Neural Operator: Learning Maps Between Function Spaces

Nikola Kovachki* NKOVACHKI@CALTECH.EDU *Caltech*
Zongyi Li* ZONGYILI@CALTECH.EDU *Caltech*
Burigede Liu BGL@CALTECH.EDU *Caltech*
Kamyar Azizzadenesheli KAMYAR@PURDUE.EDU *Purdue University*
Kaushik Bhattacharya BHATTA@CALTECH.EDU *Caltech*
Andrew Stuart ASTUART@CALTECH.EDU *Caltech*
Anima Anandkumar ANIMA@CALTECH.EDU *Caltech*

Editor:

Abstract

The classical development of neural networks has primarily focused on learning mappings between finite dimensional Euclidean spaces or finite sets. We propose to generalize neural networks so that they can learn operators mapping between infinite dimensional function spaces. We formulate the approximation of operators by composition of a class of linear integral operators and nonlinear activation functions, so that the composed operator can approximate complex nonlinear operators. We prove a universal approximation theorem for our construction. Furthermore, we propose four classes of operator parametrizations: graph-based operators, low-rank operators, multipole graph-based operators, and Fourier operators and describe efficient algorithms for computing with each one. The proposed neural operators are resolution-invariant: they share the same network parameters between different discretizations of the underlying function space and can be used for zero-shot super-resolutions. Numerically, the proposed models show superior performance compared to existing machine learning based methodologies on Burgers' equation, Darcy flow, and the Navier-Stokes equation, while being several order of magnitude faster compared to conventional PDE solvers.

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

1. Introduction

Learning mappings between infinite-dimensional function spaces is a challenging problem with numerous applications across various disciplines. Examples include models in science and engineering, differential equations, and also tasks in computer vision. In particular, any map where either the input or the output space, or both, can be infinite-dimensional. The possibility of learning such mappings opens up a new class of problems in the design of neural networks with widespread applicability. New ideas are required to build on traditional neural networks which are mappings between finite-dimensional Euclidean spaces.

We formulate a new class of deep neural network based models, called neural operators, which map between spaces of functions on bounded domains $D \subset \mathbb{R}^d$, $D' \subset \mathbb{R}^{d'}$. Such models, once trained, have the property of being discretization invariant i.e. sharing the same network parameters between different discretizations of the underlying functional data. In contrast, standard neural network architectures depend heavily on this discretization as a new model with new parameters is needed to achieve the same error for differently discretized data if at all possible. We demonstrate, numerically, that the same neural operator can achieve a constant error for any discretization of the data while standard feed-forward and convolutional neural networks cannot. We further develop an approximation theory for the neural operator, proving its ability to approximate linear and non-linear operators arbitrary well.

We experiment with the proposed model within the context of partial differential equations (PDEs), where we study various solution operators or flow maps arising from the PDE model. In particular, we investigate mappings between functions spaces where the input data can be, for example, the initial condition, boundary condition, or coefficient function, and the output data is the respective solution. We perform numerical experiments with operators arising from the one-dimensional Burgers' Equation, the two-dimensional steady sate of Darcy Flow, and the two-dimensional Navier-Stokes Equation (Lemarié-Rieusset, 2018).

1.1 Background and Context

PDEs. "Differential equations [...] represent the most powerful tool humanity has ever created for making sense of the material world." Strogatz (2009). Over the past few decades, significant progress has been made on 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:

- identifying the governing model for complex systems;
- efficiently solving large-scale non-linear systems of equations.

Identifying/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 fracture 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 too simplistic to be informative. The possibility of acquiring such knowledge from data can revolutionize these fields. Second, solving complicated non-linear 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 from such computations to design fast approximate solvers holds great potential for accelerating the advent of science.

Learning PDE solution operators. Neural networks have the potential to overcome these challenges when designed in a way that allows for the emulation of mappings between function spaces (Lu et al., 2019; Bhattacharya et al., 2020; Nelsen and Stuart, 2020; Li et al., 2020a,b,c; Patel et al., 2021; Opschoor et al., 2020; Schwab and Zech, 2019; O'Leary-Roseberry et al., 2020). In PDE applications, the governing 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

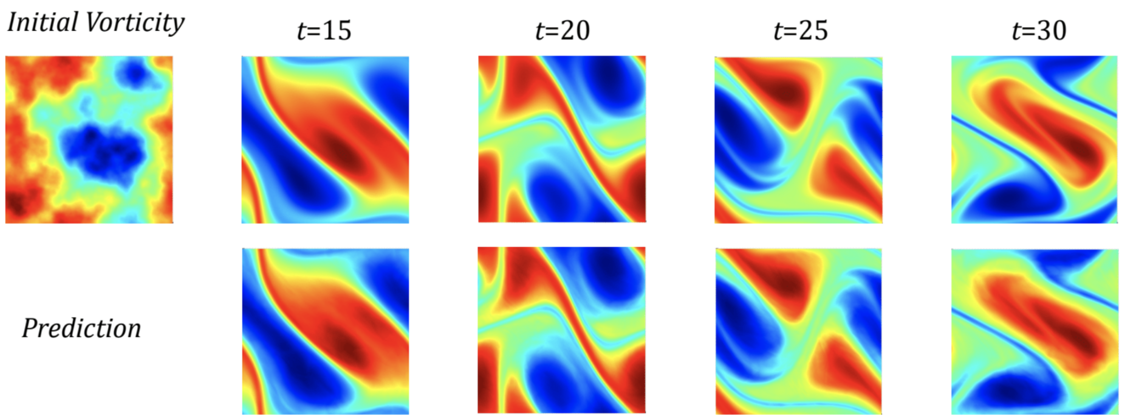


Figure 1: Zero-shot super-resolution: Vorticity field of the solution to the two-dimensional Navier-Stokes equation with viscosity 10^4; Ground truth on top and prediction on bottom. The model is trained on data that is discretized on a uniform 64×64 spatial grid and on a 20-point uniform temporal grid. The model is evaluated against data that is discretized on a uniform 256×256 spatial grid and a 80-point uniform temporal grid (see Section 7.3.1).

spatial domain, or by means of spectral domain multiplication with convolution being 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. For spectral domain calculations, there are well-developed tools that exist for approximating the continuum (Boyd, 2001; Trefethen, 2000). For these reasons, neural networks that build in non-locality via integral operators or spectral domain calculations are natural. This is governing framework for our work aimed at designing mesh invariant neural network approximations for the solution operators of PDEs.

1.2 Our Contribution

Neural Operators. We introduce the concept of neural operators by generalizing standard feed-forward neural networks to learn mappings between infinite-dimensional spaces of functions defined on bounded domains of $\mathbb{R}^d$. The non-local component of the architecture is instantiated through either a parameterized integral operator or through multiplication in the spectral domain. The methodology leads to the following contributions.

1. We propose neural operators a concept which generalizes neural networks that map between finite-dimensional Euclidean spaces to neural networks that map between infinite-dimensional function spaces.

2. By construction, our architectures share the same parameters irrespective of the discretization used on the input and output spaces done for the purposes of computation. Consequently, neural operators are capable of zero-shot super-resolution as demonstrated in Figure 1.

3. We develop approximation theorems which guarantee that neural operators are expressive enough to approximate any measurable operator mapping between a large family of possible Banach spaces arbitrarily well.

4. We propose four methods for practically implementing the neural operator framework: graph-based operators, low-rank operators, multipole graph-based operators, and Fourier operators. Specifically, 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. Furthermore, we study 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.

5. 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 with viscosity $1\text{e}{+}3$ and 8% error with viscosity $1\text{e}{+}4$.

6. The Fourier neural operator 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 (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.

In this work, we propose the neural operator models to learn mesh-free, infinite-dimensional operators with neural networks. Compared to previous methods that we will discuss in the related work section 1.3, the neural operator remedies the mesh-dependent nature of standard finite-dimensional approximation methods such as convolutional neural networks by producing a single set of network parameters that may be used with different discretizations. It also has the ability to transfer solutions between meshes. Furthermore, the neural operator needs to be trained only once, and obtaining a solution for a new instance of the parameter requires only a forward pass of the network, alleviating the major computational issues incurred in physics-informed neural network methods Raissi et al. (2019). Lastly, the neural operator requires no knowledge of the underlying PDE, only data. While previous continuous methods have not yielded efficient numerical algorithms that can parallel the success of convolutional or recurrent neural networks in the finite-dimensional setting due to the cost of evaluating integral operators, our work alleviates this issue through the use kernel approximation methods and fast transform algorithms.

1.3 Related Works

We outline the major neural network-based approaches for the solution of PDEs. To make the discussion concrete, we will consider the family of PDEs in the form

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

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 $\mathsf{L} : \mathcal{A} \to \mathcal{L}(\mathcal{U}; \mathcal{U}^*)$ 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 $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ 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 $\nabla \cdot (a\nabla)$. 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 $\{a_j, u_j\}_{j=1}^N$ that are used to train the model (see Section 2.1).

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). Khoo et al. (2017) 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, we show, for our method, both invariance of the error to grid resolution, and the ability to transfer the solution between meshes in Section 7. 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.

Physics Informed Neural Networks (PINNs). A different approach is to directly parameterize the solution u as a neural network $u : \bar{D} \times \Theta \to \mathbb{R}$ (E and Yu, 2018; Raissi et al., 2019; Bar and Sochen, 2019; Smith et al., 2020; Pan and Duraisamy, 2020). 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, 2020; Opschoor et al., 2020; Schwab and Zech, 2019; O'Leary-Roseberry et al., 2020; Lu et al., 2019), is, to the best of our knowledge, amongst the first practical supervised learning methods designed to learn maps between infinite-dimensional spaces. It remedies 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. Moreover, it need only 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; Weinan, 2017). The concept of defining neural networks in infinite-dimensional spaces is a central problem that 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.

Nyström approximation, GNNs, and graph neural operators (GNOs). The graph neural operators (Section 5.1) 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 which have seen 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 5.2 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) similarly proposes to use neural networks 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, making our low-rank operator resolution invariant. On the other hand, the number of parameters of the networks in DeepONet(s) depend on the resolution of the data; see Section 3.2 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 to 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 5.3, 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. In theory, they appear in the proof of the universal approximation theorem (Hornik et al., 1989) and, 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 5.4, 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.

Paper outline. In Section 2, we define the general operator learning problem, which is not limited to PDEs. In Section 3, we define the general framework in term of kernel integral operators and relate our proposed approach to existing methods in the literature. In Section 4, we study the approximation theoretic properties of the proposed methodology. In Section 5, we define four different ways of efficiently computing with neural operators: graph-based operators (GNO), low-rank operators (LNO), multipole graph-based operators (MGNO), and Fourier operators (FNO). In Section 6 we define four partial differential equations which serve as a testbed of various problems which we study numerically. In Section 7, we show the numerical results for our four approximation methods on the four test problems, and we discuss and compare the properties of each methods. In Section 8 we conclude the work, discuss potential limitations and future directions.

2. Learning Operators

2.1 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 $\{a_j, u_j\}_{j=1}^N$ where $a_j \sim \mu$ are i.i.d. samples drawn from some probability measure μ supported on $\mathcal{A}$ and $u_j = \mathcal{G}^\dagger(a_j)$ 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 $L^2_\mu(\mathcal{A};\mathcal{U})$ Bochner norm of the approximation

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

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

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

which directly parallels the classical finite-dimensional setting (Vapnik, 1998).

In Section 4, we show that given any desired error tolerance, there exists $p \in \mathbb{N}$ and an associated parameter $\theta^\dagger \in \mathbb{R}^p$, so that (3) is controlled. 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_j, u_j are discretized.

