$$|f(x) - x|_2 \leq \epsilon, \qquad \forall x \in K,$$
$$|f(x)|_2 \leq C, \qquad \forall x \in \mathbb{R}^d.$$

It is shown in (Lanthaler et al., 2021, Lemma C.1) that $\mathrm{ReLU} \in \mathsf{A}_0^{\mathsf{L}} \cap \mathsf{BA}$ with $n = 3$.

We will now define the specific class of neural operators for which we prove a universal approximation theorem. It is important to note that the class with which we work is a simplification of the one given in (6). In particular, the lifting and projection operators $\mathcal{Q}, \mathcal{P}$, together with the final activation function σ_n, are set to the identity, and the local linear operators $W_0, \dots, W_{n-1}$ are set to zero. In our numerical studies we have in any case typically set σ_n to the identity. However we have

found that learning the local operators $\mathcal{Q}, \mathcal{P}$ and $W_0, \ldots, W_{n-1}$ is beneficial in practice; extending the universal approximation theorems given here to explain this benefit would be an important but non-trivial development of the analysis we present here.

Let $D \subset \mathbb{R}^d$ be a domain. For any $\sigma \in \mathsf{A}_0$, we define the set of affine kernel integral operators by

$$\mathsf{IO}(\sigma; D, \mathbb{R}^{d_1}, \mathbb{R}^{d_2}) = \{f \mapsto \int_D \kappa(\cdot, y) f(y) \, \mathsf{d}y + b : \kappa \in \mathsf{N}_{n_1}(\sigma; \mathbb{R}^d \times \mathbb{R}^d, \mathbb{R}^{d_2 \times d_1}),$$
$$b \in \mathsf{N}_{n_2}(\sigma; \mathbb{R}^d, \mathbb{R}^{d_2}), \ n_1, n_2 \in \mathbb{N}\},$$

for any $d_1, d_2 \in \mathbb{N}$. Clearly, since $\sigma \in \mathsf{A}_0$, any $S \in \mathsf{IO}(\sigma; D, \mathbb{R}^{d_1}, \mathbb{R}^{d_2})$ acts as $S : L^p(D; \mathbb{R}^{d_1}) \to L^p(D; \mathbb{R}^{d_2})$ for any $1 \leq p \leq \infty$ since $\kappa \in C(\bar{D} \times \bar{D}; \mathbb{R}^{d_2 \times d_1})$ and $b \in C(\bar{D}; \mathbb{R}^{d_2})$. For any $n \in \mathbb{N}_{\geq 2}, d_a, d_u \in \mathbb{N}, D \subset \mathbb{R}^d, D' \subset \mathbb{R}^{d'}$ domains, and $\sigma_1 \in \mathsf{A}_0^{\mathsf{L}}, \sigma_2, \sigma_3 \in \mathsf{A}_0$, we define the set of n-layer neural operators by

$$\mathsf{NO}_n(\sigma_1, \sigma_2, \sigma_3; D, D', \mathbb{R}^{d_a}, \mathbb{R}^{d_u}) = \{f \mapsto \int_D \kappa_n(\cdot, y) \big(S_{n-1}\sigma_1(\ldots S_2\sigma_1(S_1(S_0 f)) \ldots)\big)(y) \, \mathsf{d}y :$$
$$S_0 \in \mathsf{IO}(\sigma_2, D; \mathbb{R}^{d_a}, \mathbb{R}^{d_1}), \ldots S_{n-1} \in \mathsf{IO}(\sigma_2, D; \mathbb{R}^{d_{n-1}}, \mathbb{R}^{d_n}),$$
$$\kappa_n \in \mathsf{N}_l(\sigma_3; \mathbb{R}^{d'} \times \mathbb{R}^d, \mathbb{R}^{d_u \times d_n}), \ d_1, \ldots, d_n, l \in \mathbb{N}\}.$$

When $d_a = d_u = 1$, we will simply write $\mathsf{NO}_n(\sigma_1, \sigma_2, \sigma_3; D, D')$. Since σ_1 is linearly bounded, we can use a result about compositions of maps in L^p spaces such as (Dudley and Norvaiša, 2010, Theorem 7.13) to conclude that any $G \in \mathsf{NO}_n(\sigma_1, \sigma_2, \sigma_3, D, D'; \mathbb{R}^{d_a}, \mathbb{R}^{d_u})$ acts as $G : L^p(D; \mathbb{R}^{d_a}) \to L^p(D'; \mathbb{R}^{d_u})$. Note that it is only in the last layer that we transition from functions defined over domain D to functions defined over domain D'.

When the input space of an operator of interest is $C^m(\bar{D})$, for $m \in \mathbb{N}$, we will need to take in derivatives explicitly as they cannot be learned using kernel integration as employed in the current construction given in Lemma 30; note that this is *not* the case for $W^{m,p}(D)$ as shown in Lemma 28. We will therefore define the set of m-th order neural operators by

$$\mathsf{NO}_n^m(\sigma_1, \sigma_2, \sigma_3; D, D', \mathbb{R}^{d_a}, \mathbb{R}^{d_u}) = \{(\partial^{\alpha_1} f, \ldots, \partial^{\alpha_{J_m}} f) \mapsto G(\partial^{\alpha_1} f, \ldots, \partial^{\alpha_{J_m}} f) :$$
$$G \in \mathsf{NO}_n(\sigma_1, \sigma_2, \sigma_3; D, D', \mathbb{R}^{J_m d_a}, \mathbb{R}^{d_u})\}$$

where $\alpha_1, \ldots, \alpha_{J_m} \in \mathbb{N}^d$ is an enumeration of the set $\{\alpha \in \mathbb{N}^d : 0 \leq |\alpha|_1 \leq m\}$. Since we only use the m-th order operators when dealing with spaces of continuous functions, each element of NO_n^m can be thought of as a mapping from a product space of spaces of the form $C^{m-|\alpha_j|}(\bar{D}; \mathbb{R}^{d_a})$ for all $j \in \{1, \ldots, J_m\}$ to an appropriate Banach space of interest.

9.2 Discretization Invariance

Given the construction above and the definition of discretization invariance in Definition 4, in the following we prove that neural operators are discretization invariant deep learning models.

Theorem 8 *Let $D \subset \mathbb{R}^d$ and $D' \subset \mathbb{R}^{d'}$ be two domains for some $d, d' \in \mathbb{N}$. Let $\mathcal{A}$ and $\mathcal{U}$ be real-valued Banach function spaces on D and D' respectively. Suppose that $\mathcal{A}$ and $\mathcal{U}$ can be continuously embedded in $C(\bar{D})$ and $C(\bar{D}')$ respectively and that $\sigma_1, \sigma_2, \sigma_3 \in C(\mathbb{R})$. Then, for any $n \in \mathbb{N}$, the set of neural operators $\mathsf{NO}_n(\sigma_1, \sigma_2, \sigma_3; D, D')$ whose elements are viewed as maps $\mathcal{A} \to \mathcal{U}$ is discretization-invariant.*

$$
\begin{array}{ccccc}
\mathcal{A} & \xrightarrow{\;\;F\;\;} & \mathbb{R}^J & \longrightarrow & \mathcal{A} \\
{\scriptstyle \mathcal{G}^\dagger}\downarrow & & {\scriptstyle \psi}\downarrow & & {\scriptstyle \mathcal{G}^\dagger}\downarrow \\
\mathcal{U} & \longrightarrow & \mathbb{R}^{J'} & \xrightarrow{\;\;G\;\;} & \mathcal{U}
\end{array}
$$

Figure 16: A schematic overview of the maps used to approximate $\mathcal{G}^\dagger$.

The proof, provided in appendix E, constructs a sequence of finite dimensional maps which approximate the neural operator by Riemann sums and shows uniform converges of the error over compact sets of $\mathcal{A}$.

9.3 Approximation Theorems

Let $\mathcal{A}$ and $\mathcal{U}$ be Banach function spaces on the domains $D \subset \mathbb{R}^d$ and $D' \subset \mathbb{R}^{d'}$ respectively. We will work in the setting where functions in $\mathcal{A}$ or $\mathcal{U}$ are real-valued, but note that all results generalize in a straightforward fashion to the vector-valued setting. We are interested in the approximation of nonlinear operators $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ by neural operators. We will make the following assumptions on the spaces $\mathcal{A}$ and $\mathcal{U}$.

Assumption 9 *Let $D \subset \mathbb{R}^d$ be a Lipschitz domain for some $d \in \mathbb{N}$. One of the following holds*

1. $\mathcal{A} = L^{p_1}(D)$ *for some* $1 \leq p_1 < \infty$.

2. $\mathcal{A} = W^{m_1, p_1}(D)$ *for some* $1 \leq p_1 < \infty$ *and* $m_1 \in \mathbb{N}$,

3. $\mathcal{A} = C(\bar{D})$.

Assumption 10 *Let $D' \subset \mathbb{R}^{d'}$ be a Lipschitz domain for some $d' \in \mathbb{N}$. One of the following holds*

1. $\mathcal{U} = L^{p_2}(D')$ *for some* $1 \leq p_2 < \infty$, *and* $m_2 = 0$,

2. $\mathcal{U} = W^{m_2, p_2}(D')$ *for some* $1 \leq p_2 < \infty$ *and* $m_2 \in \mathbb{N}$,

3. $\mathcal{U} = C^{m_2}(\bar{D}')$ *and* $m_2 \in \mathbb{N}_0$.

We first show that neural operators are dense in the continuous operators $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ in the topology of uniform convergence on compacta. The proof proceeds by making three main approximations which are schematically shown in Figure 16. First, inputs are mapped to a finite-dimensional representation through a set of appropriate linear functionals on $\mathcal{A}$ denoted by $F : \mathcal{A} \to \mathbb{R}^J$. We show in Lemmas 21 and 23 that, when $\mathcal{A}$ satisfies Assumption 9, elements of $\mathcal{A}^*$ can be approximated by integration against smooth functions. This generalizes the idea from (Chen and Chen, 1995) where functionals on $C(\bar{D})$ are approximated by a weighted sum of Dirac measures. We then show in Lemma 25 that, by lifting the dimension, this representation can be approximated by a single element of IO. Second, the representation is non-linearly mapped to a new representation by a continuous function $\psi : \mathbb{R}^J \to \mathbb{R}^{J'}$ which finite-dimensionalizes the action of $\mathcal{G}^\dagger$. We show, in Lemma 28, that this map can be approximated by a neural operator by reducing the architecture to that of a standard neural network. Third, the new representation is used as the coefficients of an expansion onto representers of $\mathcal{U}$, the map denoted $G : \mathbb{R}^{J'} \to \mathcal{U}$, which we show can be

approximated by a single IO layer in Lemma 27 using density results for continuous functions. The structure of the overall approximation is similar to (Bhattacharya et al., 2020) but generalizes the ideas from working on Hilbert spaces to the spaces in Assumptions 9 and 10. Statements and proofs of the lemmas used in the theorems are given in the appendices.

Theorem 11 *Let Assumptions 9 and 10 hold and suppose $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ is continuous. Let $\sigma_1 \in \mathsf{A}_0^\mathsf{L}$, $\sigma_2 \in \mathsf{A}_0$, and $\sigma_3 \in \mathsf{A}_{m_2}$. Then for any compact set $K \subset \mathcal{A}$ and $0 < \epsilon \leq 1$, there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that*

$$\sup_{a \in K} \|\mathcal{G}^\dagger(a) - \mathcal{G}(a)\|_\mathcal{U} \leq \epsilon.$$

Furthermore, if $\mathcal{U}$ is a Hilbert space and $\sigma_1 \in \mathsf{BA}$ and, for some $M > 0$, we have that $\|\mathcal{G}^\dagger(a)\|_\mathcal{U} \leq M$ for all $a \in \mathcal{A}$ then $\mathcal{G}$ can be chosen so that

$$\|\mathcal{G}(a)\|_\mathcal{U} \leq 4M, \qquad \forall a \in \mathcal{A}.$$

The proof is provided in appendix F In the following theorem, we extend this result to the case $\mathcal{A} = C^{m_1}(\bar{D})$, showing density of the m_1-th order neural operators.

Theorem 12 *Let $D \subset \mathbb{R}^d$ be a Lipschitz domain, $m_1 \in \mathbb{N}$, define $\mathcal{A} := C^{m_1}(\bar{D})$, suppose Assumption 10 holds and assume that $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ is continuous. Let $\sigma_1 \in \mathsf{A}_0^\mathsf{L}$, $\sigma_2 \in \mathsf{A}_0$, and $\sigma_3 \in \mathsf{A}_{m_2}$. Then for any compact set $K \subset \mathcal{A}$ and $0 < \epsilon \leq 1$, there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N^{m_1}(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that*

$$\sup_{a \in K} \|\mathcal{G}^\dagger(a) - \mathcal{G}(a)\|_\mathcal{U} \leq \epsilon.$$

Furthermore, if $\mathcal{U}$ is a Hilbert space and $\sigma_1 \in \mathsf{BA}$ and, for some $M > 0$, we have that $\|\mathcal{G}^\dagger(a)\|_\mathcal{U} \leq M$ for all $a \in \mathcal{A}$ then $\mathcal{G}$ can be chosen so that

$$\|\mathcal{G}(a)\|_\mathcal{U} \leq 4M, \qquad \forall a \in \mathcal{A}.$$

Proof The proof follows as in Theorem 11, replacing the use of Lemma 32 with Lemma 33. ∎

With these results in hand, we show density of neural operators in the space $L_\mu^2(\mathcal{A}; \mathcal{U})$ where μ is a probability measure and $\mathcal{U}$ is a separable Hilbert space. The Hilbertian structure of $\mathcal{U}$ allows us to uniformly control the norm of the approximation due to the isomorphism with ℓ_2 as shown in Theorem 11. It remains an interesting future direction to obtain similar results for Banach spaces. The proof follows the ideas in (Lanthaler et al., 2021) where similar results are obtained for Deep-ONet(s) on $L^2(D)$ by using Lusin's theorem to restrict the approximation to a large enough compact set and exploit the decay of μ outside it. Bhattacharya et al. (2020) also employ a similar approach but explicitly constructs the necessary compact set after finite-dimensionalizing.

Theorem 13 *Let $D' \subset \mathbb{R}^{d'}$ be a Lipschitz domain, $m_2 \in \mathbb{N}_0$, and suppose Assumption 9 holds. Let μ be a probability measure on $\mathcal{A}$ and suppose $\mathcal{G}^\dagger : \mathcal{A} \to H^{m_2}(D)$ is μ-measurable and $\mathcal{G}^\dagger \in L_\mu^2(\mathcal{A}; H^{m_2}(D))$. Let $\sigma_1 \in \mathsf{A}_0^\mathsf{L} \cap \mathsf{BA}$, $\sigma_2 \in \mathsf{A}_0$, and $\sigma_3 \in \mathsf{A}_{m_2}$. Then for any $0 < \epsilon \leq 1$, there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that*

$$\|\mathcal{G}^\dagger - \mathcal{G}\|_{L_\mu^2(\mathcal{A}; H^{m_2}(D))} \leq \epsilon.$$

The proof is provided in appendix G. In the following we extend this result to the case $\mathcal{A} = C^{m_1}(D)$ using the m_1-th order neural operators.

Theorem 14 *Let $D \subset \mathbb{R}^d$ be a Lipschitz domain, $m_1 \in \mathbb{N}$, define $\mathcal{A} := C^{m_1}(D)$ and suppose Assumption 10 holds. Let μ be a probability measure on $C^{m_1}(D)$ and let $\mathcal{G}^\dagger : C^{m_1}(D) \to \mathcal{U}$ be μ-measurable and suppose $\mathcal{G}^\dagger \in L^2_\mu(C^{m_1}(D); \mathcal{U})$. Let $\sigma_1 \in \mathsf{A}_0^\mathsf{L} \cap \mathsf{BA}$, $\sigma_2 \in \mathsf{A}_0$, and $\sigma_3 \in \mathsf{A}_{m_2}$. Then for any $0 < \epsilon \leq 1$, there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N^{m_1}(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that*

$$\|\mathcal{G}^\dagger - \mathcal{G}\|_{L^2_\mu(C^{m_1}(D); \mathcal{U})} \leq \epsilon.$$

Proof The proof follows as in Theorem 13 by replacing the use of Theorem 11 with Theorem 12. ∎

10. Conclusions

We have introduced the concept of Neural Operator, the goal being to construct a neural network architecture adapted to the problem of mapping elements of one function space into elements of another function space. The network is comprised of four steps which, in turn, (i) extract features from the input functions, (ii) iterate a recurrent neural network on feature space, defined through composition of a sigmoid function and a nonlocal operator, and (iii) a final mapping from feature space into the output function.