2.2 Discretization

Since our data a_j and u_j 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 function are real-valued. Let $D_j = \{x_j^{(1)}, \dots, x_j^{(n_j)}\} \subset D$ be a n_j-point discretization of the domain D and assume we have observations $a_j|_{D_j}, u_j|_{D_j} \in \mathbb{R}^{n_j}$, 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 $a \sim \mu$. That is to say that our approach is independent of the discretization D_j and therefore a true function space method; we verify this claim numerically by showing invariance of the error as $n_j \to \infty$. Such a property is highly desirable as it allows a transfer of solutions between different grid geometries and discretization sizes with a single architecture which 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.

3. Proposed Architecture

3.1 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 $d_{v_0} > d_a$ hence this is a lifting operation performed by a fully local operator.

2. **Iterative kernel integration**: For $t = 0, \dots, T-1$, 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 composed with a fixed, pointwise non-linearity. 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. Similarly to the first step, we usually pick $d_{v_T} > d_u$ hence this is a projection step performed by a fully local operator.

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

$$\mathcal{G}_\theta \coloneqq \mathcal{Q} \circ \sigma_T(W_{T-1} + \mathcal{K}_{T-1} + b_{T-1}) \circ \cdots \circ \sigma_1(W_0 + \mathcal{K}_0 + b_0) \circ \mathcal{P} \tag{5}$$

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 $d_{v_0}, \dots, d_{v_T}$ as well as the input dimensions $d_1, \dots, d_{T-1}$ and domains of definition $D_1, \dots, D_{T-1}$ 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 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 (5) and a standard feed-forward neural network is that all operations are directly defined in function space 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. This is then learned by successively approximating using integral kernel operators composed with a local non-linearity. 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 [CITE], we further add a local linear operator acting on the output of the previous layer before applying the non-linearity. 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 (5). For the first, let $\kappa^{(t)} \in C(D_{t+1} \times D_t; \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}})$. Then we define $\mathcal{K}_t$ by

$$(\mathcal{K}_t(v_t))(x) = \int_{D_t} \kappa^{(t)}(x,y) v_t(y) \, \mathrm{d}\nu_t(y) \qquad \forall x \in D_{t+1} \tag{6}$$

where ν_t is Borel measure on D_t. Normally, we take ν_t to simply be the Lebesgue measure on $\mathbb{R}^{d_t}$ but, as discussed in Section 5, 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 (6) defines the basic form of the neural operator and is the one we analyze in Section 4 and study the most numerically in Section 7.

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

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

where $\Pi_t^D : D_t \to D$ are fixed mappings. We have found numerically that, for certain PDE problems, the form (7) outperforms (6) due to the strong dependence of the solution u on the parameters a. Indeed, if we think of (5) 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

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

where $\Pi_t : D_{t+1} \to D_t$ are fixed mappings. Note that, in contrast to (6) and (7), the integral operator (8) is non-linear 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 3.3 that neural operators are a continuous 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 (5). For simplicity, we choose the integral kernel operator given by (6), but note that any of the other definitions work analogously. We then have that a single hidden layer update is given by

$$v_{t+1}(x) = \sigma_{t+1}\left(W_t v_t(\Pi_t(x)) + \int_{D_t} \kappa^{(t)}(x,y) v_t(y) \, \mathrm{d}\nu_t(y) + b_t(x)\right) \qquad \forall x \in D_{t+1} \tag{9}$$

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 $\mathbb{R}^d$. We then have that (5) becomes

$$(\mathcal{G}_\theta(a))(x) = \mathcal{Q}\left(\int_D \kappa^{(1)}(x,y)\sigma\left(W_0\mathcal{P}(a(y)) + \int_D \kappa^{(0)}(y,z)\mathcal{P}(a(z)) \, \mathrm{d}z + b_0(y)\right) \mathrm{d}y\right) \tag{10}$$

for any $x \in D'$. In this example, $\mathcal{P} \in C(\mathbb{R}^{d_a}; \mathbb{R}^{d_{v_0}})$, $\kappa^{(0)} \in C(D \times D; \mathbb{R}^{d_{v_1} \times d_{v_0}})$, $b_0 \in C(D; \mathbb{R}^{d_{v_1}})$, $W_0 \in \mathbb{R}^{d_{v_1} \times d_{v_0}}$, $\kappa^{(0)} \in C(D' \times D; \mathbb{R}^{d_{v_2} \times d_{v_1}})$, 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. In Section 5, we discuss various choices for parametrizing the kernels and 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 in all of our numerical experiments in Section 7. Similarly, when dealing with a smoothing operators, 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 interested 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 per-problem basis.

3.2 DeepONets are Neural Operators

We will now draw a parallel between the recently proposed DeepONet architecture in Lu et al. (2019) and our neural operator framework. In fact, we will show that DeepONet(s) are a special case of a single hidden layer neural operator construction. To that end, we work with (10) 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 $d_{v_0} = d_{v_1} = d_{v_2} = n \in \mathbb{N}$. Define $\mathcal{P} : \mathbb{R} \to \mathbb{R}^n$ by $\mathcal{P}(x) = (x, \dots, x)$ and $\mathcal{Q} : \mathbb{R}^n \to \mathbb{R}$ by $\mathcal{Q}(x) = x_1 + \cdots + x_n$. Furthermore let $\kappa^{(1)} : D' \times D \to \mathbb{R}^{n \times n}$ be given as $\kappa^{(1)}(x, y) = \operatorname{diag}(\kappa_1^{(1)}(x, y), \dots, \kappa_n^{(1)}(x, y))$ for some $\kappa_1^{(1)}, \dots \kappa_n^{(1)} : D' \times D \to \mathbb{R}$. Similarly let $\kappa^{(0)} : D \times D \to \mathbb{R}^{n \times n}$ be given as $\kappa^{(0)}(x, y) = \operatorname{diag}(\kappa_1^{(0)}(x, y), \dots, \kappa_n^{(0)}(x, y))$ for some $\kappa_1^{(0)}, \dots \kappa_n^{(0)} : D \times D \to \mathbb{R}$. Then (10) becomes

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

where $b(y) = (b_1(y), \dots, b_n(y))$ for some $b_1, \dots, b_n : D \to \mathbb{R}$. Choose each $\kappa_j^{(1)}(x, y) = w_j(y)\varphi_j(x)$ for some $w_1, \dots, w_n : D \to \mathbb{R}$ and $\varphi_1, \dots, \varphi_n : D' \to \mathbb{R}$ then we obtain

$$(\mathcal{G}_\theta(a))(x) = \sum_{j=1}^{n} G_j(a)\varphi_j(x) \tag{11}$$

where $G_1, \dots, G_n : \mathcal{A} \to \mathbb{R}$ are functionals defined as

$$G_j(a) = \int_D w_j(y) \sigma \left(\int_D \kappa_j^{(0)}(y, z) a(z) \, \mathrm{d}z + b_j(y) \right) \, \mathrm{d}y. \tag{12}$$

The set of maps $\varphi_1, \dots, \varphi_n$ form the *trunk net* while the functionals $G_1, \dots, G_n$ form the *branch net* of a DeepONet. The only difference between DeepONet(s) and (11) is the parametrization used for the functionals $G_1, \dots, G_n$. Following Chen and Chen (1995), DeepONet(s) define the functional G_j as maps between finite dimensional spaces. Indeed, they are viewed as $G_j : \mathbb{R}^q \to \mathbb{R}$ and defined to map pointwise evaluations $(a(x_1), \dots, a(x_q))$ of a for some set of points $x_1, \dots, x_q \in D$. We note that, in practice, this set of evaluation points is not known *a priori* and could potentially be very large. The proof of Theorem 4 shows that if we instead define the functionals G_j directly in function space via (12), the set of evaluation points can be discovered *automatically* by learning the kernels $\kappa_j^{(0)}$ and weighting functions w_j. Indeed we show that (12) can approximate the functionals defined by DeepONet(s) arbitrary well therefore making DeepONet(s) a special case of neural operators. Furthermore (12) is consistent in function space as the number of parameters used to define each G_j is *independent* of any discretization that may be used for a, while this is not true in the DeepONet case as the number of parameters grow as we refine the discretization of a. We demonstrate numerically in Section 7 that the error incurred by DeepONet(s) grows with the discretization of a while it is remains constant for neural operators.

We point out that parametrizations of the form (11) fall within the class of *linear* approximation methods since the non-linear space $\mathcal{G}^\dagger(\mathcal{A})$ is approximated by the linear space $\operatorname{span}\{\varphi_1, \dots, \varphi_n\}$ DeVore (1998). The quality of the best possible linear approximation to a non-linear 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 non-linear 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 (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 (10) and more generally (5), constitute a form of *non-linear* approximation. The benefits of non-linear 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 non-linear parametrizations such as (10) outperform linear one such as DeepONets or the low-rank method introduced in Section 5.2. We acknowledge, however, that the theory presented in Section 4 is based on the reduction (11) and therefore does not capture the benefits of the non-linear approximation. Furthermore, in practice, we have found that deeper architectures than (10) (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. We leave both characterizations for interesting future work.

3.3 Transformers are 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 our removing words from the sentence simply corresponds to refining or coarsening the discretization of $[0, 1]$. We will now make this intuition precise.

We will show that by making a particular choice of the non-linear integral kernel operator (8) 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 $t = 0, \dots, T$, the bias term is zero, and $W = I$ is the identity. Further, to simplify notation, we will drop the layer index t from (9) and, employing (8), obtain

$$u(x) = \sigma\left(v(x) + \int_D \kappa_v(x, y, v(x), v(y))v(y)\,\mathrm{d}y\right) \qquad \forall x \in D \tag{13}$$

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 and not simply its pointwise values. While this is not explicitly done in (8), 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 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 (13) becomes

$$u(x) = \sigma \left(v(x) + \int_D \frac{\exp\left(\frac{\langle Av(x), Bv(y)\rangle}{\sqrt{m}}\right)}{\int_D \exp\left(\frac{\langle Av(s), Bv(y)\rangle}{\sqrt{m}}\right) \, \mathrm{d}s} Rv(y) \, \mathrm{d}y \right) \qquad \forall x \in D. \tag{14}$$

Equation (14) 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 $\{x_1, \dots, x_k\} \subset D$ 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 $j = 1, \dots, k$. Approximating the inner-integral in (14) 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 (14) and using the same approximation for the outer integral yields

$$u_j = \sigma \left(v_j + \sum_{q=1}^{k} \frac{\exp\left(\frac{\langle Av_j, Bv_q\rangle}{\sqrt{m}}\right)}{\sum_{l=1}^{k} \exp\left(\frac{\langle Av_l, Bv_q\rangle}{\sqrt{m}}\right)} Rv_q \right), \qquad j = 1, \dots, k. \tag{15}$$

Equation (15) can be viewed as a Nyström approximation of (14). Define the vectors $z_q \in \mathbb{R}^k$ by

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

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

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

Then we may re-write (15) as

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

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

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

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 straightforwardly be adapted to the continuum setting and incorporated into (14). Furthermore multi-headed self-attention can be realized by simply allowing κ_v to be a sum of 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.

While we have not rigorously experimented with using transformer architectures for the problems outlined in Section 6, we have found, in initial tests, that they perform worse, are slower, and are more memory expensive than neural operators using (6)-(8). Their high computational complexity is evident from (14) as we must evaluate a *nested* integral of v for each $x \in D$. Recently more efficient transformer architectures have been proposed e.g. (Choromanski et al., 2020) and some have been applied to computer vision tasks. We leave as interesting future work experimenting and comparing these architectures to the neural operator both on problems in scientific computing and more traditional machine learning tasks.

4. Approximation Theory

In this section, we prove our main approximation result Theorem 1: given any $\epsilon > 0$, there exists a neural operator $\mathcal{G}_\theta$ that is an ϵ-close approximation to $\mathcal{G}^\dagger$. The approximation is close in the topology of the $L^2_\mu(\mathcal{A};\mathcal{U})$ Bochner norm (3). In order to state the theorem, we need the notion of a *zero-extended* neural operator originally introduced, for standard neural networks, in Bhattacharya et al. (2020). Given $M > 0$ define the zero- extended neural operator

$$\mathcal{G}_{M,\theta}(a) = \begin{cases} \mathcal{G}_\theta(a), & \|a\|_{\mathcal{A}} \leq M \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

for any $a \in \mathcal{A}$ where $\mathcal{G}_\theta$ is a neural operator of the form (5). For Theorem 1 to remain valid, any bounded operator on $\mathcal{A}$ can replace zero in (16); we simply choose zero for convenience. We also note that this zero-extension does not have negative practical implications since M can always be chosen to be arbitrarily far away from the upper bound on the training set. Its purpose is only to control the unbounded set $\{a \in \mathcal{A} : \|a\|_{\mathcal{A}} > M\}$ which will never be seen unless the model is used in a scenario where the size of inputs grows unboundedly. In such scenarios, Theorem 1 is not applicable, however, we note that such applications do not often occur in practice.

We further remark that the zero-extension can be avoided if the activation function for the kernel network is chosen to be ReLU or another activation function for which global constructions over $\mathbb{R}^n$ can be made. In particular, one can design a ReLU network which approximates the identity over any closed ball in $\mathbb{R}^n$ and remains bounded outside it. This network can then be composed with the functionals in the finite dimensional approximation (11) to produce an approximation bounded on all of $\mathcal{A}$ thus avoiding the need for the zero-extension (16). For details we refer the reader to Lanthaler et al. (2021). We choose to state the result using the zero-extension so that it remains valid for any neural network used to approximate the kernels.

We remark that the conditions placed on σ in Theorem 1 are not restrictive and satisfied by most activation functions used in practice; see Appendix B for a further discussion. Furthermore, we note that extensions to vector-valued functions follow immediately by repeatedly applying the result to each component function.

Theorem 1 *Let $D \subset \mathbb{R}^d$, $D' \subset \mathbb{R}^{d'}$ be compact domains and suppose μ is a probability measure supported on $C(D;\mathbb{R})$. Furthermore let $\mathcal{U}$ be a separable Banach space of real-valued functions on D' such that $C(D';\mathbb{R}) \subset \mathcal{U}$ is dense, and suppose there exists a constant $C > 0$ such that $\|u\|_{\mathcal{U}} \leq C\|u\|_{L^\infty(D';\mathbb{R})}$ for all $u \in \mathcal{U} \cap L^\infty(D';\mathbb{R})$. Assume $\sigma : \mathbb{R} \to \mathbb{R}$ is Hölder continuous and of the Tauber-Wiener class. Then for any μ-measurable $\mathcal{G}^\dagger \in L^2_\mu(C(D;\mathbb{R});\mathcal{U})$ and $\epsilon > 0$, there exists $M_0 > 0$ such that, for any $M \geq M_0$, there exists a neural operator $\mathcal{G}_\theta$ of the form* (5) *such that the zero-extended neural operator $\mathcal{G}_{M,\theta}$ satisfies*

$$\|\mathcal{G}^\dagger - \mathcal{G}_{M,\theta}\|_{L^2_\mu(C(D;\mathbb{R});\mathcal{U})} < \epsilon.$$

Proof Let $R > 0$ and define

$$\mathcal{G}^\dagger_R(a) = \begin{cases} \mathcal{G}^\dagger(a), & \|\mathcal{G}^\dagger(a)\|_{\mathcal{U}} \leq R \\ \frac{R}{\|\mathcal{G}^\dagger(a)\|_{\mathcal{U}}}\mathcal{G}^\dagger(a), & \|\mathcal{G}^\dagger(a)\|_{\mathcal{U}} > R. \end{cases}$$

Since $\mathcal{G}^\dagger_R \to \mathcal{G}^\dagger$ as $R \to \infty$ μ-almost everywhere, $\mathcal{G}^\dagger \in L^2_\mu(C(D;\mathbb{R});\mathcal{U})$, and clearly $\|\mathcal{G}^\dagger_R(a)\|_{\mathcal{U}} \leq \|\mathcal{G}^\dagger(a)\|_{\mathcal{U}}$ for any $a \in C(D;\mathbb{R})$, we can apply the dominated convergence theorem for Bochner integrals to find $R > 0$ large enough such that

$$\|\mathcal{G}^\dagger_R - \mathcal{G}^\dagger\|_{L^2_\mu(C(D;\mathbb{R});\mathcal{U})} < \epsilon.$$

It therefore suffices to approximate $\mathcal{G}^\dagger_R$ which is uniformly bounded on $C(D;\mathbb{R})$. Since $\mathcal{G}^\dagger_R$ is μ-measurable, Lusin's theorem , see e.g. (Aaronson, 1997, Theorem 1.0.0), implies that there exists a compact set $K \subset C(D;\mathbb{R})$ such that $\mathcal{G}^\dagger_R|_K$ is continuous and $\mu(C(D;\mathbb{R}) \setminus K) < \epsilon$, noting that we can apply Lusin's theorem since $C(D;\mathbb{R})$ and $\mathcal{U}$ are both Polish spaces. Since $\mathcal{G}^\dagger_R|_K$ is continuous, $\mathcal{G}^\dagger_R(K) \subset \mathcal{U}$ is compact hence we can apply Lemma 7 to find a number $n \in \mathbb{N}$, continuous, linear, functionals $G_1, \dots, G_n \in C(\mathcal{G}^\dagger_R(K);\mathbb{R})$ and functions $\varphi_1, \dots, \varphi_n \in C(D';\mathbb{R})$ such that

$$\sup_{a \in K} \Big\|\mathcal{G}^\dagger_R(a) - \sum_{j=1}^n G_j(\mathcal{G}^\dagger_R(a))\varphi_j\Big\|_{\mathcal{U}} = \sup_{a \in K} \|\mathcal{G}^\dagger_R(a) - (P_n \circ \mathcal{G}^\dagger_R)(a)\|_{\mathcal{U}} < \epsilon$$

where we define the operator $P_n : \mathcal{G}^\dagger_R(K) \to C(D';\mathbb{R})$ by

$$P_n(v) = \sum_{j=1}^n G_j(v)\varphi_j, \qquad \forall v \in \mathcal{G}^\dagger_R(K).$$

P_n is continuous by continuity of the functionals $G_1, \dots, G_n$ hence $(P_n \circ \mathcal{G}^\dagger_R) : K \subset C(D;\mathbb{R}) \to C(D';\mathbb{R})$ is a continuous mapping so we can apply Theorem 4 to find a neural operator $\mathcal{G}_\theta$ such that

$$\sup_{a \in K} \|\mathcal{G}_\theta(a) - (P_n \circ \mathcal{G}^\dagger_R)(a)\|_{C(D';\mathbb{R})} < \frac{\epsilon}{C}.$$

Therefore,

$$\sup_{a \in K} \|\mathcal{G}_\theta(a) - (P_n \circ \mathcal{G}^\dagger_R)(a)\|_{\mathcal{U}} \leq C \sup_{a \in K} \|\mathcal{G}_\theta(a) - (P_n \circ \mathcal{G}^\dagger_R)(a)\|_{C(D';\mathbb{R})} < \epsilon,$$

by noting that since $\mathcal{G}_\theta(a) - (P_n \circ \mathcal{G}^\dagger_R)(a) \in C(D';\mathbb{R})$, the norm on $L^\infty(D';\mathbb{R})$ coincides with the one on $C(D';\mathbb{R})$. Since K is compact, we can find $M_0 > 0$ large enough such that $K \subset B_M \subset C(D;\mathbb{R})$ for any $M \geq M_0$, see e.g. (Ward, 1974, Theorem 1), where we denote $B_M = \{a \in C(D;\mathbb{R}) : \|a\|_{C(D;\mathbb{R})} \leq M\}$. Define, for any $a \in C(D;\mathbb{R})$,

$$\mathcal{G}_{M,\theta}(a) = \begin{cases} \mathcal{G}_\theta(a), & \|a\|_{C(D;\mathbb{R})} \leq M \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 4 implies that we can choose $\mathcal{G}_\theta$ such that

$$\sup_{a \in B_M} \|\mathcal{G}_\theta(a)\|_{C(D';\mathbb{R})} \leq L$$

for some constant $L > 0$. Therefore,

$$\sup_{a \in B_M} \|\mathcal{G}_\theta(a)\|_{\mathcal{U}} \leq C \sup_{a \in B_M} \|\mathcal{G}_\theta(a)\|_{C(D';\mathbb{R})} \leq CL.$$

Repeated use of the triangle inequality shows

$$\begin{aligned}
\|\mathcal{G}_{M,\theta} - \mathcal{G}^\dagger\|_{L^2_\mu(C(D;\mathbb{R});\mathcal{U})} &\leq \|\mathcal{G}_{M,\theta} - \mathcal{G}^\dagger_R\|_{L^2_\mu(C(D;\mathbb{R});\mathcal{U})} + \|\mathcal{G}^\dagger_R - \mathcal{G}^\dagger\|_{L^2_\mu(C(D;\mathbb{R});\mathcal{U})} \\
&< \epsilon + \|\mathcal{G}_{M,\theta} - P_n \circ \mathcal{G}^\dagger_R\|_{L^2_\mu(C(D;\mathbb{R});\mathcal{U})} + \|P_n \circ \mathcal{G}^\dagger_R - \mathcal{G}^\dagger_R\|_{L^2_\mu(C(D;\mathbb{R});\mathcal{U})} \\
&= \epsilon + \|\mathcal{G}_{M,\theta} - P_n \circ \mathcal{G}^\dagger_R\|_{L^2_\mu(K;\mathcal{U})} + \|\mathcal{G}_{M,\theta} - P_n \circ \mathcal{G}^\dagger_R\|_{L^2_\mu(C(D;\mathbb{R})\setminus K;\mathcal{U})} \\
&\quad + \|P_n \circ \mathcal{G}^\dagger_R - \mathcal{G}^\dagger_R\|_{L^2_\mu(K;\mathcal{U})} + \|P_n \circ \mathcal{G}^\dagger_R - \mathcal{G}^\dagger_R\|_{L^2_\mu(C(D;\mathbb{R})\setminus K;\mathcal{U})} \\
&< \epsilon + \mu(K)^{\frac{1}{2}}\epsilon + \mu(K)^{\frac{1}{2}}\epsilon + \|P_n \circ \mathcal{G}^\dagger_R - \mathcal{G}^\dagger_R\|_{L^2_\mu(C(D;\mathbb{R})\setminus K;\mathcal{U})} \\
&\quad + \|\mathcal{G}_\theta - P_n \circ \mathcal{G}^\dagger_R\|_{L^2_\mu(B_M\setminus K;\mathcal{U})} + \|\mathcal{G}_\theta - P_n \circ \mathcal{G}^\dagger_R\|_{L^2_\mu(C(D;\mathbb{R})\setminus B_M;\mathcal{U})} \\
&\leq 3\epsilon + 2\mu(C(D;\mathbb{R}) \setminus K)^{\frac{1}{2}} R + \mu(B_M \setminus K)^{\frac{1}{2}}(CL + R) \\
&\quad + \mu(C(D;\mathbb{R}) \setminus B_M)^{1/2} R \\
&< 3\epsilon + \epsilon^{\frac{1}{2}}(CL + 4R).
\end{aligned}$$

Since ϵ is arbitrary, the proof is complete. ■

The proof of Theorem 1 follows the approach in Lanthaler et al. (2021), but generalizes the result from $L^2(D';\mathbb{R})$ to the Banach space $\mathcal{U}$. Indeed, the conditions on $\mathcal{U}$ in Theorem 1 are satisfied by all of the spaces $L^p(D';\mathbb{R})$ for $1 \leq p < \infty$ as well as $C(D';\mathbb{R})$. The input space $\mathcal{A}$, in Theorem 1, is asserted to be $C(D;\mathbb{R})$ which can, for some applications, be restrictive. We show in Theorem 2 that this restriction may be lifted and $\mathcal{A}$ can be chosen to be a much more general Banach space. This comes at the price of assuming that the true operator of interest $\mathcal{G}^\dagger$ is Hölder continuous and the addition of an extra linear layer to $\mathcal{G}_{M,\theta}$.