We have studied four nonlocal operators in step (iii), one based on graph kernel networks, one based on the low-rank decomposition, one based on the multi-level graph structure, and the last one based on convolution in Fourier space. The designed network architectures are constructed to be mesh-free and our numerical experiments demonstrate that they have the desired property of being able to train and generalize on different meshes. This is because the networks learn the mapping between infinite-dimensional function spaces, which can then be shared with approximations at different levels of discretization. A further advantage of the integral operator approach is that data may be incorporated on unstructured grids, using the Nyström approximation; these methods, however, are quadratic in the number of discretization points; we describe variants on this methodology, using low rank and multiscale ideas, to reduce this complexity. On the other hand the Fourier approach leads directly to fast methods, linear-log linear in the number of discretization points, provided structured grids are used. We demonstrate that our methods can achieve competitive performance with other mesh-free approaches developed in the numerical analysis community. Specifically, the Fourier neural operator achieves the best numerical performance among our experiments, potentially due to the smoothness of the solution function and the underlying uniform grids. The methods developed in the numerical analysis community are less flexible than the approach we introduce here, relying heavily on the structure of an underlying PDE mapping input to output; our method is entirely data-driven.

10.1 Future Directions

We foresee three main directions in which this work will develop: firstly as a method to speed-up scientific computing tasks which involve repeated evaluation of a mapping between spaces of

functions, following the example of the Bayesian inverse problem 7.3.4, or when the underlying model is unknown as in computer vision or robotics; and secondly the development of more advanced methodologies beyond the four approximation schemes presented in Section 4 that are more efficient or better in specific situations; thirdly, the development of an underpinning theory which captures the expressive power, and approximation error properties, of the proposed neural network, following Section 9, and quantifies the computational complexity required to achieve given error.

10.1.1 NEW APPLICATIONS

The proposed neural operator is a blackbox surrogate model for function-to-function mappings. It naturally fits into solving PDEs for physics and engineering problems. In the paper we mainly studied three partial differential equations: Darcy Flow, Burgers' equation, and Navier-Stokes equation, which cover a broad range of scenarios. Due to its blackbox structure, the neural operator is easily applied on other problems. We foresee applications on more challenging turbulent flows, such as those arising in subgrid models with in climate GCMs, high contrast media in geological models generalizing the Darcy model, and general physics simulation for games and visual effects. The operator setting leads to an efficient and accurate representation, and the resolution-invariant properties make it possible to training and a smaller resolution dataset, and be evaluated on arbitrarily large resolution.

The operator learning setting is not restricted to scientific computing. For example, in computer vision, images can naturally be viewed as real-valued functions on 2D domains and videos simply add a temporal structure. Our approach is therefore a natural choice for problems in computer vision where invariance to discretization is crucial. We leave this as an interesting future direction.

10.1.2 NEW METHODOLOGIES

Despite their excellent performance, there is still room for improvement upon the current methodologies. For example, the full $O(J^2)$ integration method still outperforms the FNO by about 40%, albeit at greater cost. It is of potential interest to develop more advanced integration techniques or approximation schemes that follows the neural operator framework. For example, one can use adaptive graph or probability estimation in the Nyström approximation. It is also possible to use other basis than the Fourier basis such as the PCA basis and the Chebyshev basis.

Another direction for new methodologies is to combine the neural operator in other settings. The current problem is set as a supervised learning problem. Instead, one can combine the neural operator with solvers (Pathak et al., 2020; Um et al., 2020b), augmenting and correcting the solvers to get faster and more accurate approximation. Similarly, one can combine operator learning with physics constraints (Wang et al., 2021; Li et al., 2021).

10.1.3 THEORY

In this work, we develop a universal approximation theory (Section 9) for neural operators. As in the work of Lu et al. (2019) studying universal approximation for DeepONet, we use linear approximation techniques. The power of non-linear approximation (DeVore, 1998), which is likely intrinsic to the success of neural operators in some settings, is still less studied, as discussed in Section 5.1; we note that DeepOnet is intrinsically limited by linear approximation properties. For functions between Euclidean spaces, we clearly know, by combining two layers of linear functions with one layer of non-linear activation function, the neural network can approximate arbitrary con-

tinuous functions, and that deep neural networks can be exponentially more expressive compared to shallow networks (Poole et al., 2016). However issues are less clear when it comes to the choice of architecture and the scaling of the number of parameters within neural operators between Banach spaces. The approximation theory of operators is much more complex and challenging compared to that of functions over Euclidean spaces. It is important to study the class of neural operators with respect to their architecture – what spaces the true solution operators lie in, and which classes of PDEs the neural operator approximate efficiently. We leave these as exciting, but open, research directions.

Acknowledgements

References

J. Aaronson. *An Introduction to Infinite Ergodic Theory*. Mathematical surveys and monographs. American Mathematical Society, 1997. ISBN 9780821804940.

R. A. Adams and J. J. Fournier. *Sobolev Spaces*. Elsevier Science, 2003.

Jonas Adler and Ozan Oktem. Solving ill-posed inverse problems using iterative deep neural networks. *Inverse Problems*, nov 2017. doi: 10.1088/1361-6420/aa9581. URL `https://doi.org/10.1088%2F1361-6420%2Faa9581`.

Fernando Albiac and Nigel J. Kalton. *Topics in Banach space theory*. Graduate Texts in Mathematics. Springer, 1 edition, 2006.

Ferran Alet, Adarsh Keshav Jeewajee, Maria Bauza Villalonga, Alberto Rodriguez, Tomas Lozano-Perez, and Leslie Kaelbling. Graph element networks: adaptive, structured computation and memory. In *36th International Conference on Machine Learning*. PMLR, 2019. URL `http://proceedings.mlr.press/v97/alet19a.html`.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Francis Bach. Sharp analysis of low-rank kernel matrix approximations. In *Conference on Learning Theory*, pages 185–209, 2013.

Leah Bar and Nir Sochen. Unsupervised deep learning algorithm for PDE-based forward and inverse problems. *arXiv preprint arXiv:1904.05417*, 2019.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al.

Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Christian Beck, Sebastian Becker, Philipp Grohs, Nor Jaafari, and Arnulf Jentzen. Solving the kolmogorov pde by means of deep learning. *Journal of Scientific Computing*, 88(3), 2021.

Serge Belongie, Charless Fowlkes, Fan Chung, and Jitendra Malik. Spectral partitioning with indefinite kernels using the nyström extension. In *European conference on computer vision*. Springer, 2002.

Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, pages 1–21, 2019.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric PDEs. *arXiv preprint arXiv:2005.03180*, 2020.

V. I. Bogachev. *Measure Theory*, volume 2. Springer-Verlag Berlin Heidelberg, 2007.

Andrea Bonito, Albert Cohen, Ronald DeVore, Diane Guignard, Peter Jantsch, and Guergana Petrova. Nonlinear methods for model reduction. *arXiv preprint arXiv:2005.02565*, 2020.

Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. Hierarchical matrices. *Lecture notes*, 21: 2003, 2003.

George EP Box. Science and statistics. *Journal of the American Statistical Association*, 71(356): 791–799, 1976.

John P Boyd. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Alexander Brudnyi and Yuri Brudnyi. *Methods of Geometric Analysis in Extension and Trace Problems*, volume 1. Birkhäuser Basel, 2012.

Oscar P Bruno, Youngae Han, and Matthew M Pohlman. Accurate, high-order representation of complex three-dimensional surfaces via fourier continuation analysis. *Journal of computational Physics*, 227(2):1094–1125, 2007.

Gary J. Chandler and Rich R. Kerswell. Invariant recurrent solutions embedded in a turbulent two-dimensional kolmogorov flow. *Journal of Fluid Mechanics*, 722:554–595, 2013.

Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. Graph networks as a universal machine learning framework for molecules and crystals. *Chemistry of Materials*, 31(9): 3564–3572, 2019.

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.

Z. Ciesielski and J. Domsta. Construction of an orthonormal basis in cm(id) and wmp(id). *Studia Mathematica*, 41:211–224, 1972.

Albert Cohen and Ronald DeVore. Approximation of high-dimensional parametric PDEs. *Acta Numerica*, 2015. doi: 10.1017/S0962492915000033.

Albert Cohen, Ronald Devore, Guergana Petrova, and Przemyslaw Wojtaszczyk. Optimal stable nonlinear approximation. *arXiv preprint arXiv:2009.09907*, 2020.

J. B. Conway. *A Course in Functional Analysis*. Springer-Verlag New York, 2007.

S. L. Cotter, G. O. Roberts, A. M. Stuart, and D. White. Mcmc methods for functions: Modifying old algorithms to make them faster. *Statistical Science*, 28(3):424–446, Aug 2013. ISSN 0883-4237. doi: 10.1214/13-sts421. URL http://dx.doi.org/10.1214/13-STS421.

Simon L Cotter, Massoumeh Dashti, James Cooper Robinson, and Andrew M Stuart. Bayesian inverse problems for functions and applications to fluid mechanics. *Inverse problems*, 25(11): 115008, 2009.

Andreas Damianou and Neil Lawrence. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013.

Maarten De Hoop, Daniel Zhengyu Huang, Elizabeth Qian, and Andrew M Stuart. The cost-accuracy trade-off in operator learning with neural networks. *Journal of Machine Learning, to appear; arXiv preprint arXiv:2203.13181*, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

Ronald A. DeVore. *Chapter 3: The Theoretical Foundation of Reduced Basis Methods*. 2014. doi: 10.1137/1.9781611974829.ch3.

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.

R. Dudley and Rimas Norvaisa. *Concrete Functional Calculus*, volume 149. 01 2011. ISBN 978-1-4419-6949-1.

R.M. Dudley and R. Norvaiša. *Concrete Functional Calculus*. Springer Monographs in Mathematics. Springer New York, 2010.

J. Dugundji. An extension of tietze's theorem. *Pacific Journal of Mathematics*, 1(3):353 – 367, 1951.

Matthew M Dunlop, Mark A Girolami, Andrew M Stuart, and Aretha L Teckentrup. How deep are deep gaussian processes? *The Journal of Machine Learning Research*, 19(1):2100–2145, 2018.

W E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

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

Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 3 2018. ISSN 2194-6701. doi: 10.1007/s40304-018-0127-z.

Yuwei Fan, Cindy Orozco Bohorquez, and Lexing Ying. Bcr-net: A neural network based on the nonstandard wavelet form. *Journal of Computational Physics*, 384:1–15, 2019a.

Yuwei Fan, Jordi Feliu-Faba, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. A multiscale neural network based on hierarchical nested bases. *Research in the Mathematical Sciences*, 6(2): 21, 2019b.

Yuwei Fan, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. A multiscale neural network based on hierarchical matrices. *Multiscale Modeling & Simulation*, 17(4):1189–1213, 2019c.

Charles Fefferman. Cm extension by linear operators. *Annals of Mathematics*, 166:779–835, 2007.

Stefania Fresca and Andrea Manzoni. Pod-dl-rom: Enhancing deep learning-based reduced order models for nonlinear parametrized pdes by proper orthogonal decomposition. *Computer Methods in Applied Mechanics and Engineering*, 388:114–181, 2022.

Jacob R Gardner, Geoff Pleiss, Ruihan Wu, Kilian Q Weinberger, and Andrew Gordon Wilson. Product kernel interpolation for scalable gaussian processes. *arXiv preprint arXiv:1802.08903*, 2018.

Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep Convolutional Networks as shallow Gaussian Processes. *arXiv e-prints*, art. arXiv:1808.05587, Aug 2018.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Amir Globerson and Roi Livni. Learning infinite-layer networks: Beyond the kernel trick. *CoRR*, abs/1606.05316, 2016. URL http://arxiv.org/abs/1606.05316.

Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid PDE solvers. In *International Conference on Machine Learning*, pages 2415–2423. PMLR, 2019.

Leslie Greengard and Vladimir Rokhlin. A new version of the fast multipole method for the laplace equation in three dimensions. *Acta numerica*, 6:229–269, 1997.

A Grothendieck. *Produits tensoriels topologiques et espaces nucléaires*, volume 16. American Mathematical Society Providence, 1955.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Morton E Gurtin. *An introduction to continuum mechanics*. Academic press, 1982.

William H. Guss. Deep Function Machines: Generalized Neural Networks for Topological Layer Expression. *arXiv e-prints*, art. arXiv:1612.04799, Dec 2016.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

Juncai He and Jinchao Xu. Mgnet: A unified framework of multigrid and convolutional neural network. *Science china mathematics*, 62(7):1331–1354, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

L Herrmann, Ch Schwab, and J Zech. Deep relu neural network expression rates for data-to-qoi maps in bayesian PDE inversion. 2020.

Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Chiyu Max Jiang, Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A Tchelepi, Philip Marcus, Anima Anandkumar, et al. Meshfreeflownet: A physics-constrained deep continuous space-time super-resolution framework. *arXiv preprint arXiv:2005.01463*, 2020.

Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

Karthik Kashinath, Philip Marcus, et al. Enforcing physical constraints in cnns through differentiable PDE layer. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

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

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

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Risi Kondor, Nedelina Teneva, and Vikas Garg. Multiresolution matrix factorization. In *International Conference on Machine Learning*, pages 1620–1628, 2014.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for Fourier Neural Operators. *arXiv preprint arXiv:2107.07562*, 2021.

Robert H. Kraichnan. Inertial ranges in two-dimensional turbulence. *The Physics of Fluids*, 10(7): 1417–1423, 1967.

Brian Kulis, Mátyás Sustik, and Inderjit Dhillon. Learning low-rank kernel matrices. In *Proceedings of the 23rd international conference on Machine learning*, pages 505–512, 2006.

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

Liang Lan, Kai Zhang, Hancheng Ge, Wei Cheng, Jun Liu, Andreas Rauber, Xiao-Li Li, Jun Wang, and Hongyuan Zha. Low-rank decomposition meets kernel learning: A generalized nyström method. *Artificial Intelligence*, 250:1–15, 2017.

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.

G. Leoni. *A First Course in Sobolev Spaces*. Graduate studies in mathematics. American Mathematical Soc., 2009.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 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, 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.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 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, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021a.

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*, 2021b.

Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts, 2013.

Alexander G. de G. Matthews, Mark Rowland, Jiri Hron, Richard E. Turner, and Zoubin Ghahramani. Gaussian Process Behaviour in Wide Deep Neural Networks. Apr 2018.

Luis Mingo, Levon Aslanyan, Juan Castellanos, Miguel Diaz, and Vladimir Riazanov. Fourier neural networks: An approach with sinusoidal activation functions. 2004.

Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, 1996. ISBN 0387947248.

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

Evert J Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*, 1930.

Thomas O'Leary-Roseberry, Umberto Villa, Peng Chen, and Omar Ghattas. Derivative-informed projected neural networks for high-dimensional parametric maps governed by pdes. *arXiv preprint arXiv:2011.15110*, 2020.

Joost A.A. Opschoor, Christoph Schwab, and Jakob Zech. Deep learning in high dimension: Relu network expression rates for bayesian PDE inversion. *SAM Research Report*, 2020-47, 2020.

Shaowu Pan and Karthik Duraisamy. Physics-informed probabilistic learning of linear embeddings of nonlinear dynamics with guaranteed stability. *SIAM Journal on Applied Dynamical Systems*, 19(1):480–509, 2020.

Jaideep Pathak, Mustafa Mustafa, Karthik Kashinath, Emmanuel Motheau, Thorsten Kurth, and Marcus Day. Using machine learning to augment coarse-grid computational fluid dynamics simulations, 2020.

Aleksander Pełczyński and Michał Wojciechowski. Contribution to the isomorphic classification of sobolev spaces lpk(omega). *Recent Progress in Functional Analysis*, 189:133–142, 2001.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks, 2020.

A. Pinkus. *N-Widths in Approximation Theory*. Springer-Verlag Berlin Heidelberg, 1985.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8: 143–195, 1999.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. *Advances in neural information processing systems*, 29:3360–3368, 2016.

Joaquin Quiñonero Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *J. Mach. Learn. Res.*, 6:1939–1959, 2005.

Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561. IEEE, 2008.

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

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Nicolas Le Roux and Yoshua Bengio. Continuous neural networks. In Marina Meila and Xiaotong Shen, editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, 2007.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

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

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

Vincent Sitzmann, Julien NP Martel, Alexander W Bergman, David B Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *arXiv preprint arXiv:2006.09661*, 2020.

Jonathan D Smith, Kamyar Azizzadenesheli, and Zachary E Ross. Eikonet: Solving the eikonal equation with deep neural networks. *arXiv preprint arXiv:2004.00361*, 2020.

Elias M. Stein. *Singular Integrals and Differentiability Properties of Functions*. Princeton University Press, 1970.

A. M. Stuart. Inverse problems: A bayesian perspective. *Acta Numerica*, 19:451–559, 2010.

Lloyd N Trefethen. *Spectral methods in MATLAB*, volume 10. Siam, 2000.

Nicolas Garcia Trillos and Dejan Slepčev. A variational approach to the consistency of spectral clustering. *Applied and Computational Harmonic Analysis*, 45(2):239–281, 2018.

Nicolás García Trillos, Moritz Gerlach, Matthias Hein, and Dejan Slepčev. Error estimates for spectral convergence of the graph laplacian on random geometric graphs toward the laplace–beltrami operator. *Foundations of Computational Mathematics*, 20(4):827–887, 2020.

Kiwon Um, Philipp Holl, Robert Brand, Nils Thuerey, et al. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. *arXiv preprint arXiv:2007.00016*, 2020a.

Kiwon Um, Raymond, Fei, Philipp Holl, Robert Brand, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers, 2020b.