Theorem 2 *Let $D \subset \mathbb{R}^d$, $D' \subset \mathbb{R}^{d'}$ be compact domains and suppose that $\mathcal{A}$ is a separable Banach space of real-valued functions on D such that $C(D;\mathbb{R}) \subset \mathcal{U}$ is dense and that $\mathcal{U}$ is a separable Banach space of real-valued functions on D' such that $C(D';\mathbb{R}) \subset \mathcal{U}$ is dense, and that* *there exists a constant $C > 0$ such that $\|u\|_{\mathcal{U}} \leq C\|u\|_{L^\infty(D';\mathbb{R})}$ for all $u \in \mathcal{U} \cap L^\infty(D';\mathbb{R})$. Let μ be a probability measure supported on $\mathcal{A}$ and assume that, for some $\alpha > 0$, $\mathbb{E}_{a\sim\mu}\|a\|_{\mathcal{A}}^{4\alpha} < \infty$. Suppose further that $\sigma : \mathbb{R} \to \mathbb{R}$ is Hölder continuous and of the Tauber-Wiener class. Then for any μ-measurable $\mathcal{G}^\dagger \in L^2_\mu(C(D;\mathbb{R});\mathcal{U})$ that is α-Hölder continuous and $\epsilon > 0$, there exists a continuous, linear map $F : \mathcal{A} \to C(D;\mathbb{R})$, independent of $\mathcal{G}^\dagger$, as well as $M_0 > 0$ such that, for any $M \geq M_0$, there exists a neural operator $\mathcal{G}_\theta$ of the form* (5) *such that the zero-extended neural operator $\mathcal{G}_{M,\theta}$ satisfies*

$$\|\mathcal{G}^\dagger - \mathcal{G}_{M,\theta} \circ F\|_{L^2_\mu(\mathcal{A};\mathcal{U})} < \epsilon.$$

Proof Since $\mathcal{A}$ is a Polish space, we can find a compact set $K \subset \mathcal{A}$ such that $\mu(\mathcal{A} \setminus K) < \epsilon$. Therefore, we can apply Lemma 7 to find a number $n \in \mathbb{N}$, continuous, linear, functionals $G_1, \dots, G_n \in C(\mathcal{A};\mathbb{R})$ and functions $\varphi_1, \dots, \varphi_n \in C(D;\mathbb{R})$ such that

$$\sup_{a\in K} \Big\| a - \sum_{j=1}^n G_j(a)\varphi_j \Big\|_{\mathcal{A}} = \sup_{a\in K} \|a - F(a)\|_{\mathcal{A}} < \epsilon$$

where the equality defines the map $F : \mathcal{A} \to C(D;\mathbb{R})$. Linearity and continuity of F follow by linearity and continuity of the functionals $G_1, \dots, G_n$. Therefore, there exists a constant $C_1 > 0$, such that

$$\|F(a)\|_{\mathcal{A}} \leq C_1 \|a\|_{\mathcal{A}}, \qquad \forall a \in \mathcal{A}.$$

We can apply Theorem 1 to find $M_0 > 0$, such that, for any $M \geq M_0$, there is a neural operator $\mathcal{G}_{M,\theta}$ such that

$$\|\mathcal{G}^\dagger - \mathcal{G}_{M,\theta}\|_{L^2_{F_\sharp\mu}(C(D;\mathbb{R});\mathcal{U})} < \epsilon.$$

By triangle inequality,

$$\begin{aligned}
\|\mathcal{G}^\dagger - \mathcal{G}_{M,\theta} \circ F\|_{L^2_\mu(\mathcal{A};\mathcal{U})} &\leq \|\mathcal{G}^\dagger - \mathcal{G}^\dagger \circ F\|_{L^2_\mu(\mathcal{A};\mathcal{U})} + \|\mathcal{G}^\dagger \circ F - \mathcal{G}_{M,\theta} \circ F\|_{L^2_\mu(\mathcal{A};\mathcal{U})} \\
&= \|\mathcal{G}^\dagger - \mathcal{G}^\dagger \circ F\|_{L^2_\mu(\mathcal{A};\mathcal{U})} + \|\mathcal{G}^\dagger - \mathcal{G}_{M,\theta}\|_{L^2_{F_\sharp\mu}(C(D;\mathbb{R});\mathcal{U})} \\
&< \|\mathcal{G}^\dagger - \mathcal{G}^\dagger \circ F\|_{L^2_\mu(\mathcal{A};\mathcal{U})} + \epsilon.
\end{aligned}$$

We now estimate the first term using α-Hölder continuity of $\mathcal{G}^\dagger$ and the generalized triangle inequality, see e.g. Dadipour et al. (2012). In particular, there are constants $C_2, C_3 > 0$ such that

$$\begin{aligned}
\|\mathcal{G}^\dagger - \mathcal{G}^\dagger \circ F\|^2_{L^2_\mu(\mathcal{A};\mathcal{U})} &\leq C_2 \int_{\mathcal{A}} \|a - F(a)\|_{\mathcal{A}}^{2\alpha} \, \mathrm{d}\mu(a) \\
&\leq C_2 \Bigg(\int_K \|a - F(a)\|_{\mathcal{A}}^{2\alpha} \, \mathrm{d}\mu(a) \\
&\quad + C_3 \Bigg(\int_{\mathcal{A}\setminus K} \|a\|_{\mathcal{A}}^{2\alpha} \, \mathrm{d}\mu(a) + \int_{\mathcal{A}\setminus K} \|F(a)\|_{\mathcal{A}}^{2\alpha} \, \mathrm{d}\mu(a) \Bigg) \Bigg) \\
&\leq C_2 \Bigg(\epsilon^{2\alpha} + C_3 \Bigg(\epsilon^{1/2} (\mathbb{E}_{a\sim\mu}\|a\|_{\mathcal{A}}^{4\alpha})^{1/2} + \epsilon^{1/2} C_1^{2\alpha} (\mathbb{E}_{a\sim\mu}\|a\|_{\mathcal{A}}^{4\alpha})^{1/2} \Bigg) \Bigg)
\end{aligned}$$

using Cauchy–Schwarz in the last line. Since $\mathbb{E}_{a\sim\mu}\|a\|_{\mathcal{A}}^{4\alpha} < \infty$ by assumption and ϵ is arbitrary, the proof is complete. ■

In general, the linear map F from Theorem 2 may not be approximated by an integral kernel operator. However, in the special case when $\mathcal{A} = L^p(D;\mathbb{R})$ for any $1 < p < \infty$, F can indeed be approximated by an integral kernel operator. We state and prove this result in Corollary 3.

Corollary 3 *Assume the setting of Theorem 2 and suppose that $\mathcal{A} = L^p(D;\mathbb{R})$ for any $1 < p < \infty$. Then there exists a neural network $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that the conclusion of Theorem 2 holds with $F : L^p(D;\mathbb{R}) \to C(D;\mathbb{R})$ defined as*

$$F(a)(x) = \int_D \kappa(x,y)a(y)\,dy, \qquad \forall a \in L^p(D;\mathbb{R}),\ \forall x \in D.$$

Proof Let $\tilde{F} : \mathcal{A} \to C(D;\mathbb{R})$ be the mapping defined in Theorem 2 which is given as

$$\tilde{F}(a) = \sum_{j=1}^n G_j(a)\varphi_j, \qquad \forall a \in \mathcal{A}$$

for some continuous, linear, functionals $G_1,\dots,G_n \in C(\mathcal{A};\mathbb{R})$ and functions $\varphi_1,\dots,\varphi_n \in C(D;\mathbb{R})$. Perusal of the proof of Theorem 2 implies that it is enough to to show existence of a neural network $\kappa \in \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ for which the mapping $F : \mathcal{A} \to C(D;\mathbb{R})$ defined as

$$F(a)(x) = \int_D \kappa(x,y)a(y)\,\mathrm{d}y, \qquad \forall a \in \mathcal{A},\ \forall x \in D$$

satisfies, for all $\epsilon > 0$,

$$\int_{\mathcal{A}} \|\tilde{F}(a) - F(a)\|_{\mathcal{A}}^{2\alpha}\,\mathrm{d}\mu(a) < \epsilon.$$

Let $1 < q < \infty$ be the Hölder conjugate of p. Since $\mathcal{A} = L^p(D;\mathbb{R})$, by the Reisz Representation Theorem (Conway, 1985, Appendix B), there exists functions $g_1,\dots,g_n \in L^q(D;\mathbb{R})$ such that, for each $j = 1,\dots,n$,

$$G_j(a) = \int_D a(x)g_j(x)\,\mathrm{d}x, \qquad \forall a \in L^p(D;\mathbb{R}).$$

By density of $C(D;\mathbb{R})$ in $L^q(D;\mathbb{R})$, we can find functions $\psi_1,\dots,\psi_n \in C(D;\mathbb{R})$ such that

$$\sup_{j\in\{1,\dots,n\}} \|\psi_j - g_j\|_{L^q(D;\mathbb{R})} < \frac{\epsilon}{n^{1/2\alpha}}.$$

Define $\hat{F} : L^p(D;\mathbb{R}) \to C(D;\mathbb{R})$ by

$$\hat{F}(a) = \sum_{j=1}^n \int_D \psi_j(y)a(y)\,\mathrm{d}y\,\varphi_j(x) \qquad \forall a \in L^p(D;\mathbb{R}),\ \forall x \in D.$$

Furthermore, we can find a neural network $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that

$$\sup_{x,y\in D} |\kappa(x,y) - \sum_{j=1}^n \psi_j(y)\phi_j(x)| < \epsilon$$

and define F using this network. Then there is a constant $C_1 > 0$ such that

$$\|\tilde{F}(a) - F(a)\|^{2\alpha}_{L^p(D;\mathbb{R})} \leq C_1(\|\tilde{F}(a) - \hat{F}(a)\|^{2\alpha}_{L^p(D;\mathbb{R})} + \|\hat{F}(a) - F(a)\|^{2\alpha}_{L^p(D;\mathbb{R})}).$$

For the first term, we have that there is a constant $C_2 > 0$ such that

$$\begin{aligned}
\|\tilde{F}(a) - \hat{F}(a)\|^{2\alpha}_{L^p(D;\mathbb{R})} &\leq C_2 \sum_{j=1}^{n} \| \int_D a(y)(g_j(y) - \psi_j(y)) \, \mathrm{d}y \, \varphi_j \|^{2\alpha}_{L^p(D;\mathbb{R})} \\
&\leq C_2 \sum_{j=1}^{n} \|a\|^{2\alpha}_{L^p(D;\mathbb{R})} \|g_j - \psi_j\|^{2\alpha}_{L^q(D;\mathbb{R})} \|\varphi_j\|^{2\alpha}_{L^p(D;\mathbb{R})} \\
&\leq C_3 \epsilon^{2\alpha} \|a\|^{2\alpha}_{L^p(D;\mathbb{R})}
\end{aligned}$$

for some $C_3 > 0$. For the second term,

$$\begin{aligned}
\|\hat{F}(a) - F(a)\|^{2\alpha}_{L^p(D;\mathbb{R})} &= \| \int_D a(y) \Big(\sum_{j=1}^{n} \psi_j(y) \varphi_j(\cdot) - \kappa(\cdot, y) \Big) \, \mathrm{d}y \|^{2\alpha}_{L^p(D;\mathbb{R})} \\
&\leq |D|^{2\alpha} \epsilon^{2\alpha} \|a\|^{2\alpha}_{L^p(D;\mathbb{R})}.
\end{aligned}$$

Therefore, we conclude that there is a constant $C > 0$ such that

$$\int_{\mathcal{A}} \|\tilde{F}(a) - F(a)\|^{2\alpha}_{\mathcal{A}} \, \mathrm{d}\mu(a) \leq \epsilon^{2\alpha} C \mathbb{E}_{a \sim \mu} \|a\|^{2\alpha}_{L^p(D;\mathbb{R})}.$$

Since $\mathbb{E}_{a \sim \mu} \|a\|^{2\alpha}_{L^p(D;\mathbb{R})} < \infty$ by assumption and ϵ is arbitrary, the proof is complete. ■

Furthermore, we note that, in Lanthaler et al. (2021), it is shown that, in certain cases, DeepONets can learn the optimal *linear* approximation. In particular, assuming $\mathcal{U}$ is a Hilbert space, for any fixed $n \in \mathbb{N}$, there is a DeepONet with range $O_n \subset \mathcal{U}$ where O_n is closed linear subspace of $\mathcal{U}$ with dimension n that is approximately a minimizer of

$$\min_{U \in \mathcal{U}_n} \|I - P_U\|_{L^p_{\mathcal{G}^\dagger_\sharp \mu}(\mathcal{U};\mathcal{U})} \tag{17}$$

where $\mathcal{U}_n$ denotes the set of all n-dimensional linear subspaces of $\mathcal{U}$ and $P_U : \mathcal{U} \to U$ is the orthonormal projection of $\mathcal{U}$ onto U. This follows by noting that a neural network can be used to approximate the first n eigenfunctions of the convariance of $\mathcal{G}^\dagger_\sharp \mu$ whose span is indeed a minimizer of (17). In the closely related work Bhattacharya et al. (2020), the eigenfunctions of the convariance of $\mathcal{G}^\dagger_\sharp \mu$ are instead directly approximated via PCA (Blanchard et al., 2007). We note that such optimality results also hold for neural operators by the reduction (11) and therefore we do not state and prove them here but simply refer the interested reader to Lanthaler et al. (2021). We emphasize however that the general form of the neural operator (5) constitutes a form of *non-linear* approximation and we therefore expect faster rates of convergence than the optimal linear rates. We leave this characterization for future work.

4.1 Continuous Operators

The proof of Theorem 1 relies on Theorem 4 which states that for any continuous $\mathcal{G}^\dagger$, any compact set $K \subset \mathcal{A}$, and a fixed error tolerance, there is a neural operator which uniformly approximates $\mathcal{G}^\dagger$ up the error tolerance. The proof is based on the following single-layer construction

$$(\mathcal{G}_\theta(a))(x) = \mathcal{Q}\left(\int_D \kappa^{(1)}(x,y)\sigma\left(\int_D \kappa^{(0)}(y,z)\mathcal{P}(a)(z)\,\mathrm{d}z + b(y)\right)\,\mathrm{d}y\right) \tag{18}$$

which we obtain from (10) by setting $W_0 = 0$. Our result is based on the observation that (18) can be reduced to (11) which simplifies the problem to that of approximating a finite set of non-linear functionals by parmetrizations of the form (12). In Chen and Chen (1995), it is shown that functionals defined on compact sets of $\mathcal{A}$ can be uniformly approximated by single-layer, finite width, neural networks. We show in Theorem 11 of Appendix B that such neural networks may be approximated by the parametric forms (12). Since the result of Chen and Chen (1995) is at the heart of our approximation result, in Appendix B, we give an intuitive overview of their proof techniques for the interested reader.

Theorem 4 *Let $D \subset \mathbb{R}^d$, $D' \subset \mathbb{R}^{d'}$ be compact domains and $K \subset C(D;\mathbb{R})$ be a compact set. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be Hölder continuous and of the Tauber-Wiener class. Then for any $\mathcal{G}^\dagger \in C(K; C(D';\mathbb{R}))$ and $\epsilon > 0$, there exists $n = n(\epsilon) \in \mathbb{N}$ as well as neural networks $\mathcal{Q} : \mathbb{R}^n \to \mathbb{R}$, $\mathcal{P} : \mathbb{R} \to \mathbb{R}^n$, $\kappa^{(1)} : \mathbb{R}^{d'} \times \mathbb{R}^d \to \mathbb{R}^{n\times n}$, $\kappa^{(0)} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{n\times n}$, $b : \mathbb{R}^d \to \mathbb{R}^n$ with parameters concatenated in $\theta \in \mathbb{R}^p$ such that the neural operator $\mathcal{G}_\theta$ defined by* (18) *satisfies*

$$\sup_{a\in K} \|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|_{C(D';\mathbb{R})} < \epsilon.$$

Furthermore, if $B \subset C(D;\mathbb{R})$ is a bounded set such that $K \subseteq B$, then the neural networks can be chosen so that there exists a constant $C > 0$ such that

$$\sup_{a\in B} \|\mathcal{G}_\theta(a)\|_{C(D;\mathbb{R})} < C.$$

Theorem 4 is a consequence of the reduction (11) and Theorem 5 which establishes existence of the scalar kernels necessary for defining $\kappa^{(0)}$ and $\kappa^{(1)}$ as well as the bias function b. Indeed, the proof of Theorem 4 simply amounts to approximating the continuous function $\mathcal{P}$, $\kappa^{(1)}$, $\kappa^{(0)}$, b by neural networks and is given in Appendix B.

Theorem 5 *Let $D \subset \mathbb{R}^d$, $D' \subset \mathbb{R}^{d'}$ be compact domains and $K \subset C(D;\mathbb{R})$ be a compact set. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be Hölder continuous and of the Tauber-Wiener class. Then for any $\mathcal{G}^\dagger \in C(K; C(D';\mathbb{R}))$ and $\epsilon > 0$, there exists a number $n = n(\epsilon) \in \mathbb{N}$ as well as separable kernels $\kappa_1^{(1)}, \dots, \kappa_n^{(1)} \in C(D' \times D;\mathbb{R})$, smooth kernels $\kappa_1^{(0)}, \dots, \kappa_n^{(0)} \in C^\infty(D \times D;\mathbb{R})$, and smooth functions $b_1, \dots, b_n \in C^\infty(D;\mathbb{R})$ such that*

$$\sup_{a\in K}\sup_{x\in D'} \left|\mathcal{G}^\dagger(a)(x) - \sum_{j=1}^n \int_D \kappa_j^{(1)}(x,y)\sigma\left(\int_D \kappa_j^{(0)}(y,z)a(z)\,\mathrm{d}z + b_j(y)\right)\,\mathrm{d}y\right| < \epsilon.$$

Proof Since K is compact and $\mathcal{G}^\dagger$ is continuous, $\mathcal{G}^\dagger(K)$ is compact. Therefore, by Lemma 6, there exists $n \in \mathbb{N}$ as well as functionals $G_1, \dots, G_n \in C(K;\mathbb{R})$ and functions $\varphi_1, \dots, \varphi_n \in \mathcal{G}^\dagger(K)$ such that for any $a \in K$

$$\left| \mathcal{G}^\dagger(a)(x) - \sum_{j=1}^{n} G_j(a)\varphi_j(x) \right| < \frac{\epsilon}{2} \qquad \forall x \in D'. \tag{19}$$

Continuity of the functionals follows since compositions of continuous maps are continuous. Since K is compact, we can find $M > 0$ such that

$$\sup_{a \in K} \sup_{x \in D} |a(x)| \leq M.$$

Repeatedly applying Theorem 11 for $j = 1, \dots, n$, we can find smooth kernels $\kappa_j^{(0)} \in C^\infty(D \times D;\mathbb{R})$ and smooth functions $w_j, b_j \in C^\infty(D;\mathbb{R})$ such that

$$\left| G_j(a) - \int_D w_j(y)\sigma\left(\int_D \kappa_j^{(0)}(y,z)a(z)\,\mathrm{d}z + b_j(y)\right)\mathrm{d}y \right| < \frac{\epsilon}{2nM} \qquad \forall a \in K. \tag{20}$$

Define the kernels

$$\kappa_j^{(1)}(x,y) = w_j(y)\varphi_j(x) \qquad \forall y \in D,\, \forall x \in D', \quad j = 1, \dots, n$$

as well as our approximation

$$\mathcal{G}(a)(x) = \sum_{j=1}^{n} \int_D \kappa_j^{(1)}(x,y)\sigma\left(\int_D \kappa_j^{(2)}(y,z)a(z)\,\mathrm{d}z + b_j(y)\right)\mathrm{d}y \qquad \forall x \in D', \quad \forall a \in K.$$

Applying triangle inequality and combining (19) and (20), we find that, for any $a \in K$ and $x \in D'$, we have

$$\begin{aligned}
\left| \mathcal{G}^\dagger(a)(x) - \mathcal{G}(a)(x) \right| &\leq \left| \mathcal{G}^\dagger(a)(x) - \sum_{j=1}^{n} G_j(a)\varphi_j(x) \right| + \left| \sum_{j=1}^{n} G_j(a)\varphi_j(x) - \mathcal{G}(a)(x) \right| \\
&< \frac{\epsilon}{2} + \sum_{j=1}^{n} |\varphi_j(x)| \left| G_j(a) - \int_D w_j(y)\sigma\left(\int_D \kappa_j^{(2)}(y,z)a(z)\,\mathrm{d}z + b_j(y)\right)\mathrm{d}y \right| \\
&< \frac{\epsilon}{2} + M \sum_{j=1}^{n} \frac{\epsilon}{2nM} \\
&= \epsilon
\end{aligned}$$

as desired. ■

5. Parameterization and Computation

In this section, we discuss various ways of parameterizing the infinite dimensional architecture (5) such that the number of parameters used is independent of the data discretization. Furthermore we discuss the computational complexity of the proposed parameterizations and suggest algorithms which yield efficient numerical methods for approximation. Subsections 5.1-5.4 delineate each of the proposed methods.

To simplify notation, we will only consider a single layer of (5) i.e. (9) 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

$$u(x) = \sigma\left(Wv(x) + \int_D \kappa(x,y)v(y)\,\mathrm{d}\nu(y) + b(x)\right) \qquad \forall x \in D \tag{21}$$

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. We will consider σ to be fixed, and, for the time being, take $\mathrm{d}\nu(y) = \mathrm{d}y$ to be the Lebesgue measure on $\mathbb{R}^d$. Equation (21) 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. Perusal of the proof of Theorem 11, shows that we do not loose any approximation power by letting $b : D \to \mathbb{R}^m$ be a constant function. 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 (6) but note that similar ideas may also be applied to (7) and (8). Furthermore, we will drop σ, W, and b from (21) and simply consider the linear update

$$u(x) = \int_D \kappa(x,y)v(y)\,\mathrm{d}\nu(y) \qquad \forall x \in D. \tag{22}$$

To demonstrate the computational challenges associated with (22), let $\{x_1, \dots, x_J\} \subset D$ be a uniformly-sampled J-point discretization of D. Recall that we assumed $\mathrm{d}\nu(y) = \mathrm{d}y$ and, for simplicity, suppose that $\nu(D) = 1$, then the Monte Carlo approximation of (22) is

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

Therefore to compute u on the entire grid requires $\mathcal{O}(J^2)$ matrix-vector multiplications each of which requires $\mathcal{O}(mn)$ operations (for the rest of the discussion, we will treat $mn = \mathcal{O}(1)$ as constant and consider only the cost with respect to J the discretization parameter). 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 (22) becomes intractable and new ideas are needed in order to alleviate this. Subsections 5.1-5.4 propose different approaches inspired by classical methods in numerical analysis for the solution to this problem. 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 $\{x_1, \dots, x_J\} \subset D$. 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.