Benjamin Ummenhofer, Lukas Prantl, Nils Thürey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2020.

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. 2017.

Ulrike Von Luxburg, Mikhail Belkin, and Olivier Bousquet. Consistency of spectral clustering. *The Annals of Statistics*, pages 555–586, 2008.

Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1457–1466, 2020.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *arXiv preprint arXiv:2103.10974*, 2021.

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

Hassler Whitney. Functions differentiable on the boundaries of regions. *Annals of Mathematics*, 35 (3):482–485, 1934.

Christopher K. I. Williams. Computing with infinite networks. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, Cambridge, MA, USA, 1996. MIT Press.

Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. Long-short transformer: Efficient transformers for language and vision. In *Advances in Neural Information Processing Systems*, 2021.

Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.04.018. URL http://www.sciencedirect.com/science/article/pii/S0021999118302341.

Appendix A.

Notation	Meaning
Operator Learning	
$D \subset \mathbb{R}^d$	The spatial domain for the PDE.
$x \in D$	Points in the the spatial domain.
$a \in \mathcal{A} = (D; \mathbb{R}^{d_a})$	The input functions (coefficients, boundaries, and/or initial conditions).
$u \in \mathcal{U} = (D; \mathbb{R}^{d_u})$	The target solution functions.
D_j	The discretization of (a_j, u_j).
$\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$	The operator mapping the coefficients to the solutions.
μ	A probability measure where a_j sampled from.
Neural Operator	
$v(x) \in \mathbb{R}^{d_v}$	The neural network representation of $u(x)$
d_a	Dimension of the input $a(x)$.
d_u	Dimension of the output $u(x)$.
d_v	The dimension of the representation $v(x)$.
$t = 0, \ldots, T$	The layer (iteration) in the neural operator .
$\mathcal{P}, \mathcal{Q}$	The pointwise linear transformation $\mathcal{P} : a(x) \mapsto v_0(x)$ and $\mathcal{Q} : v_T(x) \mapsto u(x)$.
$\mathcal{K}$	The integral operator in the iterative update $v_t \mapsto v_{t+1}$,
$\kappa : \mathbb{R}^{2(d+1)} \to \mathbb{R}^{d_v \times d_v}$	The kernel maps $(x, y, a(x), a(y))$ to a $d_v \times d_v$ matrix
$K \in \mathbb{R}^{n \times n \times d_v \times d_v}$	The kernel matrix with $K_{xy} = \kappa(x, y)$.
$W \in \mathbb{R}^{d_v \times d_v}$	The pointwise linear transformation used as the bias term in the iterative update.
σ	The activation function.

Table 10: Table of notations: operator learning and neural operators

In the paper, we will use lowercase letters such as v, u to represent vectors and functions; uppercase letters such as W, K to represent matrices or discretized transformations; and calligraphic letters such as $\mathcal{G}, \mathcal{F}$ to represent operators.

We write $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Furthermore, we denote by $| \cdot |_p$ the p-norm on any Euclidean space. We say $\mathcal{X}$ is a Banach space if it is a Banach space over the real field $\mathbb{R}$. We denote by $\| \cdot \|_{\mathcal{X}}$ its norm and by $\mathcal{X}^*$ its topological (continuous) dual. In particular, $\mathcal{X}^*$ is the Banach space consisting of all continuous linear functionals $f : \mathcal{X} \to \mathbb{R}$ with the operator norm

$$\|f\|_{\mathcal{X}^*} = \sup_{\substack{x \in \mathcal{X} \\ \|x\|_{\mathcal{X}} = 1}} |f(x)| < \infty.$$

For any Banach space $\mathcal{Y}$, we denote by $\mathcal{L}(\mathcal{X}; \mathcal{Y})$ the Banach space of continuous linear maps $T : \mathcal{X} \to \mathcal{Y}$ with the operator norm

$$\|T\|_{\mathcal{X} \to \mathcal{Y}} = \sup_{\substack{x \in \mathcal{X} \\ \|x\|_{\mathcal{X}} = 1}} \|Tx\|_{\mathcal{Y}} < \infty.$$

We will abuse notation and write $\| \cdot \|$ for any operator norm when there is no ambiguity about the spaces in question.

Let $d \in \mathbb{N}$. We say that $D \subset \mathbb{R}^d$ is a *domain* if it is a bounded and connected open set that is topologically regular i.e. $\text{int}(\bar{D}) = D$. Note that, in the case $d = 1$, a domain is any bounded, open interval. For $d \geq 2$, we say D is a *Lipschitz domain* if ∂D can be locally represented as the graph of a Lipschitz continuous function defined on an open ball of $\mathbb{R}^{d-1}$. If $d = 1$, we will call any domain a Lipschitz domain. For any multi-index $\alpha \in \mathbb{N}_0^d$, we write $\partial^\alpha f$ for the α-th weak partial derivative of f when it exists.

Let $D \subset \mathbb{R}^d$ be a domain. For any $m \in \mathbb{N}_0$, we define the following spaces

$$C(D) = \{f : D \to \mathbb{R} : f \text{ is continuous}\},$$

$$C^m(D) = \{f : D \to \mathbb{R} : \partial^\alpha f \in C^{m-|\alpha|_1}(D) \; \forall \, 0 \leq |\alpha|_1 \leq m\},$$

$$C_{\text{b}}^m(D) = \left\{ f \in C^m(D) : \max_{0 \leq |\alpha|_1 \leq m} \sup_{x \in D} |\partial^\alpha f(x)| < \infty \right\},$$

$$C^m(\bar{D}) = \{f \in C_{\text{b}}^m(D) : \partial^\alpha f \text{ is uniformly continuous} \; \forall \, 0 \leq |\alpha|_1 \leq m\}$$

and make the equivalent definitions when D is replaced with $\mathbb{R}^d$. Note that any function in $C^m(\bar{D})$ has a unique, bounded, continuous extension from D to $\bar{D}$ and is hence uniquely defined on ∂D. We will work with this extension without further notice. We remark that when D is a Lipschitz domain, the following definition for $C^m(\bar{D})$ is equivalent

$$C^m(\bar{D}) = \{f : \bar{D} \to \mathbb{R} : \exists F \in C^m(\mathbb{R}^d) \text{ such that } f \equiv F|_{\bar{D}}\},$$

see Whitney (1934); Brudnyi and Brudnyi (2012). We define $C^\infty(D) = \bigcap_{m=0}^\infty C^m(D)$ and, similarly, $C_{\text{b}}^\infty(D)$ and $C^\infty(\bar{D})$. We further define

$$C_c^\infty(D) = \{f \in C^\infty(D) : \text{supp}(f) \subset D \text{ is compact}\}$$

and, again, note that all definitions hold analogously for $\mathbb{R}^d$. We denote by $\|\cdot\|_{C^m} : C_{\text{b}}^m(D) \to \mathbb{R}_{\geq 0}$ the norm

$$\|f\|_{C^m} = \max_{0 \leq |\alpha|_1 \leq m} \sup_{x \in D} |\partial^\alpha f(x)|$$

which makes $C_{\text{b}}^m(D)$ (also with $D = \mathbb{R}^d$) and $C^m(\bar{D})$ Banach spaces. For any $n \in \mathbb{N}$, we write $C(D; \mathbb{R}^n)$ for the n-fold Cartesian product of $C(D)$ and similarly for all other spaces we have defined or will define subsequently. We will continue to write $\|\cdot\|_{C^m}$ for the norm on $C_{\text{b}}^m(D; \mathbb{R}^n)$ and $C^m(\bar{D}; \mathbb{R}^n)$ defined as

$$\|f\|_{C^m} = \max_{j \in \{1, \dots, n\}} \|f_j\|_{C^m}.$$

For any $m \in \mathbb{N}$ and $1 \leq p \leq \infty$, we use the notation $W^{m,p}(D)$ for the standard L^p-type Sobolev space with m derivatives; we refer the reader to Adams and Fournier (2003) for a formal definition. Furthermore, we, at times, use the notation $W^{0,p}(D) = L^p(D)$ and $W^{m,2}(D) = H^m(D)$. Since we use the standard definitions of Sobolev spaces that can be found in any reference on the subject, we do not give the specifics here.

Appendix B.

In this section we gather various results on the approximation property of Banach spaces. The main results are Lemma 22 which states that if two Banach spaces have the approximation property then continuous maps between them can be approximated in a finite-dimensional manner, and Lemma 26 which states the spaces in Assumptions 9 and 10 have the approximation property.

Definition 15 *A Banach space $\mathcal{X}$ has a Schauder basis if there exist some $\{\varphi_j\}_{j=1}^{\infty} \subset \mathcal{X}$ and $\{c_j\}_{j=1}^{\infty} \subset \mathcal{X}^*$ such that*

1. *$c_j(\varphi_k) = \delta_{jk}$ for any $j, k \in \mathbb{N}$,*

2. *$\lim_{n \to \infty} \|x - \sum_{j=1}^{n} c_j(x)\varphi_j\|_{\mathcal{X}} = 0$ for all $x \in \mathcal{X}$.*

We remark that definition 15 is equivalent to the following. The elements $\{\varphi_j\}_{j=1}^{\infty} \subset \mathcal{X}$ are called a *Schauder basis* for $\mathcal{X}$ if, for each $x \in \mathcal{X}$, there exists a unique sequence $\{\alpha_j\}_{j=1}^{\infty} \subset \mathbb{R}$ such that

$$\lim_{n \to \infty} \|x - \sum_{j=1}^{n} \alpha_j \varphi_j\|_{\mathcal{X}} = 0.$$

For the equivalence, see, for example (Albiac and Kalton, 2006, Theorem 1.1.3). Throughout this paper we will simply write the term *basis* to mean Schauder basis. Furthermore, we note that if $\{\varphi\}_{j=1}^{\infty}$ is a basis then so is $\{\varphi_j/\|\varphi\|_{\mathcal{X}}\}_{j=1}^{\infty}$, so we will assume that any basis we use is normalized.

Definition 16 *Let $\mathcal{X}$ be a Banach space and $U \in \mathcal{L}(\mathcal{X}; \mathcal{X})$. U is called a finite rank operator if $U(\mathcal{X}) \subseteq \mathcal{X}$ is finite dimensional.*

By noting that any finite dimensional subspace has a basis, we may equivalently define a finite rank operator $U \in \mathcal{L}(\mathcal{X}; \mathcal{X})$ to be one such that there exists a number $n \in \mathbb{N}$ and some $\{\varphi_j\}_{j=1}^{n} \subset \mathcal{X}$ and $\{c_j\}_{j=1}^{n} \subset \mathcal{X}^*$ such that

$$Ux = \sum_{j=1}^{n} c_j(x)\varphi_j, \qquad \forall x \in \mathcal{X}.$$

Definition 17 *A Banach space $\mathcal{X}$ is said to have the approximation property (*AP*) if, for any compact set $K \subset \mathcal{X}$ and $\epsilon > 0$, there exists a finite rank operator $U : \mathcal{X} \to \mathcal{X}$ such that*

$$\|x - Ux\|_{\mathcal{X}} \leq \epsilon, \qquad \forall x \in K.$$

We now state and prove some well-known results about the relationship between basis and the AP. We were unable to find the statements of the following lemmas in the form given here in the literature and therefore we provide full proofs.

Lemma 18 *Let $\mathcal{X}$ be a Banach space with a basis. Then $\mathcal{X}$ has the* AP.

Proof Let $\{c_j\}_{j=1}^{\infty} \subset \mathcal{X}^*$ and $\{\varphi_j\}_{j=1}^{\infty} \subset \mathcal{X}$ be a basis for $\mathcal{X}$. Note that there exists a constant $C > 0$ such that, for any $x \in \mathcal{X}$ and $n \in \mathbb{N}$,

$$\|\sum_{j=1}^{n} c_j(x)\varphi_j\|_{\mathcal{X}} \leq \sup_{J \in \mathbb{N}} \|\sum_{j=1}^{J} c_j(x)\varphi_j\|_{\mathcal{X}} \leq C\|x\|_{\mathcal{X}},$$

see, for example (Albiac and Kalton, 2006, Remark 1.1.6). Assume, without loss of generality, that $C \geq 1$. Let $K \subset \mathcal{X}$ be compact and $\epsilon > 0$. Since K is compact, we can find a number

$n = n(\epsilon, C) \in \mathbb{N}$ and elements $y_1, \ldots, y_n \in K$ such that for any $x \in K$ there exists a number $l \in \{1, \ldots, n\}$ with the property that

$$\|x - y_l\|_{\mathcal{X}} \le \frac{\epsilon}{3C}.$$

We can then find a number $J = J(\epsilon, n) \in \mathbb{N}$ such that

$$\max_{j \in \{1, \ldots, n\}} \|y_j - \sum_{k=1}^{J} c_k(y_j)\varphi_k\|_{\mathcal{X}} \le \frac{\epsilon}{3}.$$

Define the finite rank operator $U : \mathcal{X} \to \mathcal{X}$ by

$$Ux = \sum_{j=1}^{J} c_j(x)\varphi_j, \qquad \forall x \in \mathcal{X}.$$

Triangle inequality implies that, for any $x \in K$,

$$
\begin{aligned}
\|x - U(x)\|_{\mathcal{X}} &\le \|x - y_l\|_{\mathcal{X}} + \|y_l - U(y_l)\|_{\mathcal{X}} + \|U(y_l) - U(x)\|_{\mathcal{X}} \\
&\le \frac{2\epsilon}{3} + \|\sum_{j=1}^{J} (c_j(y_l) - c_j(x))\varphi_j\|_{\mathcal{X}} \\
&\le \frac{2\epsilon}{3} + C\|y_l - x\|_{\mathcal{X}} \\
&\le \epsilon
\end{aligned}
$$

as desired. $\blacksquare$

Lemma 19 *Let $\mathcal{X}$ be a Banach space with a basis and $\mathcal{Y}$ be any Banach space. Suppose there exists a continuous linear bijection $T : \mathcal{X} \to \mathcal{Y}$. Then $\mathcal{Y}$ has a basis.*

Proof Let $y \in \mathcal{Y}$ and $\epsilon > 0$. Since T is a bijection, there exists an element $x \in \mathcal{X}$ so that $Tx = y$ and $T^{-1}y = x$. Since $\mathcal{X}$ has a basis, we can find $\{\varphi_j\}_{j=1}^{\infty} \subset \mathcal{X}$ and $\{c_j\}_{j=1}^{\infty} \subset \mathcal{X}^*$ and a number $n = n(\epsilon, \|T\|) \in \mathbb{N}$ such that

$$\|x - \sum_{j=1}^{n} c_j(x)\varphi_j\|_{\mathcal{X}} \le \frac{\epsilon}{\|T\|}.$$

Note that

$$\|y - \sum_{j=1}^{n} c_j(T^{-1}y)T\varphi_j\|_{\mathcal{Y}} = \|Tx - T\sum_{j=1}^{n} c_j(x)\varphi_j\| \le \|T\|\|x - \sum_{j=1}^{n} c_j(x)\varphi_j\|_{\mathcal{X}} \le \epsilon$$

hence $\{T\varphi_j\}_{j=1}^{\infty} \subset \mathcal{Y}$ and $\{c_j(T^{-1}\cdot)\}_{j=1}^{\infty} \subset \mathcal{Y}^*$ form a basis for $\mathcal{Y}$ by linearity and continuity of T and T^{-1}. $\blacksquare$

Lemma 20 *Let $\mathcal{X}$ be a Banach space with the* AP *and $\mathcal{Y}$ be any Banach space. Suppose there exists a continuous linear bijection $T : \mathcal{X} \to \mathcal{Y}$. Then $\mathcal{Y}$ has the* AP.

Proof Let $K \subset \mathcal{Y}$ be a compact set and $\epsilon > 0$. The set $R = T^{-1}(K) \subset \mathcal{X}$ is compact since T^{-1} is continuous. Since $\mathcal{X}$ has the AP, there exists a finite rank operator $U : \mathcal{X} \to \mathcal{X}$ such that

$$\|x - Ux\|_{\mathcal{X}} \leq \frac{\epsilon}{\|T\|}, \qquad \forall x \in R.$$

Define the operator $W : \mathcal{Y} \to \mathcal{Y}$ by $W = TUT^{-1}$. Clearly W is a finite rank operator since U is a finite rank operator. Let $y \in K$ then, since $K = T(R)$, there exists $x \in R$ such that $Tx = y$ and $x = T^{-1}y$. Then

$$\|y - Wy\|_{\mathcal{Y}} = \|Tx - TUx\|_{\mathcal{Y}} \leq \|T\|\|x - Ux\|_{\mathcal{X}} \leq \epsilon.$$

hence $\mathcal{Y}$ has the AP. ■

The following lemma shows than the infinite union of compact sets is compact if each set is the image of a fixed compact set under a convergent sequence of continuous maps. The result is instrumental in proving Lemma 22.