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

where we use (j, l) to index an individual block rather than a matrix element. Various numerical algorithms for the efficient computation of (22) can be derived based on assumptions made about the structure of this matrix, for example, bounds on its rank or sparsity.

5.1 Graph Neural Operator (GNO)

We first outline the Graph Neural Operator (GNO) which approximates (22) 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 (22) 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 $x_{k_1}, \dots, x_{k_{J'}} \subset \{x_1, \dots, x_J\}$ be $J' \ll J$ randomly selected points and, assuming $\nu(D) = 1$, approximate (22) by

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

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

$$K \approx K_{JJ'} K_{J'J'} K_{J'J} \tag{23}$$

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 (22) 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 (22) becomes

$$u(x) = \int_{s(x)} \kappa(x, y) v(y) \, \mathrm{d}y \qquad \forall x \in D. \tag{24}$$

If the size of each set $s(x)$ is smaller than D then the cost of computing (24) 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 [CITE]. 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 $r = 1$, then (24) reverts to (22). 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 (24) 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 (22) 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 (22) 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 5.3.

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 (22) on arbitrary discretizations of the domain D. To do so, we treat a discretization $\{x_1, \dots, x_J\} \subset D$ 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 $j = 1, \dots, J$, we assign the node x_j the value $v(x_j)$ and emanate from it edges to the nodes $s(x_j) \cap \{x_1, \dots, x_J\} = \mathcal{N}(x_j)$ 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 (22), 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 (7) or (8).

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

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

which corresponds to the Monte-Carlo approximation of the integral (24). 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 5.3.

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 gained by enforcing that the convolution kernel have local support an idea similar to our truncation approximation. Furthermore by directly using a translation invariant kernel, a CNN architecture becomes translation equivariant which 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 transnational symmetries that, unlike CNNs, remains 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 (22) becomes the convolution

$$u(x) = (\kappa * v)(x) = \int_{B(x,r) \cap D} \kappa(x - y) v(y) \, \mathrm{d}y \qquad \forall x \in D. \tag{25}$$

Notice that (25) is precisely (24) 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, (25) becomes a layer of a convolution neural network that is consistent in function space. Indeed the parameters of (25) 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* is functions space and applying a Monte Carlo approximation to the integral, (25) becomes a standard CNN. This is most easily demonstrated when $D = [0, 1]$ and the discretization $\{x_1, \dots, x_J\}$ is equispaced i.e. $|x_{j+1} - x_j| = h$ for any $j = 1, \dots, J - 1$. Let $k \in \mathbb{N}$ be an odd filter size and let $z_1, \dots, z_k \in \mathbb{R}$ be the points $z_j = (j - 1 - (k - 1)/2)h$ for $j = 1, \dots, k$. It is easy to see that $\{z_1, \dots, z_k\} \subset \bar{B}(0, (k - 1)h/2)$ 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 $z_1, \dots, z_k$ thus yielding kmn parameters. Then (25) becomes

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

where we define $v(x) = 0$ if $x \notin \{x_1, \dots, x_J\}$. Up to the constant factor $1/k$ which can be re-absobred 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 only simplicity of the 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.

5.2 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 that is similar to the DeepONet construction of Lu et al. (2019) discussed in Section 3.2, but parameterized to be consistent in function space. 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 (22) 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

$$u = \sum_{j=1}^{r} (\varphi^{(j)} \otimes \psi^{(j)})v \tag{26}$$

which corresponds preceisely to applying the SVD of a rank r operator to the function v. Equation (26) 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, (26) defines an operator mapping $L^2(D;\mathbb{R}^m) \to L^2(D;\mathbb{R}^n)$ 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. While this method enjoys a linear computational complexity, similar to the DeepONets of Lu et al. (2019), it also constitutes a *linear* approximation method which may not be able to effectively capture the solution manifold; see Section 3.2 for further discussion.

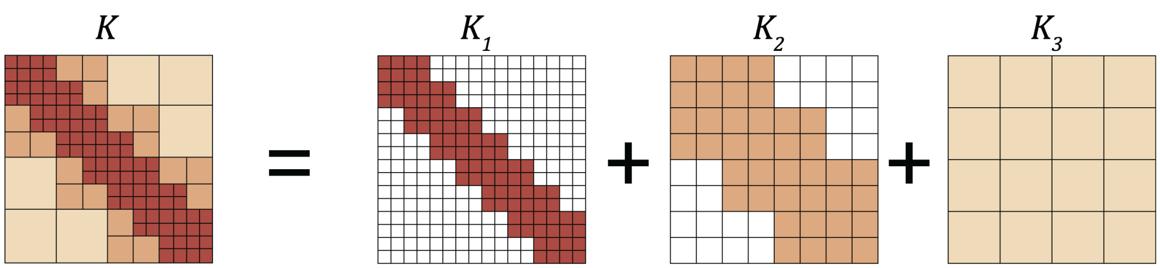
5.3 Multipole Graph Neural Operator (MGNO)

A natural extension to directly working with kernels in a tensor product form as in Section 5.2 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 (22); for a detailed discussion see e.g. (E, 2011, Section 3.2). FFM can be viewed as a systematic approach of combining 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 constraint 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 2:

$$K = K_1 + K_2 + \ldots + K_L \tag{27}$$

where K_1 corresponds to the shortest-range interaction, and K_L corresponds to the longest-range interaction. While the uniform grids depicted in Figure 2 produce an orthogonal decomposition of the kernel, the decomposition may be generalized to arbitrary discretizations by allowing slight overlap of the ranges.



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.

Figure 2: Hierarchical matrix decomposition

Multi-scale discretization. We construct $L \in \mathbb{N}$ levels of discretization, where the finest grid corresponds to the shortest-range interaction K_1, and the coarsest discretization corresponds to the longest-range interaction K_L. In general, the underlying discretization can be arbitrary and we produce a hierarchy of L discretization 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 (2). 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 forms an orthogonal decomposition to 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: 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 (23). 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 (2), K_1 is full-rank but very sparse while K_L is dense but low-rank. Such structure can be achieved by applying equation (23) recursively to equation (27), leading to the multi-resolution matrix factorization (Kondor et al., 2014):

$$K \approx K_{1,1} + K_{1,2}K_{2,2}K_{2,1} + K_{1,2}K_{2,3}K_{3,3}K_{3,2}K_{2,1} + \cdots \tag{28}$$

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 5.1, we define the transition matrices as discretizations of the following integral kernel operators:

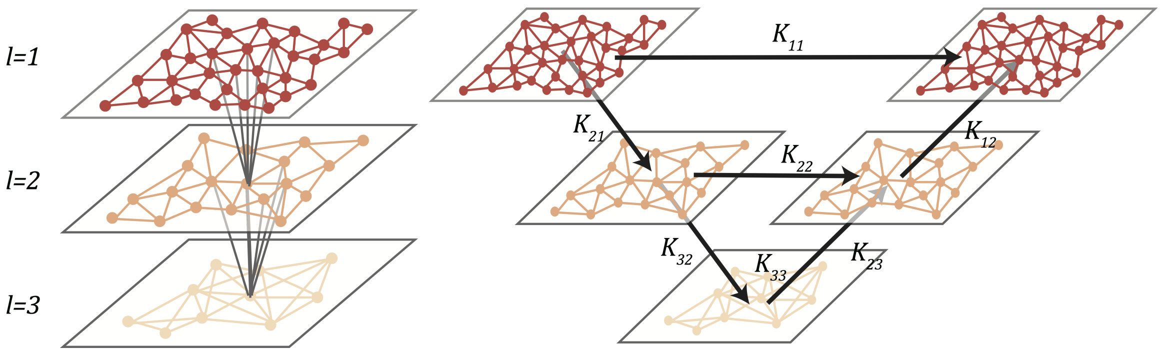
$$K_{l,l} : v_l \mapsto u_l = \int_{B(x,r_{l,l})} \kappa_{l,l}(x, y)v_l(y)\, \mathrm{d}y \tag{29}$$

$$K_{l+1,l} : v_l \mapsto u_{l+1} = \int_{B(x,r_{l+1,l})} \kappa_{l+1,l}(x, y)v_l(y)\, \mathrm{d}y \tag{30}$$

$$K_{l,l+1} : v_{l+1} \mapsto u_l = \int_{B(x,r_{l,l+1})} \kappa_{l,l+1}(x, y)v_{l+1}(y)\, \mathrm{d}y \tag{31}$$

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 3, for efficiently computing (28). 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



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

Figure 3: V-cycle

$\check{v}_{l+1} = K_{l+1,l}\check{v}_l$. 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 the 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 (28).

Employing (29)-(31), 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 (5), we introduce the linear operator $W \in \mathbb{R}^{n \times n}$ (denoting it by W_l for each corresponding resolution) to help regularizing the iteration, as well as the nonlinear activation function σ to increase . 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

$$\text{For } l = 1, \ldots, L: \qquad \check{v}_{l+1}^{(t+1)} = \sigma(\hat{v}_{l+1}^{(t)} + K_{l+1,l}\check{v}_l^{(t+1)}) \tag{32}$$

- Upward Pass

$$\text{For } l = L, \ldots, 1: \qquad \hat{v}_l^{(t+1)} = \sigma((W_l + K_{l,l})\check{v}_l^{(t+1)} + K_{l,l-1}\hat{v}_{l-1}^{(t+1)}). \tag{33}$$

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 (27) 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 previous graph and so on. 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 2 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 problem where J is very large or combined with active learning method to adaptively choose the regions where a finer discretization is needed.

Linear complexity. Each term in the decomposition (27) is represented by the kernel matrix $K_{l,l}$ for $l = 1, \dots, L$, and $K_{l+1,l}, K_{l,l+1}$ for $l = 1, \dots, L-1$ 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 $\sum_{l=1}^{L} \mathcal{O}(J_l^2 r_l^d) = \mathcal{O}(J)$. Combined with a Nyström approximation, we can obtain $\mathcal{O}(J')$ complexity for some $J' \ll J$.