Lemma 21 *Let $\mathcal{X}, \mathcal{Y}$ be Banach spaces and $F : \mathcal{X} \to \mathcal{Y}$ be a continuous map. Let $K \subset \mathcal{X}$ be a compact set in $\mathcal{X}$ and $\{F_n : \mathcal{X} \to \mathcal{Y}\}_{n=1}^{\infty}$ be a sequence of continuous maps such that*

$$\lim_{n \to \infty} \sup_{x \in K} \|F(x) - F_n(x)\|_{\mathcal{Y}} = 0.$$

Then the set

$$W := \bigcup_{n=1}^{\infty} F_n(K) \cup F(K)$$

is compact in $\mathcal{Y}$.

Proof Let $\epsilon > 0$ then there exists a number $N = N(\epsilon) \in \mathbb{N}$ such that

$$\sup_{x \in K} \|F(x) - F_n(x)\|_{\mathcal{Y}} \leq \frac{\epsilon}{2}, \qquad \forall n \geq N.$$

Define the set

$$W_N = \bigcup_{n=1}^{N} F_n(K) \cup F(K)$$

which is compact since F and each F_n are continuous. We can therefore find a number $J = J(\epsilon, N) \in \mathbb{N}$ and elements $y_1, \ldots, y_J \in W_N$ such that, for any $z \in W_N$, there exists a number $l = l(z) \in \{1, \ldots, J\}$ such that

$$\|z - y_l\|_{\mathcal{Y}} \leq \frac{\epsilon}{2}.$$

Let $y \in W \setminus W_N$ then there exists a number $m > N$ and an element $x \in K$ such that $y = F_m(x)$. Since $F(x) \in W_N$, we can find a number $l \in \{1, \ldots, J\}$ such that

$$\|F(x) - y_l\|_{\mathcal{Y}} \leq \frac{\epsilon}{2}.$$

Therefore,

$$\|y - y_l\|_{\mathcal{Y}} \leq \|F_m(x) - F(x)\|_{\mathcal{Y}} + \|F(x) - y_l\|_{\mathcal{Y}} \leq \epsilon$$

hence $\{y_j\}_{j=1}^J$ forms a finite ϵ-net for W, showing that W is totally bounded.

We will now show that W is closed. To that end, let $\{p_n\}_{n=1}^\infty$ be a convergent sequence in W, in particular, $p_n \in W$ for every $n \in \mathbb{N}$ and $p_n \to p \in \mathcal{Y}$ as $n \to \infty$. We can thus find convergent sequences $\{x_n\}_{n=1}^\infty$ and $\{\alpha_n\}_{n=1}^\infty$ such that $x_n \in K$, $\alpha_n \in \mathbb{N}_0$, and $p_n = F_{\alpha_n}(x_n)$ where we define $F_0 := F$. Since K is closed, $\lim_{n\to\infty} x_n = x \in K$ thus, for each fixed $n \in \mathbb{N}$,

$$\lim_{j\to\infty} F_{\alpha_n}(x_j) = F_{\alpha_n}(x) \in W$$

by continuity of F_{α_n}. Since uniform convergence implies point-wise convergence

$$p = \lim_{n\to\infty} F_{\alpha_n}(x) = F_\alpha(x) \in W$$

for some $\alpha \in \mathbb{N}_0$ thus $p \in W$, showing that W is closed. ■

The following lemma shows that any continuous operator acting between two Banach spaces with the AP can be approximated in a finite-dimensional manner. The approximation proceeds in three steps which are shown schematically in Figure 16. First an input is mapped to a finite-dimensional representation via the action of a set of functionals on $\mathcal{X}$. This representation is then mapped by a continuous function to a new finite-dimensional representation which serves as the set of coefficients onto representers of $\mathcal{Y}$. The resulting expansion is an element of $\mathcal{Y}$ that is ϵ-close to the action of $\mathcal{G}$ on the input element. A similar finite-dimensionalization was used in (Bhattacharya et al., 2020) by using PCA on $\mathcal{X}$ to define the functionals acting on the input and PCA on $\mathcal{Y}$ to define the output representers. However the result in that work is restricted to separable Hilbert spaces; here, we generalize it to Banach spaces with the AP.

Lemma 22 *Let $\mathcal{X}, \mathcal{Y}$ be two Banach spaces with the AP and let $\mathcal{G} : \mathcal{X} \to \mathcal{Y}$ be a continuous map. For every compact set $K \subset \mathcal{X}$ and $\epsilon > 0$, there exist numbers $J, J' \in \mathbb{N}$ and continuous linear maps $F_J : \mathcal{X} \to \mathbb{R}^J$, $G_{J'} : \mathbb{R}^{J'} \to \mathcal{Y}$ as well as $\varphi \in C(\mathbb{R}^J; \mathbb{R}^{J'})$ such that*

$$\sup_{x \in K} \|\mathcal{G}(x) - (G_{J'} \circ \varphi \circ F_J)(x)\|_{\mathcal{Y}} \leq \epsilon.$$

Furthermore there exist $w_1, \ldots, w_J \in \mathcal{X}^*$ such that F_J has the form

$$F_J(x) = \big(w_1(x), \ldots, w_J(x)\big), \qquad \forall x \in \mathcal{X}$$

and there exist $\beta_1, \ldots, \beta_{J'} \in \mathcal{Y}$ such that $G_{J'}$ has the form

$$G_{J'}(v) = \sum_{j=1}^{J'} v_j \beta_j, \qquad \forall v \in \mathbb{R}^{J'}.$$

If $\mathcal{Y}$ admits a basis then $\{\beta_j\}_{j=1}^{J'}$ can be picked so that there is an extension $\{\beta_j\}_{j=1}^\infty \subset \mathcal{Y}$ which is a basis for $\mathcal{Y}$.

Proof Since $\mathcal{X}$ has the AP, there exists a sequence of finite rank operators $\{U_n^{\mathcal{X}} : \mathcal{X} \to \mathcal{X}\}_{n=1}^{\infty}$ such that

$$\lim_{n\to\infty} \sup_{x\in K} \|x - U_n^{\mathcal{X}} x\|_{\mathcal{X}} = 0.$$

Define the set

$$Z = \bigcup_{n=1}^{\infty} U_n^{\mathcal{X}}(K) \cup K$$

which is compact by Lemma 21. Therefore, $\mathcal{G}$ is uniformly continuous on Z hence there exists a modulus of continuity $\omega : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ which is non-decreasing and satisfies $\omega(t) \to \omega(0) = 0$ as $t \to 0$ as well as

$$\|\mathcal{G}(z_1) - \mathcal{G}(z_2)\|_{\mathcal{Y}} \leq \omega(\|z_1 - z_2\|_{\mathcal{X}}) \qquad \forall z_1, z_2 \in Z.$$

We can thus find, a number $N = N(\epsilon) \in \mathbb{N}$ such that

$$\sup_{x\in K} \omega(\|x - U_N^{\mathcal{X}} x\|_{\mathcal{X}}) \leq \frac{\epsilon}{2}.$$

Let $J = \dim U_N^{\mathcal{X}}(\mathcal{X}) < \infty$. There exist elements $\{\alpha_j\}_{j=1}^{J} \subset \mathcal{X}$ and $\{w_j\}_{j=1}^{J} \subset \mathcal{X}^*$ such that

$$U_N^{\mathcal{X}} x = \sum_{j=1}^{J} w_j(x) \alpha_j, \qquad \forall x \in X.$$

Define the maps $F_J^{\mathcal{X}} : \mathcal{X} \to \mathbb{R}^J$ and $G_J^{\mathcal{X}} : \mathbb{R}^J \to \mathcal{X}$ by

$$F_J^{\mathcal{X}}(x) = (w_1(x), \dots, w_J(x)), \qquad \forall x \in \mathcal{X},$$

$$G_J^{\mathcal{X}}(v) = \sum_{j=1}^{J} v_j \alpha_j, \qquad \forall v \in \mathbb{R}^J,$$

noting that $U_N^{\mathcal{X}} = G_J^{\mathcal{X}} \circ F_J^{\mathcal{X}}$. Define the set $W = (\mathcal{G} \circ U_N^{\mathcal{X}})(K) \subseteq \mathcal{Y}$ which is clearly compact. Since $\mathcal{Y}$ has the AP, we can similarly find a finite rank operator $U_{J'}^{\mathcal{Y}} : \mathcal{Y} \to \mathcal{Y}$ with $J' = \dim U_{J'}^{\mathcal{Y}}(\mathcal{Y}) < \infty$ such that

$$\sup_{y\in W} \|y - U_{J'}^{\mathcal{Y}} y\|_{\mathcal{Y}} \leq \frac{\epsilon}{2}.$$

Analogously, define the maps $F_{J'}^{\mathcal{Y}} : \mathcal{Y} \to \mathbb{R}^{J'}$ and $G_{J'}^{\mathcal{Y}} : \mathbb{R}^{J'} \to \mathcal{Y}$ by

$$F_{J'}^{\mathcal{Y}}(y) = (q_1(y), \dots, q_{J'}(y)), \qquad \forall y \in \mathcal{Y},$$

$$G_{J'}^{\mathcal{Y}}(v) = \sum_{j=1}^{J'} v_j \beta_j, \qquad \forall v \in \mathbb{R}^{J'}$$

for some $\{\beta_j\}_{j=1}^{J'} \subset \mathcal{Y}$ and $\{q_j\}_{j=1}^{J'} \subset \mathcal{Y}^*$ such that $U_{J'}^{\mathcal{Y}} = G_{J'}^{\mathcal{Y}} \circ F_{J'}^{\mathcal{Y}}$. Clearly if $\mathcal{Y}$ admits a basis then we could have defined $F_{J'}^{\mathcal{Y}}$ and $G_{J'}^{\mathcal{Y}}$ through it instead of through $U_{J'}^{\mathcal{Y}}$. Define $\varphi : \mathbb{R}^J \to \mathbb{R}^{J'}$ by

$$\varphi(v) = (F_{J'}^{\mathcal{Y}} \circ \mathcal{G} \circ G_J^{\mathcal{X}})(v), \qquad \forall v \in \mathbb{R}^J$$

76

which is clearly continuous and note that $G^{\mathcal{Y}}_{J'} \circ \varphi \circ F^{\mathcal{X}}_J = U^{\mathcal{Y}}_{J'} \circ \mathcal{G} \circ U^{\mathcal{X}}_N$. Set $F_J = F^{\mathcal{X}}_J$ and $G_{J'} = G^{\mathcal{Y}}_{J'}$ then, for any $x \in K$,

$$\|\mathcal{G}(x) - (G_{J'} \circ \varphi \circ F_J)(x)\|_{\mathcal{Y}} \le \|\mathcal{G}(x) - \mathcal{G}(U^{\mathcal{X}}_N x)\|_{\mathcal{Y}} + \|\mathcal{G}(U^{\mathcal{X}}_N x) - (U^{\mathcal{Y}}_{J'} \circ \mathcal{G} \circ U^{\mathcal{X}}_N)(x)\|_{\mathcal{Y}}$$
$$\le \omega\big(\|x - U^{\mathcal{X}}_N x\|_{\mathcal{X}}\big) + \sup_{y \in W} \|y - U^{\mathcal{Y}}_{J'} y\|_{\mathcal{Y}}$$
$$\le \epsilon$$

as desired. ∎

We now state and prove some results about isomorphisms of function spaces defined on different domains. These results are instrumental in proving Lemma 26.

Lemma 23 *Let $D, D' \subset \mathbb{R}^d$ be domains. Suppose that, for some $m \in \mathbb{N}_0$, there exists a C^m-diffeomorphism $\tau : \bar{D}' \to \bar{D}$. Then the mapping $T : C^m(\bar{D}) \to C^m(\bar{D}')$ defined as*

$$T(f)(x) = f(\tau(x)), \qquad \forall f \in C^m(\bar{D}), \ x \in \bar{D}'$$

is a continuous linear bijection.

Proof Clearly T is linear since the evaluation functional is linear. To see that it is continuous, note that by the chain rule we can find a constant $Q = Q(m) > 0$ such that

$$\|T(f)\|_{C^m} \le Q\|\tau\|_{C^m}\|f\|_{C^m}, \qquad \forall f \in C^m(\bar{D}).$$

We will now show that it is bijective. Let $f, g \in C^m(\bar{D})$ so that $f \ne g$. Then there exists a point $x \in \bar{D}$ such that $f(x) \ne g(x)$. Then $T(f)(\tau^{-1}(x)) = f(x)$ and $T(g)(\tau^{-1}(x)) = g(x)$ hence $T(f) \ne T(g)$ thus T is injective. Now let $g \in C^m(\bar{D}')$ and define $f : \bar{D} \to \mathbb{R}$ by $f = g \circ \tau^{-1}$. Since $\tau^{-1} \in C^m(\bar{D}; \bar{D}')$, we have that $f \in C^m(\bar{D})$. Clearly, $T(f) = g$ hence T is surjective. ∎

Corollary 24 *Let $M > 0$ and $m \in \mathbb{N}_0$. There exists a continuous linear bijection $T : C^m([0,1]^d) \to C^m([-M, M]^d)$.*

Proof Let $\mathbb{1} \in \mathbb{R}^d$ denote the vector in which all entries are 1. Define the map $\tau : \mathbb{R}^d \to \mathbb{R}^d$ by

$$\tau(x) = \frac{1}{2M}x + \frac{1}{2}\mathbb{1}, \qquad \forall x \in \mathbb{R}^d. \tag{48}$$

Clearly τ is a C^∞-diffeomorphism between $[-M, M]^d$ and $[0,1]^d$ hence Lemma 23 implies the result. ∎

Lemma 25 *Let $M > 0$ and $m \in \mathbb{N}$. There exists a continuous linear bijection $T : W^{m,1}((0,1)^d) \to W^{m,1}((-M, M)^d)$.*

Proof Define the map $\tau : \mathbb{R}^d \to \mathbb{R}^d$ by (48). We have that $\tau((-M, M)^d) = (0, 1)^d$. Define the operator T by

$$Tf = f \circ \tau, \qquad \forall f \in W^{m,1}((0, 1)^d).$$

which is clearly linear since composition is linear. We compute that, for any $0 \leq |\alpha|_1 \leq m$,

$$\partial^\alpha (f \circ \tau) = (2M)^{-|\alpha|_1} (\partial^\alpha f) \circ \tau$$

hence, by the change of variables formula,

$$\|Tf\|_{W^{m,1}((-M,M)^d)} = \sum_{0 \leq |\alpha|_1 \leq m} (2M)^{d-|\alpha|_1} \|\partial^\alpha f\|_{L^1((0,1)^d)}.$$

We can therefore find numbers $C_1, C_2 > 0$, depending on M and m, such that

$$C_1 \|f\|_{W^{m,1}((0,1)^d)} \leq \|Tf\|_{W^{m,1}((-M,M)^d)} \leq C_2 \|f\|_{W^{m,1}((0,1)^d)}.$$

This shows that $T : W^{m,1}((0, 1)^d) \to W^{m,1}((-M, M)^d)$ is continuous and injective. Now let $g \in W^{m,1}((-M, M)^d)$ and define $f = g \circ \tau^{-1}$. A similar argument shows that $f \in W^{m,1}((0, 1)^d)$ and, clearly, $Tf = g$ hence T is surjective. ∎

We now show that the spaces in Assumptions 9 and 10 have the AP. While the result is well-known when the domain is $(0, 1)^d$ or $\mathbb{R}^d$, we were unable to find any results in the literature for Lipschitz domains and we therefore give a full proof here. The essence of the proof is to either exhibit an isomorphism to a space that is already known to have AP or to directly show AP by embedding the Lipschitz domain into an hypercube for which there are known basis constructions. Our proof shows the stronger result that $W^{m,p}(D)$ for $m \in \mathbb{N}_0$ and $1 \leq p < \infty$ has a basis, but, for $C^m(\bar{D})$, we only establish the AP and not necessarily a basis. The discrepancy comes from the fact that there is an isomorphism between $W^{m,p}(D)$ and $W^{m,p}(\mathbb{R}^d)$ while there is not one between $C^m(\bar{D})$ and $C^m(\mathbb{R}^d)$.

Lemma 26 *Let Assumptions 9 and 10 hold. Then $\mathcal{A}$ and $\mathcal{U}$ have the* AP.

Proof It is enough to show that the spaces $W^{m,p}(D)$, and $C^m(\bar{D})$ for any $1 \leq p < \infty$ and $m \in \mathbb{N}_0$ with $D \subset \mathbb{R}^d$ a Lipschitz domain have the AP. Consider first the spaces $W^{0,p}(D) = L^p(D)$. Since the Lebesgue measure on D is σ-finite and has no atoms, $L^p(D)$ is isometrically isomorphic to $L^p((0, 1))$ (see, for example, (Albiac and Kalton, 2006, Chapter 6)). Hence by Lemma 20, it is enough to show that $L^p((0, 1))$ has the AP. Similarly, consider the spaces $W^{m,p}(D)$ for $m > 0$ and $p > 1$. Since D is Lipschitz, there exists a continuous linear operator $W^{m,p}(D) \to W^{m,p}(\mathbb{R}^d)$ (Stein, 1970, Chapter 6, Theorem 5) (this also holds for $p = 1$). We can therefore apply (Pełczyński and Wojciechowski, 2001, Corollary 4) (when $p > 1$) to conclude that $W^{m,p}(D)$ is isomorphic to $L^p((0, 1))$. By (Albiac and Kalton, 2006, Proposition 6.1.3), $L^p((0, 1))$ has a basis hence Lemma 18 implies the result.