5.4 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 (22) with almost linear complexity. The method we outline was first described in Li et al. (2020a) and is termed the Fourier Neural Operator (FNO). For simplicity, we will assume that $D = \mathbb{T}^d$ is the unit torus and all functions are complex-valued. Let $\mathcal{F} : L^1(D; \mathbb{C}^n) \to L^1(D; \mathbb{C}^n)$ denote the Fourier transform of a function $v : D \to \mathbb{C}^n$ and $\mathcal{F}^{-1}$ its inverse

$$(\mathcal{F}v)_j(k) = \int_D v_j(x) e^{-2i\pi\langle x,k\rangle} \mathrm{d}x$$
$$(\mathcal{F}^{-1}v)_j(x) = \int_D v_j(k) e^{2i\pi\langle x,k\rangle} \mathrm{d}k$$

for $j = 1, \dots, n$ where $i = \sqrt{-1}$ is the imaginary unit and $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product on $\mathbb{R}^d$. By letting $\kappa(x, y) = \kappa(x-y)$ for some $\kappa : D \to \mathbb{C}^{m \times n}$ in (22) and applying the convolution theorem, we find that

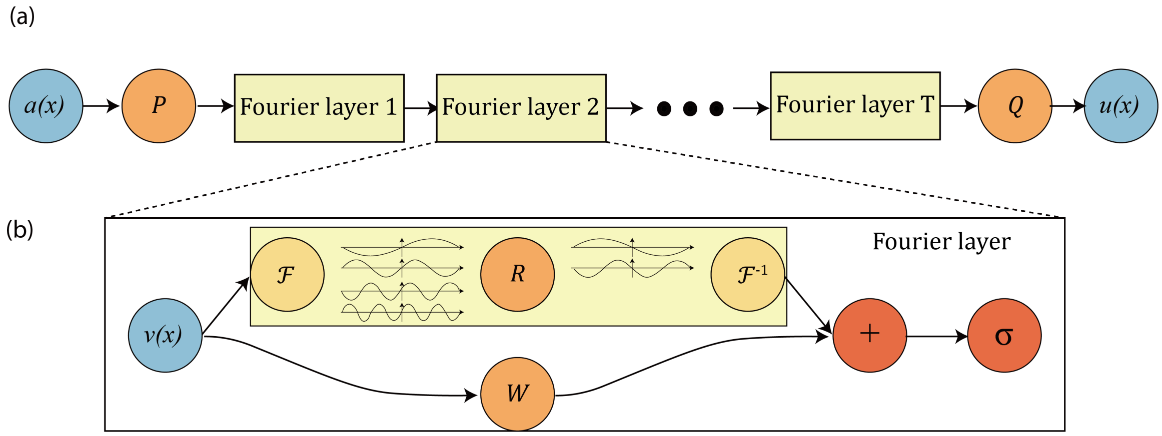
$$u(x) = \mathcal{F}^{-1}\big(\mathcal{F}(\kappa) \cdot \mathcal{F}(v)\big)(x) \qquad \forall x \in D.$$

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

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

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

For frequency mode $k \in D$, we have $(\mathcal{F}v)(k) \in \mathbb{C}^n$ and $R_\phi(k) \in \mathbb{C}^{m \times n}$. Notice that since we assume κ is periodic, it admits a Fourier series expansion, so we may work with the discrete modes $k \in \mathbb{Z}^d$. We pick a finite-dimensional parameterization by truncating the Fourier series at a maximal number of modes $k_{\max} = |Z_{k_{\max}}| = |\{k \in \mathbb{Z}^d : |k_j| \leq k_{\max,j},\ \text{for } j = 1, \dots, d\}|$. 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 that v is



(a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four 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 and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

Figure 4: **top:** The architecture of the neural operators; **bottom:** Fourier layer.

real-valued and we want u to also be a real-valued, we will impose that κ is real-valued by enforcing conjugate symmetry in the parameterization i.e.

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

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 4 gives a pictorial representation of an entire Neural Operator architecture employing Fourier layer.

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

$$\big(R \cdot (\mathcal{F} v_t)\big)_{k,l} = \sum_{j=1}^{n} R_{k,l,j} (\mathcal{F} v)_{k,j}, \qquad k = 1, \dots, k_{\max}, \quad l = 1, \dots, m. \tag{34}$$

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 $v \in \mathbb{C}^{J \times n}$, $k = (k_1, \dots, k_d) \in \mathbb{Z}_{s_1} \times \cdots \times \mathbb{Z}_{s_d}$, and $x = (x_1, \dots, x_d) \in D$, the FFT $\hat{\mathcal{F}}$ and its inverse $\hat{\mathcal{F}}^{-1}$ are defined as

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

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

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

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

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 (34) via matrix-vector multiplication. In practice, we have found that choosing $k_{\max,j} = 12$ 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 (7). 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:

- *Constant.* 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 previously described direct parameterization, 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 neural network parameterization has a worse performance. This is likely due to the discrete structure of the space $\mathbb{Z}^d$. 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 $e^{2\pi i \langle x,k \rangle}$ which 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 $\mathcal{O}(Jk_{\max})$, 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.

5.5 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

$$u(x) = \int_{B(x,r)} \kappa(x,y)v(y) \, \mathrm{d}y \tag{35}$$

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

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

$$u(x) = \sum_{j=1}^{r} \langle \psi^{(j)}, v \rangle \varphi^{(j)}(x) \tag{36}$$

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

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

$$\begin{aligned} K &= K_{1,1} + K_{1,2}K_{2,2}K_{2,1} + K_{1,2}K_{2,3}K_{3,3}K_{3,2}K_{2,1} + \cdots \\ u(x) &= (Kv)(x) \end{aligned} \tag{37}$$

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

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

$$u(x) = \mathcal{F}^{-1}\big(R_\phi \cdot \mathcal{F}(v)\big)(x) \tag{38}$$

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

6. Test Problems

One of the major application of neural operator is to learn the solution operator for partial differential equations. In this section, we give define four test problems–Poisson equation, Darcy flow, Burger's equation, and Navier-Stokes equation. In general, the equation has the following form:

$$\begin{aligned} (\mathsf{L}_a u)(x) &= f(x), \qquad && x \in D \\ u(x) &= 0, && x \in \partial D, \end{aligned} \tag{39}$$

with solution function $u : D \to \mathbb{R}$, and function parameter $a : D \to \mathbb{R}$ entering the definition of L_a. f is usually considered as fixed.

For most of the cases, the goal is to learn the map

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

The domain D is discretized into n points. Additionally, we want to study approximation of the Green's function in the Possion equation problem and the inverse problem of the Navier-Stokes equation.

6.1 Poisson equation and its Green's function representation

First, as a toy example, let's consider 1d Possion equation. The 1d poisson equation has a well-defined Green function formation. We want to visualize the learn kernel function κ and compared it with the true underlying Green function kernel. Now consider the setting where $D = [0,1]$, $\mathsf{L}_a = \partial_{xx}$, so that (39) reduces to the 1-dimensional Poisson equation.

$$\begin{aligned} \partial_{xx} u(x) = f(x), \qquad & x \in (0,1) \\ u(x) = 0, \qquad & x = 0, 1 \end{aligned}$$

where $f : [0,1] \to \mathbb{R}$ is the source function; $u : [0,1] \to \mathbb{R}$ is the solution function. The goal is to learn the map

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

The data is generated on a 85×85 grid. We set the number of training data to be 1000; the number of testing data to be 1000 as well.

Green's function. The above equation has an explicit Green's function:

$$G(x,y) = \frac{1}{2}\left(x + y - |y - x|\right) - xy.$$

such that

$$u(x) = \int_0^1 G(x,y) f(y) \mathrm{d}y$$

Note that although the map $\mathcal{G}^\dagger : f \mapsto u$ is, in function space, linear, the Green's function itself is not linear in either argument.

For this problem, we consider a simplification of the iterative update framework defined in section 3. We omit step 1.,2. and 4. and only use one iteration of step 3. In other words, we set $v_0(x) = f(x)$, $T = 1$, $n = 1$, $\sigma(x) = x$, $W = w = 0$, and $\nu_x(dy) = dy$ (the Lebesgue measure). The purposed operator $\mathcal{G}$ can be directly written as

$$(\mathcal{G}f)(x) = \int_0^1 \kappa(x, y; \phi) f(y) \mathrm{d}y$$

For this test problem, we want to show whether the neural network κ will match the Green function G.

6.2 Darcy Flow

The 2d Darcy Flow is a linear elliptic equation with the form

$$\begin{aligned} -\nabla \cdot (a(x) \nabla u(x)) = f(x) \qquad & x \in (0,1)^2 \\ u(x) = 0 \qquad & x \in \partial(0,1)^2. \end{aligned} \tag{40}$$

The coefficients a are generated according to $a \sim \mu$ where $\mu = \psi_{\#}\mathcal{N}(0, (-\Delta + 9I)^{-2})$ with a Neumann boundry condition on the operator $-\Delta + 9I$ and $f(x)$ is fixed. The mapping $\psi : \mathbb{R} \to \mathbb{R}$ takes the value 12 on the positive part of the real line and 3 on the negative. Such constructions are prototypical models for many physical systems such as permeability in sub-surface flows and material microstructures in elasticity. Solutions u are obtained by using a second-order finite difference scheme on 241×241 and 421×421 grids. Different resolutions are downsampled from this dataset. Without specific notice, the number of training data is set to be 1000; the number of testing data to be 100. For this problem we want to learn the operator for the coefficient function a to the solution function u:

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

6.3 Burgers' Equation

The 1d Burgers' equation is a non-linear equation. It is a common test problem for computational fluid dynamic.

$$\begin{aligned} \partial_t u(x,t) + \partial_x(u^2(x,t)/2) &= \nu \partial_{xx} u(x,t), \qquad && x \in (0, 2\pi), t \in (0, 1] \\ u(x,0) &= u_0(x), && x \in (0, 2\pi) \end{aligned} \tag{41}$$

with periodic boundary conditions. We consider mapping the initial condition to the solution at time one $u_0 \mapsto u(\cdot, 1)$. The initial condition is generated according to $u_0 \sim \mu$ where $\mu = \mathcal{N}(0, 625(-\Delta + 25I)^{-2})$ with periodic boundary conditions. We set the viscosity to $\nu = 0.1$ and solve the equation using a split step method where the heat equation part is solved exactly in Fourier space then the non-linear part is advanced, again in Fourier space, using a very fine forward Euler method. We solve on a spatial mesh with resolution $2^{13} = 8192$ and use this dataset to subsample other resolutions. Without specific notice, the number of training data is set to be 1000; the number of testing data to be 100. Again, we want to learn the operator for the coefficient function a to the solution function u:

$$\mathcal{G}^\dagger : u|_{t=0} \mapsto u|_{t=1}$$

6.4 Navier-Stokes Equation

We consider the 2-d Navier-Stokes equation for a viscous, incompressible fluid in vorticity form on the unit torus:

$$\begin{aligned} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) &= \nu \Delta w(x,t) + f(x), \qquad && x \in (0,1)^2, t \in (0, T] \\ \nabla \cdot u(x,t) &= 0, && x \in (0,1)^2, t \in [0, T] \\ w(x,0) &= w_0(x), && x \in (0,1)^2 \end{aligned} \tag{42}$$

where $u \in C([0,T]; H^r_{\text{per}}((0,1)^2; \mathbb{R}^2))$ for any $r > 0$ is the velocity field, $w = \nabla \times u$ is the vorticity, $w_0 \in L^2_{\text{per}}((0,1)^2; \mathbb{R})$ is the initial vorticity, $\nu \in \mathbb{R}_+$ is the viscosity coefficient, and $f \in L^2_{\text{per}}((0,1)^2; \mathbb{R})$ is the forcing function. We are interested in learning the operator mapping the vorticity up to time 10 to the vorticity up to some later time $T > 10$, $\mathcal{G}^\dagger : C([0,10]; H^r_{\text{per}}((0,1)^2; \mathbb{R})) \to C((10,T]; H^r_{\text{per}}((0,1)^2; \mathbb{R}))$ defined by

$$\mathcal{G}^\dagger : w|_{t \in [0,10]} \mapsto w|_{t \in (10,T]}$$

We choose to formulate the problem in terms of vorticity. Given the vorticity it is easy to derive the velocity. By formulating the problem on vorticity, the neural network models mimic the pseudo-spectral method. The initial condition $w_0(x)$ is generated according to $w_0 \sim \mu$ where

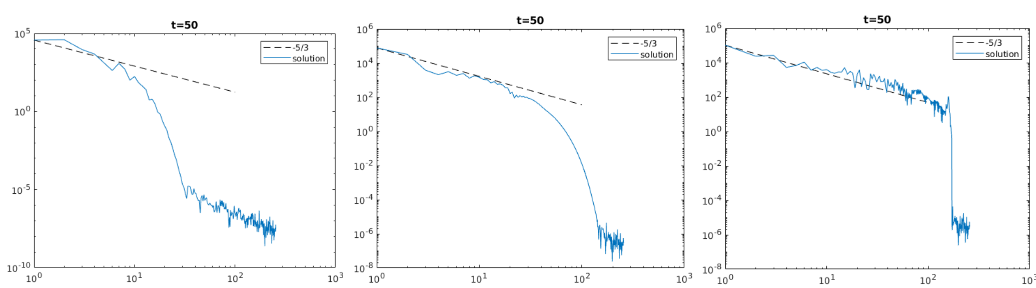
$\mu = \mathcal{N}(0, 7^{3/2}(-\Delta + 49I)^{-2.5})$ with periodic boundary conditions. The forcing is kept fixed $f(x) = 0.1(\sin(2\pi(x_1+x_2))+\cos(2\pi(x_1+x_2)))$. The equation is solved using the stream-function formulation with a pseudospectral method. First a Poisson equation is solved in Fourier space to find the velocity field. Then the vorticity is differentiated and the non-linear term is computed is physical space after which it is dealiased. Time is advanced with a Crank–Nicolson update where the non-linear term does not enter the implicit part. All data are generated on a 256×256 grid and are downsampled to 64×64. We use a time-step of $1\text{e}{-4}$ for the Crank–Nicolson scheme in the data-generated process where we record the solution every $t = 1$ time units. We experiment with the viscosities $\nu = 1\text{e}{-3}, 1\text{e}{-4}, 1\text{e}{-5}$, decreasing the final time T as the dynamic becomes chaotic.