Now, consider the spaces $C^m(\bar{D})$. Since D is bounded, there exists a number $M > 0$ such that $\bar{D} \subseteq [-M, M]^d$. Hence, by Corollary 24, $C^m([0, 1]^d)$ is isomorphic to $C^m([-M, M]^d)$. Since $C^m([0, 1]^d)$ has a basis (Ciesielski and Domsta, 1972, Theorem 5), Lemma 19 then implies that $C^m([-M, M]^d)$ has a basis. By (Fefferman, 2007, Theorem 1), there exists a continuous linear

operator $E : C^m(\bar{D}) \to C_b^m(\mathbb{R}^d)$ such that $E(f)|_{\bar{D}} = f$ for all $f \in C(\bar{D})$. Define the restriction operators $R_M : C_b^m(\mathbb{R}^d) \to C^m([-M,M]^d)$ and $R_D : C^m([-M,M]^d) \to C^m(\bar{D})$ which are both clearly linear and continuous and $\|R_M\| = \|R_D\| = 1$. Let $\{c_j\}_{j=1}^\infty \subset \left(C^m([-M,M]^d)\right)^*$ and $\{\varphi_j\}_{j=1}^\infty \subset C^m([-M,M]^d)$ be a basis for $C^m([-M,M]^d)$. As in the proof of Lemma 18, there exists a constant $C_1 > 0$ such that, for any $n \in \mathbb{N}$ and $f \in C^m([-M,M]^d)$,

$$\|\sum_{j=1}^n c_j(f)\varphi_j\|_{C^m([-M,M]^d)} \leq C_1\|f\|_{C^m([-M,M]^d)}.$$

Suppose, without loss of generality, that $C_1\|E\| \geq 1$. Let $K \subset C^m(\bar{D})$ be a compact set and $\epsilon > 0$. Since K is compact, we can find a number $n = n(\epsilon) \in \mathbb{N}$ and elements $y_1, \dots, y_n \in K$ such that, for any $f \in K$ there exists a number $l \in \{1, \dots, n\}$ such that

$$\|f - y_l\|_{C^m(\bar{D})} \leq \frac{\epsilon}{3C_1\|E\|}.$$

For every $l \in \{1, \dots, n\}$, define $g_l = R_M(E(y_l))$ and note that $g_l \in C^m([-M,M]^d)$ hence there exists a number $J = J(\epsilon, n) \in \mathbb{N}$ such that

$$\max_{l \in \{1,\dots,n\}} \|g_l - \sum_{j=1}^J c_j(g_l)\varphi_j\|_{C^m([-M,M]^d)} \leq \frac{\epsilon}{3}.$$

Notice that, since $y_l = R_D(g_l)$, we have

$$\max_{l \in \{1,\dots,n\}} \|y_l - \sum_{j=1}^J c_j\big(R_M(E(y_l))\big)R_D(\varphi_j)\|_{C^m(\bar{D})} \leq \|R_D\| \max_{l \in \{1,\dots,n\}} \|g_l - \sum_{j=1}^J c_j(g_l)\varphi_j\|_{C^m([-M,M]^d)} \leq \frac{\epsilon}{3}.$$

Define the finite rank operator $U : C^m(\bar{D}) \to C^m(\bar{D})$ by

$$Uf = \sum_{j=1}^J c_j\big(R_M(E(f))\big)R_D(\varphi_j), \qquad \forall f \in C^m(\bar{D}).$$

We then have that, for any $f \in K$,

$$\|f - Uf\|_{C^m(\bar{D})} \leq \|f - y_l\|_{C^m(\bar{D})} + \|y_l - Uy_l\|_{C^m(\bar{D})} + \|Uy_l - Uf\|_{C^m(\bar{D})}$$
$$\leq \frac{2\epsilon}{3} + \|\sum_{j=1}^J c_j\big(R_M(E(y_l - f))\big)\varphi_j\|_{C^m([-M,M]^d)}$$
$$\leq \frac{2\epsilon}{3} + C_1\|R_M(E(y_l - f))\|_{C^m([-M,M]^d)}$$
$$\leq \frac{2\epsilon}{3} + C_1\|E\|\|y_l - f\|_{C^m(\bar{D})}$$
$$\leq \epsilon$$

hence $C^m(\bar{D})$ has the AP.

We are left with the case $W^{m,1}(D)$. A similar argument as for the $C^m(\bar{D})$ case holds. In particular the basis from (Ciesielski and Domsta, 1972, Theorem 5) is also a basis for $W^{m,1}((0,1)^d)$. Lemma 25 gives an isomorphism between $W^{m,1}((0,1)^d)$ and $W^{m,1}((-M,M)^d)$ hence we may use the extension operator $W^{m,1}(D) \to W^{m,1}(\mathbb{R}^d)$ from (Stein, 1970, Chapter 6, Theorem 5) to complete the argument. In fact, the same construction yields a basis for $W^{m,1}(D)$ due to the isomorphism with $W^{m,1}(\mathbb{R}^d)$, see, for example (Pełczyński and Wojciechowski, 2001, Theorem 1). ∎

Appendix C.

In this section, we prove various results about the approximation of linear functionals by kernel integral operators. Lemma 27 establishes a Riesz-representation theorem for C^m. The proof proceeds exactly as in the well-known result for $W^{m,p}$ but, since we did not find it in the literature, we give full details here. Lemma 28 shows that linear functionals on $W^{m,p}$ can be approximated uniformly over compact set by integral kernel operators with a C^∞ kernel. Lemmas 30 and 31 establish similar results for C and C^m respectively by employing Lemma 27. These lemmas are crucial in showing that NO(s) are universal since they imply that the functionals from Lemma 22 can be approximated by elements of IO.

Lemma 27 *Let $D \subset \mathbb{R}^d$ be a domain and $m \in \mathbb{N}_0$. For every $L \in \big(C^m(\bar{D})\big)^*$ there exist finite, signed, Radon measures $\{\lambda_\alpha\}_{0 \le |\alpha|_1 \le m}$ such that*

$$L(f) = \sum_{0 \le |\alpha|_1 \le m} \int_{\bar{D}} \partial^\alpha f \, \mathrm{d}\lambda_\alpha, \qquad \forall f \in C^m(\bar{D}).$$

Proof The case $m = 0$ follow directly from (Leoni, 2009, Theorem B.111), so we assume that $m > 0$. Let $\alpha_1, \dots, \alpha_J$ be an enumeration of the set $\{\alpha \in \mathbb{N}^d : |\alpha|_1 \le m\}$. Define the mapping $T : C^m(\bar{D}) \to C(\bar{D}; \mathbb{R}^J)$ by

$$Tf = \big(\partial^{\alpha_0} f, \dots, \partial^{\alpha_J} f\big), \qquad \forall f \in C^m(\bar{D}).$$

Clearly $\|Tf\|_{C(\bar{D};\mathbb{R}^J)} = \|f\|_{C^m(\bar{D})}$ hence T is an injective, continuous linear operator. Define $W := T(C^m(\bar{D})) \subset C(\bar{D}; \mathbb{R}^J)$ then $T^{-1} : W \to C^m(\bar{D})$ is a continuous linear operator since T preserves norm. Thus $W = \big(T^{-1}\big)^{-1}(C^m(\bar{D}))$ is closed as the pre-image of a closed set under a continuous map. In particular, W is a Banach space since $C(\bar{D}; \mathbb{R}^J)$ is a Banach space and T is an isometric isomorphism between $C^m(\bar{D})$ and W. Therefore, there exists a continuous linear functional $\tilde{L} \in W^*$ such that

$$L(f) = \tilde{L}(Tf), \qquad \forall f \in C^m(\bar{D}).$$

By the Hahn-Banach theorem, $\tilde{L}$ can be extended to a continuous linear functional $\bar{L} \in \big(C(\bar{D}; \mathbb{R}^J)\big)^*$ such that $\|L\|_{(C^m(\bar{D}))^*} = \|\tilde{L}\|_{W^*} = \|\bar{L}\|_{(C(\bar{D};\mathbb{R}^J))^*}$. We have that

$$L(f) = \tilde{L}(Tf) = \bar{L}(Tf), \qquad \forall f \in C^m(\bar{D}).$$

Since

$$\big(C(\bar{D}; \mathbb{R}^J)\big)^* \cong \bigtimes_{j=1}^{J} \big(C(\bar{D})\big)^* \cong \bigoplus_{j=1}^{J} \big(C(\bar{D})\big)^*,$$

we have, by applying (Leoni, 2009, Theorem B.111) J times, that there exist finite, signed, Radon measures $\{\lambda_\alpha\}_{0 \leq |\alpha|_1 \leq m}$ such that

$$\bar{L}(Tf) = \sum_{0 \leq |\alpha|_1 \leq m} \int_{\bar{D}} \partial^\alpha f \, \mathsf{d}\lambda_\alpha, \qquad \forall f \in C^m(\bar{D})$$

as desired. ∎

Lemma 28 *Let $D \subset \mathbb{R}^d$ be a bounded, open set and $L \in (W^{m,p}(D))^*$ for some $m \geq 0$ and $1 \leq p < \infty$. For any closed and bounded set $K \subset W^{m,p}(D)$ (compact if $p = 1$) and $\epsilon > 0$, there exists a function $\kappa \in C_c^\infty(D)$ such that*

$$\sup_{u \in K} |L(u) - \int_D \kappa u \, \mathsf{d}x| < \epsilon.$$

Proof First consider the case $m = 0$ and $1 \leq p < \infty$. By the Riesz Representation Theorem (Conway, 2007, Appendix B), there exists a function $v \in L^q(D)$ such that

$$L(u) = \int_D vu \, \mathsf{d}x.$$

Since K is bounded, there is a constant $M > 0$ such that

$$\sup_{u \in K} \|u\|_{L^p} \leq M.$$

Suppose $p > 1$, so that $1 < q < \infty$. Density of $C_c^\infty(D)$ in $L^q(D)$ (Adams and Fournier, 2003, Corollary 2.30) implies there exists a function $\kappa \in C_c^\infty(D)$ such that

$$\|v - \kappa\|_{L^q} < \frac{\epsilon}{M}.$$

By the Hölder inequality,

$$|L(u) - \int_D \kappa u \, \mathsf{d}x| \leq \|u\|_{L^p} \|v - \kappa\|_{L^q} < \epsilon.$$

Suppose that $p = 1$ then $q = \infty$. Since K is totally bounded, there exists a number $n \in \mathbb{N}$ and functions $g_1, \ldots, g_n \in K$ such that, for any $u \in K$,

$$\|u - g_l\|_{L^1} < \frac{\epsilon}{3\|v\|_{L^\infty}}$$

for some $l \in \{1, \ldots, n\}$. Let $\psi_\eta \in C_c^\infty(D)$ denote a standard mollifier for any $\eta > 0$. We can find $\eta > 0$ small enough such that

$$\max_{l \in \{1,\ldots,n\}} \|\psi_\eta * g_l - g_l\|_{L^1} < \frac{\epsilon}{9\|v\|_{L^\infty}}$$

81

Define $f = \psi_\eta * v \in C(D)$ and note that $\|f\|_{L^\infty} \le \|v\|_{L^\infty}$. By Fubini's theorem, we find

$$|\int_D (f - v)g_l \, dx| = \int_D v(\psi_\eta * g_l - g_l) \, dx \le \|v\|_{L^\infty} \|\psi_\eta * g_l - g_l\|_{L^1} < \frac{\epsilon}{9}.$$

Since $g_l \in L^1(D)$, by Lusin's theorem, we can find a compact set $A \subset D$ such that

$$\max_{l \in \{1,\dots,n\}} \int_{D \setminus A} |g_l| \, dx < \frac{\epsilon}{18\|v\|_{L^\infty}}$$

Since $C_c^\infty(D)$ is dense in $C(D)$ over compact sets (Leoni, 2009, Theorem C.16), we can find a function $\kappa \in C_c^\infty(D)$ such that

$$\sup_{x \in A} |\kappa(x) - f(x)| \le \frac{\epsilon}{9M}$$

and $\|\kappa\|_{L^\infty} \le \|f\|_{L^\infty} \le \|v\|_{L^\infty}$. We have,

$$\begin{aligned}
|\int_D (\kappa - v)g_l \, dx| &\le \int_A |(\kappa - v)g_l| \, dx + \int_{D \setminus A} |(\kappa - v)g_l| \, dx \\
&\le \int_A |(\kappa - f)g_l| \, dx + \int_D |(f - v)g_l| \, dx + 2\|v\|_{L^\infty} \int_{D \setminus A} |g_l| \, dx \\
&\le \sup_{x \in A} |\kappa(x) - f(x)| \|g_l\|_{L^1} + \frac{2\epsilon}{9} \\
&< \frac{\epsilon}{3}.
\end{aligned}$$

Finally,

$$\begin{aligned}
|L(u) - \int_D \kappa u \, dx| &\le |\int_D vu \, dx - \int_D vg_l \, dx| + |\int_D vg_l \, dx - \int_D \kappa u \, dx| \\
&\le \|v\|_{L^\infty} \|u - g_l\|_{L^1} + |\int_D \kappa u \, dx - \int_D \kappa g_l \, dx| + |\int_D \kappa g_l \, dx - \int_D vg_l \, dx| \\
&\le \frac{\epsilon}{3} + \|\kappa\|_{L^\infty} \|u - g_l\|_{L^1} + |\int_D (\kappa - v)g_l \, dx| \\
&\le \frac{2\epsilon}{3} + \|v\|_{L^\infty} \|u - g_l\|_{L^1} \\
&< \epsilon.
\end{aligned}$$

Suppose $m \ge 1$. By the Riesz Representation Theorem (Adams and Fournier, 2003, Theorem 3.9), there exist elements $(v_\alpha)_{0 \le |\alpha|_1 \le m}$ of $L^q(D)$ where $\alpha \in \mathbb{N}^d$ is a multi-index such that

$$L(u) = \sum_{0 \le |\alpha|_1 \le m} \int_D v_\alpha \partial_\alpha u \, dx.$$

Since K is bounded, there is a constant $M > 0$ such that

$$\sup_{u \in K} \|u\|_{W^{m,p}} \le M.$$

Suppose $p > 1$, so that $1 < q < \infty$. Density of $C_0^\infty(D)$ in $L^q(D)$ implies there exist functions $(f_\alpha)_{0 \leq |\alpha|_1 \leq m}$ in $C_c^\infty(D)$ such that

$$\|f_\alpha - v_\alpha\|_{L^q} < \frac{\epsilon}{MJ}$$

where $J = |\{\alpha \in \mathbb{N}^d : |\alpha|_1 \leq m\}|$. Let

$$\kappa = \sum_{0 \leq |\alpha|_1 \leq m} (-1)^{|\alpha|_1} \partial_\alpha f_\alpha$$

then, by definition of a weak derivative,

$$\int_D \kappa u \, \mathrm{d}x = \sum_{0 \leq |\alpha|_1 \leq m} (-1)^{|\alpha|_1} \int_D \partial_\alpha f_\alpha u \, \mathrm{d}x = \sum_{0 \leq |\alpha|_1 \leq m} \int_D f_\alpha \partial_\alpha u \, \mathrm{d}x.$$

By the Hölder inequality,

$$\left| L(u) - \int_D \kappa u \, \mathrm{d}x \right| \leq \sum_{0 \leq |\alpha|_1 \leq m} \|\partial_\alpha u\|_{L^p} \|f_\alpha - v_\alpha\|_{L^q} < M \sum_{0 \leq |\alpha|_1 \leq m} \frac{\epsilon}{MJ} = \epsilon.$$

Suppose that $p = 1$ then $q = \infty$. Define the constant $C_v > 0$ by

$$C_v = \sum_{0 \leq |\alpha|_1 \leq m} \|v_\alpha\|_{L^\infty}.$$

Since K is totally bounded, there exists a number $n \in \mathbb{N}$ and functions $g_1, \ldots, g_n \in K$ such that, for any $u \in K$,

$$\|u - g_l\|_{W^{m,1}} < \frac{\epsilon}{3C_v}$$

for some $l \in \{1, \ldots, n\}$. Let $\psi_\eta \in C_c^\infty(D)$ denote a standard mollifier for any $\eta > 0$. We can find $\eta > 0$ small enough such that

$$\max_\alpha \max_{l \in \{1,\ldots,n\}} \|\psi_\eta * \partial_\alpha g_l - \partial_\alpha g_l\|_{L^1} < \frac{\epsilon}{9C_v}.$$

Define $f_\alpha = \psi_\eta * v_\alpha \in C(D)$ and note that $\|f_\alpha\|_{L^\infty} \leq \|v_\alpha\|_{L^\infty}$. By Fubini's theorem, we find

$$\sum_{0 \leq |\alpha|_1 \leq m} \left| \int_D (f_\alpha - v_\alpha) \partial_\alpha g_l \, \mathrm{d}x \right| = \sum_{0 \leq |\alpha|_1 \leq m} \left| \int_D v_\alpha (\psi_\eta * \partial_\alpha g_l - \partial_\alpha g_l) \, \mathrm{d}x \right|$$

$$\leq \sum_{0 \leq |\alpha|_1 \leq m} \|v_\alpha\|_{L^\infty} \|\psi_\eta * \partial_\alpha g_l - \partial_\alpha g_l\|_{L^1}$$

$$< \frac{\epsilon}{9}.$$

Since $\partial_\alpha g_l \in L^1(D)$, by Lusin's theorem, we can find a compact set $A \subset D$ such that

$$\max_\alpha \max_{l \in \{1,\ldots,n\}} \int_{D \setminus A} |\partial_\alpha g_l| \, \mathrm{d}x < \frac{\epsilon}{18C_v}.$$

83

Since $C_c^\infty(D)$ is dense in $C(D)$ over compact sets, we can find functions $w_\alpha \in C_c^\infty(D)$ such that

$$\sup_{x \in A} |w_\alpha(x) - f_\alpha(x)| \le \frac{\epsilon}{9MJ}$$

where $J = |\{\alpha \in \mathbb{N}^d : |\alpha|_1 \le m\}|$ and $\|w_\alpha\|_{L^\infty} \le \|f_\alpha\|_{L^\infty} \le \|v_\alpha\|_{L^\infty}$. We have,

$$
\begin{aligned}
\sum_{0 \le |\alpha|_1 \le m} \int_D |(w_\alpha - v_\alpha)\partial_\alpha g_l| &= \sum_{0 \le |\alpha|_1 \le m} \left(\int_A |(w_\alpha - v_\alpha)\partial_\alpha g_l| dx + \int_{D \backslash A} |(w_\alpha - v_\alpha)\partial_\alpha g_l| dx \right) \\
&\le \sum_{0 \le |\alpha|_1 \le m} \left(\int_A |(w_\alpha - f_\alpha)\partial_\alpha g_l| \, \mathrm{d}x + \int_D |(f_\alpha - v_\alpha)\partial_\alpha g_l| \, \mathrm{d}x \right. \\
&\qquad\qquad \left. + 2\|v_\alpha\|_{L^\infty} \int_{D \backslash A} |\partial_\alpha g_l| \, \mathrm{d}x \right) \\
&\le \sum_{0 \le |\alpha|_1 \le m} \sup_{x \in A} |w_\alpha(x) - f_\alpha(x)| \|\partial_\alpha g_l\|_{L^1} + \frac{2\epsilon}{9} \\
&< \frac{\epsilon}{3}.
\end{aligned}
$$

Let

$$\kappa = \sum_{0 \le |\alpha|_1 \le m} (-1)^{|\alpha|_1} \partial_\alpha w_\alpha.$$

then, by definition of a weak derivative,

$$\int_D \kappa u \, \mathrm{d}x = \sum_{0 \le |\alpha|_1 \le m} (-1)^{|\alpha|_1} \int_D \partial_\alpha w_\alpha u \, \mathrm{d}x = \sum_{0 \le |\alpha|_1 \le m} \int_D w_\alpha \partial_\alpha u \, \mathrm{d}x.$$

Finally,

$$
\begin{aligned}
\left| L(u) - \int_D \kappa u \, \mathrm{d}x \right| &\le \sum_{0 \le |\alpha|_1 \le m} \int_D |v_\alpha \partial_\alpha u - w_\alpha \partial_\alpha u| \, \mathrm{d}x \\
&\le \sum_{0 \le |\alpha|_1 \le m} \left(\int_D |v_\alpha(\partial_\alpha u - \partial_\alpha g_l)| \, \mathrm{d}x + \int_D |v_\alpha \partial_\alpha g_l - w_\alpha \partial_\alpha u| \, \mathrm{d}x \right) \\
&\le \sum_{0 \le |\alpha|_1 \le m} \left(\|v_\alpha\|_{L^\infty} \|u - g_l\|_{W^{m,1}} + \int_D |(v_\alpha - w_\alpha)\partial_\alpha g_l| \, \mathrm{d}x + \int_D |(\partial_\alpha g_l - \partial_\alpha u)w_\alpha| \, \mathrm{d}x \right) \\
&< \frac{2\epsilon}{3} + \sum_{0 \le |\alpha|_1 \le m} \|w_\alpha\|_{L^\infty} \|u - g_l\|_{W^{m,1}} \\
&< \epsilon.
\end{aligned}
$$

$\blacksquare$

Lemma 29 *Let $D \subset \mathbb{R}^d$ be a domain and $L \in \left(C^m(\bar{D})\right)^*$ for some $m \in \mathbb{N}_0$. For any compact set $K \subset C^m(\bar{D})$ and $\epsilon > 0$, there exists distinct points $y_{11}, \ldots, y_{1n_1}, \ldots, y_{Jn_J} \in D$ and numbers $c_{11}, \ldots, c_{1n_1}, \ldots, c_{Jn_J} \in \mathbb{R}$ such that*

$$\sup_{u \in K} |L(u) - \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \epsilon$$

where $\alpha_1, \ldots, \alpha_J$ is an enumeration of the set $\{\alpha \in \mathbb{N}_0^d : 0 \leq |\alpha|_1 \leq m\}$.

Proof By Lemma 27, there exist finite, signed, Radon measures $\{\lambda_\alpha\}_{0 \leq |\alpha|_1 \leq m}$ such that

$$L(u) = \sum_{0 \leq |\alpha|_1 \leq m} \int_{\bar{D}} \partial^\alpha u \, \mathsf{d}\lambda_\alpha, \qquad \forall u \in C^m(\bar{D}).$$

Let $\alpha_1, \ldots, \alpha_J$ be an enumeration of the set $\{\alpha \in \mathbb{N}_0^d : 0 \leq |\alpha|_1 \leq m\}$. By weak density of the Dirac measures (Bogachev, 2007, Example 8.1.6), we can find points $y_{11}, \ldots, y_{1n_1}, \ldots, y_{J1}, \ldots, y_{Jn_J} \in \bar{D}$ as well as numbers $c_{11}, \ldots, c_{Jn_J} \in \mathbb{R}$ such that

$$|\int_{\bar{D}} \partial^{\alpha_j} u \, \mathsf{d}\lambda_{\alpha_j} - \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \frac{\epsilon}{4J}, \qquad \forall u \in C^m(\bar{D})$$

for any $j \in \{1, \ldots, J\}$. Therefore,

$$|\sum_{j=1}^{J} \int_{\bar{D}} \partial^{\alpha_j} u \, \mathsf{d}\lambda_{\alpha_j} - \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \frac{\epsilon}{4}, \qquad \forall u \in C^m(\bar{D}).$$

Define the constant

$$Q := \sum_{j=1}^{J} \sum_{k=1}^{n_j} |c_{jk}|.$$

Since K is compact, we can find functions $g_1, \ldots, g_N \in K$ such that, for any $u \in K$, there exists $l \in \{1, \ldots, N\}$ such that

$$\|u - g_l\|_{C^k} \leq \frac{\epsilon}{4Q}.$$

Suppose that some $y_{jk} \in \partial D$. By uniform continuity, we can find a point $\tilde{y}_{jk} \in D$ such that

$$\max_{l \in \{1, \ldots, N\}} |\partial^{\alpha_j} g_l(y_{jk}) - \partial^{\alpha_j} g_l(\tilde{y}_{jk})| \leq \frac{\epsilon}{4Q}.$$

Denote

$$S(u) = \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})$$

85

and by $\tilde{S}(u)$ the sum $S(u)$ with y_{jk} replaced by $\tilde{y}_{jk}$. Then, for any $u \in K$, we have

$$
\begin{aligned}
|L(u) - \tilde{S}(u)| &\leq |L(u) - S(u)| + |S(u) - \tilde{S}(u)| \\
&\leq \frac{\epsilon}{4} + |c_{jk}\partial^{\alpha_j}u(\tilde{y}_{jk}) - c_{jk}\partial^{\alpha_j}u(y_{jk})| \\
&\leq \frac{\epsilon}{4} + |c_{jk}\partial^{\alpha_j}u(\tilde{y}_{jk}) - c_{jk}\partial^{\alpha_j}g_l(\tilde{y}_{jk})| + |c_{jk}\partial^{\alpha_j}g_l(\tilde{y}_{jk}) - c_{jk}\partial^{\alpha_j}u(y_{jk})| \\
&\leq \frac{\epsilon}{4} + |c_{jk}|\|u - g_l\|_{C^m} + |c_{jk}\partial^{\alpha_j}g_l(\tilde{y}_{jk}) - c_{jk}\partial^{\alpha_j}g_l(y_{jk})| + |c_{jk}\partial^{\alpha_j}g_l(y_{jk}) - c_{jk}\partial^{\alpha_j}u(y_{jk})| \\
&\leq \frac{\epsilon}{4} + 2|c_{jk}|\|u - g_l\|_{C^m} + |c_{jk}||\partial^{\alpha_j}g_l(\tilde{y}_{jk}) - \partial^{\alpha_j}g_l(y_{jk})| \\
&\leq \epsilon.
\end{aligned}
$$

Since there are a finite number of points, this implies that all points y_{jk} can be chosen in D. Suppose now that $y_{jk} = y_{qp}$ for some $(j,k) \neq (q,p)$. As before, we can always find a point $\tilde{y}_{jk}$ distinct from all others such that

$$
\max_{l \in \{1,\dots,N\}} |\partial^{\alpha_j}g_l(y_{jk}) - \partial^{\alpha_j}g_l(\tilde{y}_{jk})| \leq \frac{\epsilon}{4Q}.
$$

Repeating the previous argument then shows that all points y_{jk} can be chosen distinctly as desired. ■

Lemma 30 *Let $D \subset \mathbb{R}^d$ be a domain and $L \in \left(C(\bar{D})\right)^*$. For any compact set $K \subset C(\bar{D})$ and $\epsilon > 0$, there exists a function $\kappa \in C_c^\infty(D)$ such that*

$$
\sup_{u \in K} |L(u) - \int_D \kappa u \, \mathrm{d}x| < \epsilon.
$$

Proof By Lemma 29, we can find points distinct points $y_1, \dots, y_n \in D$ as well as numbers $c_1, \dots, c_n \in \mathbb{R}$ such that

$$
\sup_{u \in K} |L(u) - \sum_{j=1}^n c_j u(y_j)| \leq \frac{\epsilon}{3}.
$$

Define the constants

$$
Q := \sum_{j=1}^n |c_j|.
$$

Since K is compact, there exist functions $g_1, \dots, g_J \in K$ such that, for any $u \in K$, there exists some $l \in \{1, \dots, J\}$ such that

$$
\|u - g_l\|_C \leq \frac{\epsilon}{6nQ}.
$$

Let $r > 0$ be such that the open balls $B_r(y_j) \subset D$ and are pairwise disjoint. Let $\psi_\eta \in C_c^\infty(\mathbb{R}^d)$ denote the standard mollifier with parameter $\eta > 0$, noting that $\operatorname{supp} \psi_r = B_r(0)$. We can find a number $0 < \gamma \leq r$ such that

$$
\max_{\substack{l \in \{1,\dots,J\} \\ j \in \{1,\dots,n\}}} |\int_D \psi_\gamma(x - y_j)g_l(x) \, \mathrm{d}x - g_l(y_j)| \leq \frac{\epsilon}{3nQ}.
$$

Define $\kappa : \mathbb{R}^d \to \mathbb{R}$ by

$$\kappa(x) = \sum_{j=1}^{n} c_j \psi_\gamma(x - y_j), \qquad \forall x \in \mathbb{R}^d.$$

Since supp $\psi_\gamma(\cdot - y_j) \subseteq B_r(y_j)$, we have that $\kappa \in C_c^\infty(D)$. Then, for any $u \in K$,

$$|L(u) - \int_D \kappa u \, \mathrm{d}x| \leq |L(u) - \sum_{j=1}^{n} c_j u(y_j)| + |\sum_{j=1}^{n} c_j u(y_j) - \int_D \kappa u \, \mathrm{d}x|$$

$$\leq \frac{\epsilon}{3} + \sum_{j=1}^{n} |c_j| |u(y_j) - \int_D \psi_\eta(x - y_j) u(x) \, \mathrm{d}x|$$

$$\leq \frac{\epsilon}{3} + Q \sum_{j=1}^{n} |u(y_j) - g_l(y_j)| + |g_l(y_j) - \int_D \psi_\eta(x - y_j) u(x) \, \mathrm{d}x|$$

$$\leq \frac{\epsilon}{3} + nQ\|u - g_l\|_C + Q \sum_{j=1}^{n} |g_l(y_j) - \int_D \psi_\eta(x - y_j) g_l(x) \, \mathrm{d}x|$$

$$+ |\int_D \psi_\eta(x - y_j)\big(g_l(x) - u(x)\big) \, \mathrm{d}x|$$

$$\leq \frac{\epsilon}{3} + nQ\|u - g_l\|_C + nQ\frac{\epsilon}{3nQ} + Q\|g_l - u\|_C \sum_{j=1}^{n} \int_D \psi_\gamma(x - y_j) \, \mathrm{d}x$$

$$= \frac{2\epsilon}{3} + 2nQ\|u - g_l\|_C$$

$$= \epsilon$$

where we use the fact that mollifiers are non-negative and integrate to one. $\blacksquare$

Lemma 31 *Let $D \subset \mathbb{R}^d$ be a domain and $L \in \big(C^m(\bar{D})\big)^*$. For any compact set $K \subset C^m(\bar{D})$ and $\epsilon > 0$, there exist functions $\kappa_1, \ldots, \kappa_J \in C_c^\infty(D)$ such that*

$$\sup_{u \in K} |L(u) - \sum_{j=1}^{J} \int_D \kappa_j \partial^{\alpha_j} u \, \mathrm{d}x| < \epsilon$$

where $\alpha_1, \ldots, \alpha_J$ is an enumeration of the set $\{\alpha \in \mathbb{N}_0^d : 0 \leq |\alpha|_1 \leq m\}$.

Proof By Lemma 29, we find distinct points $y_{11}, \ldots, y_{1n_1}, \ldots, y_{Jn_J} \in D$ and numbers $c_{11}, \ldots, c_{Jn_J} \in \mathbb{R}$ such that

$$\sup_{u \in K} |L(u) - \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \frac{\epsilon}{2}.$$

Applying the proof of Lemma 31 J times to each of the inner sums, we find functions $\kappa_1, \ldots, \kappa_J \in C_c^\infty(D)$ such that

$$\max_{j \in \{1, \ldots, J\}} |\int_D \kappa_j \partial^{\alpha_j} u \, \mathrm{d}x - \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \frac{\epsilon}{2J}.$$

Then, for any $u \in K$,

$$|L(u) - \sum_{j=1}^{J} \int_D \kappa_j \partial^{\alpha_j} u \, \mathsf{d}x|$$

$$\leq |L(u) - \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| + \sum_{j=1}^{J} |\int_D \kappa_j \partial^{\alpha_j} u \, \mathsf{d}x - \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \epsilon$$

as desired. ∎

Appendix D.

The following lemmas show that the three pieces used in constructing the approximation from Lemma 22, which are schematically depicted in Figure 16, can all be approximated by NO(s). Lemma 32 shows that $F_J : \mathcal{A} \to \mathbb{R}^J$ can be approximated by an element of IO by mapping to a vector-valued constant function. Similarly, Lemma 34 shows that $G_{J'} : \mathbb{R}^{J'} \to \mathcal{U}$ can be approximated by an element of IO by mapping a vector-valued constant function to the coefficients of a basis expansion. Finally, Lemma 35 shows that NO(s) can exactly represent any standard neural network by viewing the inputs and outputs as vector-valued constant functions.

Lemma 32 *Let Assumption 9 hold. Let $\{c_j\}_{j=1}^{n} \subset \mathcal{A}^*$ for some $n \in \mathbb{N}$. Define the map $F : \mathcal{A} \to \mathbb{R}^n$ by*

$$F(a) = \big(c_1(a), \ldots, c_n(a)\big), \qquad \forall a \in \mathcal{A}.$$

Then, for any compact set $K \subset \mathcal{A}$, $\sigma \in \mathsf{A}_0$, and $\epsilon > 0$, there exists a number $L \in \mathbb{N}$ and neural network $\kappa \in \mathsf{N}_L(\sigma; \mathbb{R}^d \times \mathbb{R}^d, \mathbb{R}^{n \times 1})$ such that

$$\sup_{a \in K} \sup_{y \in \bar{D}} |F(a) - \int_D \kappa(y, x) a(x) \, \mathsf{d}x|_1 \leq \epsilon.$$

Proof Since K is bounded, there exists a number $M > 0$ such that

$$\sup_{a \in K} \|a\|_{\mathcal{A}} \leq M.$$

Define the constant

$$Q := \begin{cases} M, & \mathcal{A} = W^{m,p}(D) \\ M|D|, & A = C(\bar{D}) \end{cases}$$

and let $p = 1$ if $\mathcal{A} = C(\bar{D})$. By Lemma 28 and Lemma 30, there exist functions $f_1, \ldots, f_n \in C_c^\infty(D)$ such that

$$\max_{j \in \{1, \ldots, n\}} \sup_{a \in K} |c_j(a) - \int_D f_j a \, \mathsf{d}x| \leq \frac{\epsilon}{2n^{\frac{1}{p}}}.$$

Since $\sigma \in \mathsf{A}_0$, there exits some $L \in \mathbb{N}$ and neural networks $\psi_1, \ldots, \psi_n \in \mathsf{N}_L(\sigma; \mathbb{R}^d)$ such that

$$\max_{j \in \{1, \ldots, n\}} \|\psi_j - f_j\|_C \leq \frac{\epsilon}{2Qn^{\frac{1}{p}}}.$$

By setting all weights associated to the first argument to zero, we can modify each neural network ψ_j to a neural network $\psi_j \in \mathsf{N}_L(\sigma; \mathbb{R}^d \times \mathbb{R}^d)$ so that

$$\psi_j(y, x) = \psi_j(x)\mathbb{1}(y), \qquad \forall y, x \in \mathbb{R}^d.$$

Define $\kappa \in \mathsf{N}_L(\sigma; \mathbb{R}^d \times \mathbb{R}^d, \mathbb{R}^{n \times 1})$ by

$$\kappa(y, x) = [\psi_1(y, x), \dots, \psi_n(y, x)]^T.$$

Then for any $a \in K$ and $y \in \bar{D}$, we have

$$
\begin{aligned}
|F(a) - \int_D \kappa(y, x)a \, \mathrm{d}x|_p^p &= \sum_{j=1}^n |c_j(a) - \int_D \mathbb{1}(y)\psi_j(x)a(x) \, \mathrm{d}x|^p \\
&\leq 2^{p-1} \sum_{j=1}^n |c_j(a) - \int_D f_j a \, \mathrm{d}x|^p + |\int_D (f_j - \psi_j)a \, \mathrm{d}x|^p \\
&\leq \frac{\epsilon^p}{2} + 2^{p-1} n Q^p \|f_j - \psi_j\|_C^p \\
&\leq \epsilon^p
\end{aligned}
$$

and the result follows by finite dimensional norm equivalence. ∎

Lemma 33 *Suppose $D \subset \mathbb{R}^d$ is a domain and let $\{c_j\}_{j=1}^n \subset \big(C^m(\bar{D})\big)^*$ for some $m, n \in \mathbb{N}$. Define the map $F : \mathcal{A} \to \mathbb{R}^n$ by*

$$F(a) = \big(c_1(a), \dots, c_n(a)\big), \qquad \forall a \in C^m(\bar{D}).$$

Then, for any compact set $K \subset C^m(\bar{D})$, $\sigma \in \mathsf{A}_0$, and $\epsilon > 0$, there exists a number $L \in \mathbb{N}$ and neural network $\kappa \in \mathsf{N}_L(\sigma; \mathbb{R}^d \times \mathbb{R}^d, \mathbb{R}^{n \times J})$ such that

$$\sup_{a \in K} \sup_{y \in \bar{D}} |F(a) - \int_D \kappa(y, x)\big(\partial^{\alpha_1} a(x), \dots, \partial^{\alpha_J} a(x)\big) \, \mathrm{d}x|_1 \leq \epsilon$$

where $\alpha_1, \dots, \alpha_J$ is an enumeration of the set $\{\alpha \in \mathbb{N}^d : 0 \leq |\alpha|_1 \leq m\}$.

Proof The proof follows as in Lemma 32 by replacing the use of Lemmas 28 and 30 by Lemma 31. ∎

Lemma 34 *Let Assumption 10 hold. Let $\{\varphi_j\}_{j=1}^n \subset \mathcal{U}$ for some $n \in \mathbb{N}$. Define the map $G : \mathbb{R}^n \to \mathcal{U}$ by*

$$G(w) = \sum_{j=1}^n w_j \varphi_j, \qquad \forall w \in \mathbb{R}^n.$$

Then, for any compact set $K \subset \mathbb{R}^n$, $\sigma \in \mathsf{A}_{m_2}$, and $\epsilon > 0$, there exists a number $L \in \mathbb{N}$ and a neural network $\kappa \in \mathsf{N}_L(\sigma; \mathbb{R}^{d'} \times \mathbb{R}^{d'}, \mathbb{R}^{1 \times n})$ such that

$$\sup_{w \in K} \|G(w) - \int_{D'} \kappa(\cdot, x)w\mathbb{1}(x) \, \mathrm{d}x\|_{\mathcal{U}} \leq \epsilon.$$

Proof Since $K \subset \mathbb{R}^n$ is compact, there is a number $M > 1$ such that

$$\sup_{w \in K} |w|_1 \leq M.$$

If $\mathcal{U} = L^{p_2}(D')$, then density of $C_c^\infty(D')$ implies there are functions $\tilde{\psi}_1, \ldots, \tilde{\psi}_n \in C^\infty(\bar{D}')$ such that

$$\max_{j \in \{1, \ldots, n\}} \|\varphi_j - \tilde{\psi}_j\|_{\mathcal{U}} \leq \frac{\epsilon}{2nM}.$$

Similarly if $U = W^{m_2, p_2}(D')$, then density of the restriction of functions in $C_c^\infty(\mathbb{R}^{d'})$ to D' (Leoni, 2009, Theorem 11.35) implies the same result. If $\mathcal{U} = C^{m_2}(\bar{D}')$ then we set $\tilde{\psi}_j = \varphi_j$ for any $j \in \{1, \ldots, n\}$. Define $\tilde{\kappa} : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}^{1 \times n}$ by

$$\tilde{\kappa}(y, x) = \frac{1}{|D'|}[\tilde{\psi}_1(y), \ldots, \tilde{\psi}_n(y)].$$

Then, for any $w \in K$,

$$\|G(w) - \int_{D'} \tilde{\kappa}(\cdot, x)w\mathbb{1}(x)\,\mathrm{d}x\|_{\mathcal{U}} = \|\sum_{j=1}^n w_j\varphi_j - \sum_{j=1}^n w_j\tilde{\psi}_j\|_{\mathcal{U}}$$

$$\leq \sum_{j=1}^n |w_j|\|\varphi_j - \tilde{\psi}_j\|_{\mathcal{U}}$$

$$\leq \frac{\epsilon}{2}.$$

Since $\sigma \in \mathsf{A}_{m_2}$, there exists neural networks $\psi_1, \ldots, \psi_n \in \mathsf{N}_1(\sigma; \mathbb{R}^{d'})$ such that

$$\max_{j \in \{1, \ldots, n\}} \|\tilde{\psi}_j - \psi_j\|_{C^{m_2}} \leq \frac{\epsilon}{2nM(J|D'|)^{\frac{1}{p_2}}}$$

where, if $\mathcal{U} = C^{m_2}(\bar{D}')$, we set $J = 1/|D'|$ and $p_2 = 1$, and otherwise $J = |\{\alpha \in \mathbb{N}^d : |\alpha|_1 \leq m_2\}|$. By setting all weights associated to the second argument to zero, we can modify each neural network ψ_j to a neural network $\psi_j \in \mathsf{N}_1(\sigma; \mathbb{R}^{d'} \times \mathbb{R}^{d'})$ so that

$$\psi_j(y, x) = \psi_j(y)\mathbb{1}(x), \qquad \forall y, x \in \mathbb{R}^{d'}.$$

Define $\kappa \in \mathsf{N}_1(\sigma; \mathbb{R}^{d'} \times \mathbb{R}^{d'}, \mathbb{R}^{1 \times n})$ as

$$\kappa(y, x) = \frac{1}{|D'|}[\psi_1(y, x), \ldots, \psi_n(y, x)].$$

Then, for any $w \in \mathbb{R}^n$,

$$\int_{D'} \kappa(y, x)w\mathbb{1}(x)\,\mathrm{d}x = \sum_{j=1}^n w_j\psi_j(y).$$

We compute that, for any $j \in \{1, \ldots, n\}$,

$$\|\psi_j - \tilde{\psi}_j\|_{\mathcal{U}} \leq \begin{cases} |D'|^{\frac{1}{p_2}}\|\psi_j - \tilde{\psi}_j\|_{C^{m_2}}, & \mathcal{U} = L^{p_2}(D') \\ (J|D'|)^{\frac{1}{p_2}}\|\psi_j - \tilde{\psi}_j\|_{C^{m_2}}, & \mathcal{U} = W^{m_2, p_2}(D') \\ \|\psi_j - \tilde{\psi}_j\|_{C^{m_2}}, & \mathcal{U} = C^{m_2}(\bar{D}') \end{cases}$$

hence, for any $w \in K$,

$$\| \int_{D'} \kappa(y,x) w \mathbb{1}(x) \, \mathsf{d}x - \sum_{j=1}^{n} w_j \tilde{\psi}_j \|_{\mathcal{U}} \leq \sum_{j=1}^{n} |w_j| \|\psi_j - \tilde{\psi}_j\|_{\mathcal{U}} \leq \frac{\epsilon}{2}.$$

By triangle inequality, for any $w \in K$, we have

$$\|G(w) - \int_{D} \kappa(\cdot,x) w \mathbb{1}(x) \, \mathsf{d}x\|_{\mathcal{U}} \leq \|G(w) - \int_{D} \tilde{\kappa}(\cdot,x) w \mathbb{1}(x) \, \mathsf{d}x\|_{\mathcal{U}}$$

$$+ \| \int_{D} \tilde{\kappa}(\cdot,x) w \mathbb{1}(x) \, \mathsf{d}x - \int_{D} \kappa(\cdot,x) w \mathbb{1}(x) \, \mathsf{d}x\|_{\mathcal{U}}$$

$$\leq \frac{\epsilon}{2} + \| \int_{D} \kappa(\cdot,x) w \mathbb{1}(x) - \sum_{j=1}^{n} w_j \tilde{\psi}_n \|_{\mathcal{U}}$$

$$\leq \epsilon$$

as desired. ∎

Lemma 35 *Let $N, d, d', p, q \in \mathbb{N}$, $m, n \in \mathbb{N}_0$, $D \subset \mathbb{R}^p$ and $D' \subset \mathbb{R}^q$ be domains and $\sigma_1 \in \mathsf{A}_m^{\mathsf{L}}$. For any $\varphi \in \mathsf{N}_N(\sigma_1; \mathbb{R}^d, \mathbb{R}^{d'})$ and $\sigma_2, \sigma_3 \in \mathsf{A}_n$, there exists a $G \in \mathsf{NO}_N(\sigma_1, \sigma_2, \sigma_3; D, D', \mathbb{R}^d, \mathbb{R}^{d'})$ such that*

$$\varphi(w) = G(w\mathbb{1})(x), \qquad \forall w \in \mathbb{R}^d, \ \forall x \in D'.$$

.

Proof We have that

$$\varphi(x) = W_N \sigma_1(\ldots W_1 \sigma_1(W_0 x + b_0) + b_1 \ldots) + b_N, \qquad \forall x \in \mathbb{R}^d$$

where $W_0 \in \mathbb{R}^{d_0 \times d}, W_1 \in \mathbb{R}^{d_1 \times d_0}, \ldots, W_N \in \mathbb{R}^{d' \times d_{N-1}}$ and $b_0 \in \mathbb{R}^{d_0}, b_1 \in \mathbb{R}^{d_1}, \ldots, b_N \in \mathbb{R}^{d'}$ for some $d_0, \ldots, d_{N-1} \in \mathbb{N}$. By setting all parameters to zero except for the last bias term, we can find $\kappa^{(0)} \in \mathsf{N}_1(\sigma_2; \mathbb{R}^p \times \mathbb{R}^p, \mathbb{R}^{d_0 \times d})$ such that

$$\kappa_0(x,y) = \frac{1}{|D|} W_0, \qquad \forall x, y \in \mathbb{R}^p.$$

Similarly, we can find $\tilde{b}_0 \in \mathsf{N}_1(\sigma_2; \mathbb{R}^p, \mathbb{R}^{d_0})$ such that

$$\tilde{b}_0(x) = b_0, \qquad \forall x \in \mathbb{R}^p.$$

Then

$$\int_{D} \kappa_0(y,x) w \mathbb{1}(x) \, \mathsf{d}x + \tilde{b}(y) = (W_0 w + b_0) \mathbb{1}(y), \qquad \forall w \in \mathbb{R}^d, \ \forall y \in D.$$

Continuing a similar construction for all layers clearly yields the result. ∎

Appendix E.

Proof [of Theorem 8] Without loss of generality, we will assume that $D = D'$ and, by continuous embedding, that $\mathcal{A} = \mathcal{U} = C(\bar{D})$. Furthermore, note that, by continuity, it suffices to show the result for the single layer

$$\mathsf{NO} = \left\{ f \mapsto \sigma_1\left(\int_D \kappa(\cdot, y) f(y)\,\mathrm{d}y + b\right) : \kappa \in \mathsf{N}_{n_1}(\sigma_2; \mathbb{R}^d \times \mathbb{R}^d),\ b \in \mathsf{N}_{n_2}(\sigma_2; \mathbb{R}^d),\ n_1, n_2 \in \mathbb{N} \right\}.$$

Let $K \subset \mathcal{A}$ be a compact set and $(D_j)_{j=1}^\infty$ be a discrete refinement of D. To each discretization D_j associate partitions $P_j^{(1)}, \ldots, P_j^{(j)} \subseteq D$ which are pairwise disjoint, each contains a single, unique point of D_j, each has positive Lebesgue measure, and

$$\coprod_{k=1}^{j} P_j^{(k)} = D.$$

We can do this since the points in each discretization D_j are pairwise distinct. For any $\mathcal{G} \in \mathsf{NO}$ with parameters κ, b define the sequence of maps $\hat{\mathcal{G}}_j : \mathbb{R}^{jd} \times \mathbb{R}^j \to \mathcal{Y}$ by

$$\hat{\mathcal{G}}_j(y_1, \ldots, y_j, w_1, \ldots, w_j) = \sigma_1\left(\sum_{k=1}^{j} \kappa(\cdot, y_k) w_k |P_j^{(k)}| + b(\cdot)\right)$$

for any $y_k \in \mathbb{R}^d$ and $w_k \in \mathbb{R}$. Since K is compact, there is a constant $M > 0$ such that

$$\sup_{a \in K} \|a\|_{\mathcal{U}} \leq M.$$

Therefore,

$$\sup_{x \in \bar{D}} \sup_{j \in \mathbb{N}} \left| \int_D \kappa(x, y) a(y)\,\mathrm{d}y + \sum_{k=1}^{j} \kappa(x, y_k) a(y_k) |P_j^{(k)}| + 2b(x) \right| \leq 2(M|D| \|\kappa\|_{C(\bar{D} \times \bar{D})} + \|b\|_{C(\bar{D})})$$

$$:= R.$$

Hence we need only consider σ_1 as a map $[-R, R] \to \mathbb{R}$. Thus, by uniform continuity, there exists a modulus of continuity $\omega : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ which is continuous, non-negative, and non-decreasing on $\mathbb{R}_{\geq 0}$, satisfies $\omega(z) \to \omega(0) = 0$ as $z \to 0$ and

$$|\sigma_1(z_1) - \sigma_1(z_2)| \leq \omega(|z_1 - z_2|) \qquad \forall z_1, z_2 \in [-R, R]. \tag{49}$$

Let $\epsilon > 0$. Equation (49) and the non-decreasing property of ω imply that in order to show there exists $Q = Q(\epsilon) \in \mathbb{N}$ such that for any $m \geq Q$ implies

$$\sup_{a \in K} \|\hat{\mathcal{G}}_m(D_m, a|_{D_m}) - \mathcal{G}(a)\|_{\mathcal{Y}} < \epsilon,$$

it is enough to show that

$$\sup_{a \in K} \sup_{x \in \bar{D}} \left| \int_D \kappa(x, y) a(y)\,\mathrm{d}y - \sum_{k=1}^{m} \kappa(x, y_k) a(y_k) |P_m^{(k)}| \right| < \epsilon \tag{50}$$

for any $m \geq Q$. Since K is compact, we can find functions $a_1, \ldots, a_N \in K$ such that, for any $a \in K$, there is some $n \in \{1, \ldots, N\}$ such that

$$\|a - a_n\|_{C(\bar{D})} \leq \frac{\epsilon}{4|D|\|\kappa\|_{C(\bar{D}\times\bar{D})}}.$$

Since (D_j) is a discrete refinement, by convergence of Riemann sums, we can find some $q \in \mathbb{N}$ such that for any $t \geq q$, we have

$$\sup_{x\in\bar{D}} \left| \sum_{k=1}^{t} \kappa(x, y_k)|P_t^{(k)}| - \int_D \kappa(x, y)\, \mathrm{d}y \right| < |D|\|\kappa\|_{C(\bar{D}\times\bar{D})}$$

where $D_t = \{y_1, \ldots, y_t\}$. Similarly, we can find $p_1, \ldots, p_N \in \mathbb{N}$ such that, for any $t_n \geq p_n$, we have

$$\sup_{x\in\bar{D}} \left| \sum_{k=1}^{t_n} \kappa(x, y_k^{(n)})a_n(y_k^{(n)})|P_{t_n}^{(k)}| - \int_D \kappa(x, y)a_n(y)\, \mathrm{d}y \right| < \frac{\epsilon}{4}$$

where $D_{t_n} = \{y_1^{(n)}, \ldots, y_{t_n}^{(n)}\}$. Let $m \geq \max\{q, p_1, \ldots, p_N\}$ and denote $D_m = \{y_1, \ldots, y_m\}$. Note that,

$$\sup_{x\in\bar{D}} \left| \int_D \kappa(x, y)\left(a(y) - a_n(y)\right)\, \mathrm{d}y \right| \leq |D|\|\kappa\|_{C(\bar{D}\times\bar{D})}\|a - a_n\|_{C(\bar{D})}.$$

Furthermore,

$$\sup_{x\in\bar{D}} \left| \sum_{k=1}^{m} \kappa(x, y_k)\left(a_n(y_k) - a(y_k)\right)|P_m^{(k)}| \right| \leq \|a_n - a\|_{C(\bar{D})} \sup_{x\in\bar{D}} \left| \sum_{k=1}^{m} \kappa(x, y_k)|P_m^{(k)}| \right|$$

$$\leq \|a_n - a\|_{C(\bar{D})} \left(\sup_{x\in\bar{D}} \left| \sum_{k=1}^{m} \kappa(x, y_k)|P_m^{(k)}| - \int_D \kappa(x, y)\, \mathrm{d}y \right| \right.$$

$$\left. + \sup_{x\in\bar{D}} \left| \int_D \kappa(x, y)\, \mathrm{d}y \right| \right)$$

$$\leq 2|D|\|\kappa\|_{C(\bar{D}\times\bar{D})}\|a_n - a\|_{C(\bar{D})}.$$

Therefore, for any $a \in K$, by repeated application of the triangle inequality, we find that

$$\sup_{x\in\bar{D}} \left| \int_D \kappa(x, y)a(y)\, \mathrm{d}y - \sum_{k=1}^{m} \kappa(x, y_k)a(y_k)|P_m^{(k)}| \right| \leq \sup_{x\in\bar{D}} \left| \sum_{k=1}^{m} \kappa(x, y_k)a_n(y_k)|P_m^{(k)}| - \int_D \kappa(x, y)a_n(y)\, \mathrm{d}y \right|$$

$$+ 3|D|\|\kappa\|_{C(\bar{D}\times\bar{D})}\|a - a_n\|_{C(\bar{D})}$$

$$< \frac{\epsilon}{4} + \frac{3\epsilon}{4} = \epsilon$$

which completes the proof. $\blacksquare$

Appendix F.

Proof [of Theorem 11] The statement in Lemma 26 allows us to apply Lemma 22 to find a mapping $\mathcal{G}_1 : \mathcal{A} \to \mathcal{U}$ such that

$$\sup_{a \in K} \|\mathcal{G}^\dagger(a) - \mathcal{G}_1(a)\|_\mathcal{U} \leq \frac{\epsilon}{2}$$

where $\mathcal{G}_1 = G \circ \psi \circ F$ with $F : \mathcal{A} \to \mathbb{R}^J$, $G : \mathbb{R}^{J'} \to \mathcal{U}$ continuous linear maps and $\psi \in C(\mathbb{R}^J; \mathbb{R}^{J'})$ for some $J, J' \in \mathbb{N}$. By Lemma 32, we can find a sequence of maps $F_t \in \mathsf{IO}(\sigma_2; D, \mathbb{R}, \mathbb{R}^J)$ for $t = 1, 2, \ldots$ such that

$$\sup_{a \in K} \sup_{x \in \bar{D}} |(F_t(a))(x) - F(a)|_1 \leq \frac{1}{t}.$$

In particular, $F_t(a)(x) = w_t(a)\mathbb{1}(x)$ for some $w_t : \mathcal{A} \to \mathbb{R}^J$ which is constant in space. We can therefore identify the range of $F_t(a)$ with $\mathbb{R}^J$. Define the set

$$Z := \bigcup_{t=1}^\infty F_t(K) \cup F(K) \subset \mathbb{R}^J$$

which is compact by Lemma 21. Since ψ is continuous, it is uniformly continuous on Z hence there exists a modulus of continuity $\omega : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ which is continuous, non-negative, and non-decreasing on $\mathbb{R}_{\geq 0}$, satisfies $\omega(s) \to \omega(0) = 0$ as $s \to 0$ and

$$|\psi(z_1) - \psi(z_2)|_1 \leq \omega(|z_1 - z_2|_1) \qquad \forall z_1, z_2 \in Z.$$

We can thus find $T \in \mathbb{N}$ large enough such that

$$\sup_{a \in K} \omega(|F(a) - F_T(a)|_1) \leq \frac{\epsilon}{6\|G\|}.$$

Since F_T is continuous, $F_T(K)$ is compact. Since ψ is a continuous function on the compact set $F_T(K) \subset \mathbb{R}^J$ mapping into $\mathbb{R}^{J'}$, we can use any classical neural network approximation theorem such as (Pinkus, 1999, Theorem 4.1) to find an ϵ-close (uniformly) neural network. Since Lemma 35 shows that neural operators can exactly mimic standard neural networks, it follows that we can find $S_1 \in \mathsf{IO}(\sigma_1; D, \mathbb{R}^J, \mathbb{R}^{d_1}), \ldots, S_{N-1} \in \mathsf{IO}(\sigma_1; D, \mathbb{R}^{d_{N-1}}, \mathbb{R}^{J'})$ for some $N \in \mathbb{N}_{\geq 2}$ and $d_1, \ldots, d_{N-1} \in \mathbb{N}$ such that

$$\tilde{\psi}(f) := (S_{N-1} \circ \sigma_1 \circ \cdots \circ S_2 \circ \sigma_1 \circ S_1)(f), \qquad \forall f \in L^1(D; \mathbb{R}^J)$$

satisfies

$$\sup_{q \in F_T(K)} \sup_{x \in \bar{D}} |\psi(q) - \tilde{\psi}(q\mathbb{1})(x)|_1 \leq \frac{\epsilon}{6\|G\|}.$$

By construction, $\tilde{\psi}$ maps constant functions into constant functions and is continuous in the appropriate subspace topology of constant functions hence we can identity it as an element of $C(\mathbb{R}^J; \mathbb{R}^{J'})$ for any input constant function taking values in $\mathbb{R}^J$. Then $(\tilde{\psi} \circ F_T)(K) \subset \mathbb{R}^{J'}$ is compact. Therefore, by Lemma 34, we can find a neural network $\kappa \in \mathcal{N}_L(\sigma_3; \mathbb{R}^{d'} \times \mathbb{R}^{d'}, \mathbb{R}^{1 \times J'})$ for some $L \in \mathbb{N}$ such that

$$\tilde{G}(f) := \int_{D'} \kappa(\cdot, y) f(y) \, \mathrm{d}y, \qquad \forall f \in L^1(D; \mathbb{R}^{J'})$$

satisfies

$$\sup_{y \in (\tilde{\psi} \circ F_T)(K)} \|G(y) - \tilde{G}(y \mathbb{1})\|_{\mathcal{U}} \leq \frac{\epsilon}{6}.$$

Define

$$\mathcal{G}(a) := \left(\tilde{G} \circ \tilde{\psi} \circ F_T\right)(a) = \int_{D'} \kappa(\cdot, y)\left((S_{N-1} \circ \sigma_1 \circ \ldots \sigma_1 \circ S_1 \circ F_T)(a)\right)(y) \, \mathrm{d}y, \qquad \forall a \in \mathcal{A},$$

noting that $\mathcal{G} \in \mathsf{NO}_N(\sigma_1, \sigma_2, \sigma_3; D, D')$. For any $a \in K$, define $a_1 := (\psi \circ F)(a)$ and $\tilde{a}_1 := (\tilde{\psi} \circ F_T)(a)$ so that $\mathcal{G}_1(a) = G(a_1)$ and $\mathcal{G}(a) = \tilde{G}(\tilde{a}_1)$ then

$$
\begin{aligned}
\|\mathcal{G}_1(a) - \mathcal{G}(a)\|_{\mathcal{U}} &\leq \|G(a_1) - G(\tilde{a}_1)\|_{\mathcal{U}} + \|G(\tilde{a}_1) - \tilde{G}(\tilde{a}_1)\|_{\mathcal{U}} \\
&\leq \|G\||a_1 - \tilde{a}_1|_1 + \sup_{y \in (\tilde{\psi} \circ F_T)(K)} \|G(y) - \tilde{G}(y\mathbb{1})\|_{\mathcal{U}} \\
&\leq \frac{\epsilon}{6} + \|G\||(\psi \circ F)(a) - (\psi \circ F_T)(a)|_1 + \|G\||(\psi \circ F_T)(a) - (\tilde{\psi} \circ F_T)(a)|_1 \\
&\leq \frac{\epsilon}{6} + \|G\|\omega\big(|F(a) - F_T(a)|_1\big) + \|G\| \sup_{q \in F_T(K)} |\psi(q) - \tilde{\psi}(q)|_1 \\
&\leq \frac{\epsilon}{2}.
\end{aligned}
$$

Finally we have

$$\|\mathcal{G}^{\dagger}(a) - \mathcal{G}(a)\|_{\mathcal{U}} \leq \|\mathcal{G}^{\dagger}(a) - \mathcal{G}_1(a)\|_{\mathcal{U}} + \|\mathcal{G}_1(a) - \mathcal{G}(a)\|_{\mathcal{U}} \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$$

as desired.

To show boundedness, we will exhibit a neural operator $\tilde{\mathcal{G}}$ that is ϵ-close to $\mathcal{G}$ in K and is uniformly bounded by $4M$. Note first that

$$\|\mathcal{G}(a)\|_{\mathcal{U}} \leq \|\mathcal{G}(a) - \mathcal{G}^{\dagger}(a)\|_{\mathcal{U}} + \|\mathcal{G}^{\dagger}(a)\|_{\mathcal{U}} \leq \epsilon + M \leq 2M, \qquad \forall a \in K$$

where, without loss of generality, we assume that $M \geq 1$. By construction, we have that

$$\mathcal{G}(a) = \sum_{j=1}^{J'} \tilde{\psi}_j(F_T(a))\varphi_j, \qquad \forall a \in \mathcal{A}$$

for some neural network $\varphi : \mathbb{R}^{d'} \to \mathbb{R}^{J'}$. Since $\mathcal{U}$ is a Hilbert space and by linearity, we may assume that the components φ_j are orthonormal since orthonormalizing them only requires multiplying the last layers of $\tilde{\psi}$ by an invertible linear map. Therefore

$$|\tilde{\psi}(F_T(a))|_2 = \|\mathcal{G}(a)\|_{\mathcal{U}} \leq 2M, \qquad \forall a \in K.$$

Define the set $W := (\tilde{\psi} \circ F_T)(K) \subset \mathbb{R}^{J'}$ which is compact as before. We have

$$\mathrm{diam}_2(W) = \sup_{x, y \in W} |x - y|_2 \leq \sup_{x, y \in W} |x|_2 + |y|_2 \leq 4M.$$

Since $\sigma_1 \in \mathsf{BA}$, there exists a number $R \in \mathbb{N}$ and a neural network $\beta \in \mathsf{N}_R(\sigma_1; \mathbb{R}^{J'}, \mathbb{R}^{J'})$ such that

$$
\begin{aligned}
|\beta(x) - x|_2 &\leq \epsilon, \qquad \forall x \in W \\
|\beta(x)|_2 &\leq 4M, \quad \forall x \in \mathbb{R}^{J'}.
\end{aligned}
$$

Define

$$\tilde{\mathcal{G}}(a) := \sum_{j=1}^{J'} \beta_j(\tilde{\psi}(F_T(a)))\varphi_j, \qquad \forall a \in \mathcal{A}.$$

Lemmas 34 and 35 then shows that $\tilde{\mathcal{G}} \in \mathsf{NO}_{N+R}(\sigma_1, \sigma_2, \sigma_3; D, D')$. Notice that

$$\sup_{a \in K} \|\mathcal{G}(a) - \tilde{\mathcal{G}}(a)\|_{\mathcal{U}} \leq \sup_{w \in W} |w - \beta(w)|_2 \leq \epsilon.$$

Furthermore,

$$\|\tilde{\mathcal{G}}(a)\|_{\mathcal{U}} \leq \|\tilde{\mathcal{G}}(a) - \mathcal{G}(a)\|_{\mathcal{U}} + \|\mathcal{G}(a)\|_{\mathcal{U}} \leq \epsilon + 2M \leq 3M, \qquad \forall a \in K.$$

Let $a \in \mathcal{A} \setminus K$ then there exists $q \in \mathbb{R}^{J'} \setminus W$ such that $\tilde{\psi}(F_T(a)) = q$ and

$$\|\tilde{\mathcal{G}}(a)\|_{\mathcal{U}} = |\beta(q)|_2 \leq 4M$$

as desired. ∎

Appendix G.

Proof [of Theorem 13] Let $\mathcal{U} = H^{m_2}(D)$. For any $R > 0$, define

$$\mathcal{G}_R^\dagger(a) := \begin{cases} \mathcal{G}^\dagger(a), & \|\mathcal{G}^\dagger(a)\|_{\mathcal{U}} \leq R \\ \frac{R}{\|\mathcal{G}^\dagger(a)\|_{\mathcal{U}}} \mathcal{G}^\dagger(a), & \text{otherwise} \end{cases}$$

for any $a \in \mathcal{A}$. Since $\mathcal{G}_R^\dagger \to \mathcal{G}^\dagger$ as $R \to \infty$ μ-almost everywhere, $\mathcal{G}^\dagger \in L_\mu^2(\mathcal{A}; \mathcal{U})$, and clearly $\|\mathcal{G}_R^\dagger(a)\|_{\mathcal{U}} \leq \|\mathcal{G}^\dagger(a)\|_{\mathcal{U}}$ for any $a \in \mathcal{A}$, we can apply the dominated convergence theorem for Bochner integrals to find $R > 0$ large enough such that

$$\|\mathcal{G}_R^\dagger - \mathcal{G}^\dagger\|_{L_\mu^2(\mathcal{A}; \mathcal{U})} \leq \frac{\epsilon}{3}.$$

Since $\mathcal{A}$ and $\mathcal{U}$ are Polish spaces, by Lusin's theorem (Aaronson, 1997, Theorem 1.0.0) we can find a compact set $K \subset \mathcal{A}$ such that

$$\mu(\mathcal{A} \setminus K) \leq \frac{\epsilon^2}{153R^2}$$

and $\mathcal{G}_R^\dagger|_K$ is continuous. Since K is closed, by a generalization of the Tietze extension theorem (Dugundji, 1951, Theorem 4.1), there exist a continuous mapping $\tilde{\mathcal{G}}_R^\dagger : \mathcal{A} \to \mathcal{U}$ such that $\tilde{\mathcal{G}}_R^\dagger(a) = \mathcal{G}_R^\dagger(a)$ for all $a \in K$ and

$$\sup_{a \in \mathcal{A}} \|\tilde{\mathcal{G}}_R^\dagger(a)\| \leq \sup_{a \in \mathcal{A}} \|\mathcal{G}_R^\dagger(a)\| \leq R.$$

Applying Theorem 11 to $\tilde{\mathcal{G}}_R^\dagger$, we find that there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that

$$\sup_{a \in K} \|\mathcal{G}(a) - \mathcal{G}_R^\dagger(a)\|_{\mathcal{U}} \leq \frac{\sqrt{2}\epsilon}{3}$$

and

$$\sup_{a \in \mathcal{A}} \|\mathcal{G}(a)\|_{\mathcal{U}} \leq 4R.$$

We then have

$$\|\mathcal{G}^\dagger - \mathcal{G}\|_{L^2_\mu(\mathcal{A};\mathcal{U})} \leq \|\mathcal{G}^\dagger - \mathcal{G}^\dagger_R\|_{L^2_\mu(\mathcal{A};\mathcal{U})} + \|\mathcal{G}^\dagger_R - \mathcal{G}\|_{L^2_\mu(\mathcal{A};\mathcal{U})}$$

$$\leq \frac{\epsilon}{3} + \left(\int_K \|\mathcal{G}^\dagger_R(a) - \mathcal{G}(a)\|^2_{\mathcal{U}} \, \mathsf{d}\mu(a) + \int_{\mathcal{A}\backslash K} \|\mathcal{G}^\dagger_R(a) - \mathcal{G}(a)\|^2_{\mathcal{U}} \, \mathsf{d}\mu(a) \right)^{\frac{1}{2}}$$

$$\leq \frac{\epsilon}{3} + \left(\frac{2\epsilon^2}{9} + 2 \left(\sup_{a \in \mathcal{A}} \|\mathcal{G}^\dagger_R(a)\|^2_{\mathcal{U}} + \|\mathcal{G}(a)\|^2_{\mathcal{U}} \right) \mu(\mathcal{A} \backslash K) \right)^{\frac{1}{2}}$$

$$\leq \frac{\epsilon}{3} + \left(\frac{2\epsilon^2}{9} + 34R^2 \mu(\mathcal{A} \backslash K) \right)^{\frac{1}{2}}$$

$$\leq \frac{\epsilon}{3} + \left(\frac{4\epsilon^2}{9} \right)^{\frac{1}{2}}$$

$$= \epsilon$$

as desired. ∎