6.4.1 Bayesian Inverse Problem

Further, we want to do the inverse problem of the Navier-Stokes Equation defined above with the learned surrogate model of the forward map $\mathcal{G}^\dagger : w|_{t=0} \mapsto w|_{t=T}$. Given a later time solution $w|_{t=T}$, we want to recover the initial condition $w|_{t=0}$ using a function space Markov chain Monte Carlo (MCMC) method Cotter et al. (2013) to draw samples from the posterior distribution of the initial vorticity in Navier-Stokes 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.

6.4.2 Spectral Analysis

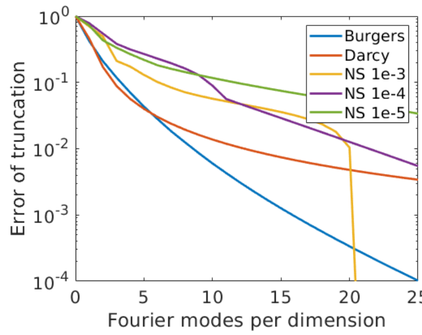
The spectral decay of the Navier Stokes equation data is shown in Figure 5. The spectrum decay has a slope $k^{-5/3}$, matching the energy spectrum in the turbulence region. And we notice the energy spectrum does not decay along with the time.



The spectral decay of the Navier-stokes equation data. The y-axis is the spectrum; the x-axis is the wavenumber $|k| = k_1 + k_2$.

Figure 5: Spectral Decay of Navier-Stokes equations

We also present the spectral decay in term of the truncation k_{max} defined in 5.4 as shown in Figure6. We note all equations (Burgers, Darcy, and Navier-Stokes with $\nu \leq 1\text{e}{-4}$) exhibit high frequency modes. Even we truncate at $k_{max} = 12$ in the Fourier layer, the Fourier neural operator can recover the high frequency modes.



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}.

Figure 6: Spectral Decay in term of k_{max}

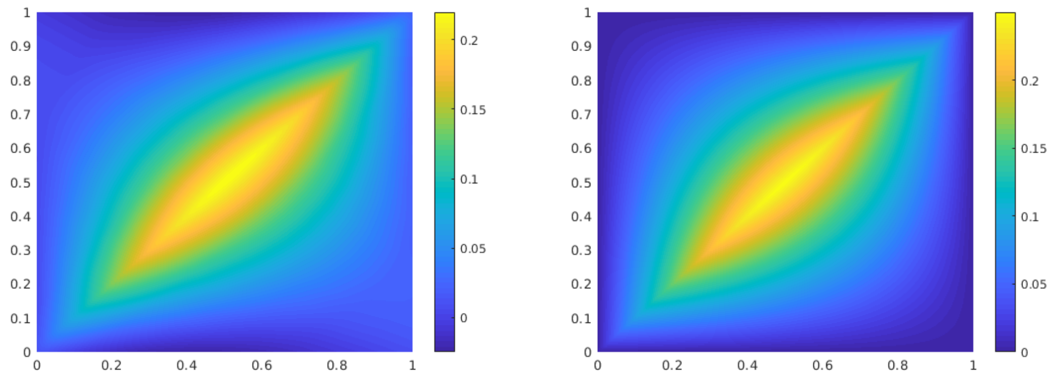
7. Numerical results

In this section, we compare the proposed neural operator with multiple finite-dimensional architectures as well as operator-based approximation methods on the 1-d Burgers' equation, the 2-d Darcy Flow problem, and 2-d Navier-Stokes equation. We do not compare against traditional solvers (FEM/FDM) or neural-FEM type methods since our goal is to produce an efficient operator approximation that can be used for downstream applications. We demonstrate one such application to the Bayesian inverse problem in Section 7.3.4. Lower resolution data are downsampled from higher resolution. The code are available at `https://github.com/zongyi-li/graph-pde` and `https://github.com/zongyi-li/fourier_neural_operator`. All the computation is carried on a single Nvidia V100 GPU with 16GB memory.

Setup of the four methods: We construct the neural operator by stacking four integral operator layers as specified in (**??**) with the ReLU activation as well as batch normalization. Unless otherwise specified, we use $N = 1000$ training instances and 200 testing instances. We use 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 dimension $d_v = 64$ for the 1-d problem and $d_v = 32$ for the 2-d problems. The kernel network κ are set to three layers of standard feedforward neural networks with width up do 256.

- **GNO:** the Nyström method **??** with truncation radius $r = 0.25$ and sampling nodes $m = 300$.
- **LNO:** the low-rank method 5.2 with rank $rank = 4$. Because of the channel space, it is usually not necessary to have additional ranks.
- **MGNO:** the multipole method 5.3. On Darcy flow we use the random construction with three levels of graph, each sampling $m_1 = 400, m_2 = 100, m_3 = 25$ nodes; on Burgers' equation we use the orthogonal construction without sampling.
- **FNO:** the Fourier method 5.4. We set $k_{\max,j} = 16$ for the 1-d problem and $k_{\max,j} = 12$ for the 2-d 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. On the



left: learned kernel function; **right:** the analytic Green's funciton.
This is a proof of concept of the graph kernel network on 1 dimensional Poisson equation and the comparison of learned and truth kernel.

Figure 7: Kernel for one-dimensional Green's function, with the Nystrom approximation method

other hand, operator approximation is independent of the ways its data is discretized as long as all relevant information is resolved. Resolution-invariant operators have consistent error rates among different resolutions as shown in Figure 8. Further, resolution-invariant operators can do zero-shot super-resolution, as shown in Section 7.3.1.

7.1 Poisson equation and Green's Function

First, as a illustration, we want to show with a specific reduction of the architecture we recover Green's function. We use the 1d poisson equation defined in section 6.1 as the test function, because it is simple enough to have an analytic Green function, and also because in the 1d case, we can visualize the kernel as a 2d image. As discussed in section 6.1, the true Green function has the form:

$$G(x,y) = \frac{1}{2}\left(x + y - |y - x|\right) - xy.$$

so that $u(x) = \int_0^1 G(x,y)f(y)\mathrm{d}y$. It is visualized on the right of Figure 7.

Again, for this problem, we consider a simplification of the iterative update framework defined in section 3. We omit step 1.,2. and 4. and only use one iteration of step 3. In other words, we set $v_0(x) = f(x)$, $T = 1$, $n = 1$, $\sigma(x) = x$, $W = w = 0$, and $\nu_x(dy) = dy$ (the Lebesgue measure). The purposed operator $\mathcal{G}$ can be directly written as

$$(\mathcal{G}f)(x) = \int_0^1 \kappa(x,y;\phi)f(y)\mathrm{d}y$$

For this test problem, we want to show whether the neural network κ will match the Green function G.

The Nyström approximation method has an test error rate of $1\mathrm{e}{-7}$ (**shall use relative error**). As shown in Figure 7, the learned kernel function $\kappa(x,y)$ is very close to the true Green function G. It can be seems that the learned kernel is "shallower" and "wider" compared to the true Green's function ($\kappa(0.5, 0.5) \approx 0.22$, $G(0.5, 0.5) = 0.25$). Because the learn kernel has near zero test error, it implies the learned “shallower” and “wider” is also a near optimal solution for this equation. It can be seems, the neural networks kernel well approximated the true kernel (the Green’s function).

7.2 Time-independent Equations

In the following section, we compare our four methods with different benchmarks on the Darcy flow 6.2 and Burgers’ equation 6.3 which are formulated in a time-independent way.

Benchmarks for time-independent problems (Burgers and Darcy):

- **NN** is a simple point-wise feedforward neural network. It is mesh-free, but performs badly due to lack of neighbor information.

- **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 data and interpolating the latent spaces with a neural network. 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.

- **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). It has the best 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.

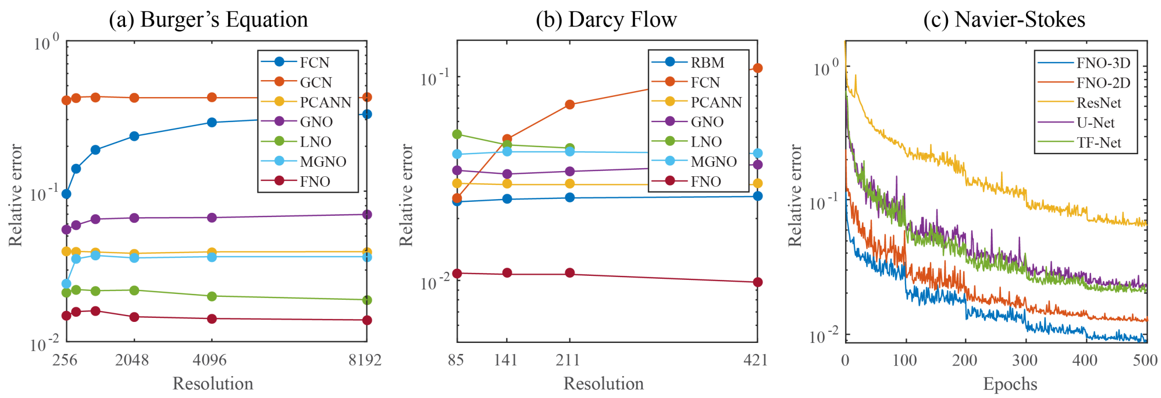
- **DeepONet** is the Deep Operator network Lu et al. (2019) that has a nice approximation guarantee. We use the unstacked version with width 200.

7.2.1 Darcy Flow

The results of the experiments on Darcy flow are shown in Figure 8 and Table 1. The Fourier neural operator (FNO) obtains the lowest relative error compared to any of the benchmarks. Further, the error is invariant with the resolution, while the error of convolution neural network based methods (FCN) grows with the resolution. Compared to other neural operator methods such as GNO and MGNO that use Nyström sampling in physical space, the Fourier neural operator is both more accurate and more computationally efficient.

7.2.2 Burgers’ Equation

The results of the experiments on Burgers’ equation are shown in Figure 8 and Table 2. Similarly, the Fourier neural operator obtains nearly one order of magnitude lower relative error compared to any benchmarks. We again observe the invariance of the error with respect to the resolution.



Left: benchmarks on Burgers equation; **Mid:** benchmarks on Darcy Flow for different resolutions; **Right:** the learning curves on Navier-Stokes $\nu = 1\mathrm{e}{-3}$ with different benchmarks. Train and test on the same resolution. For acronyms, see Section 7; details in Tables 3, 2, 1.

Figure 8: Benchmark on Burger's equation, Darcy Flow, and Navier-Stokes

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 1: Benchmarks on 2-d Darcy Flow

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 show 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-time.

7.3 Time-dependent Equation

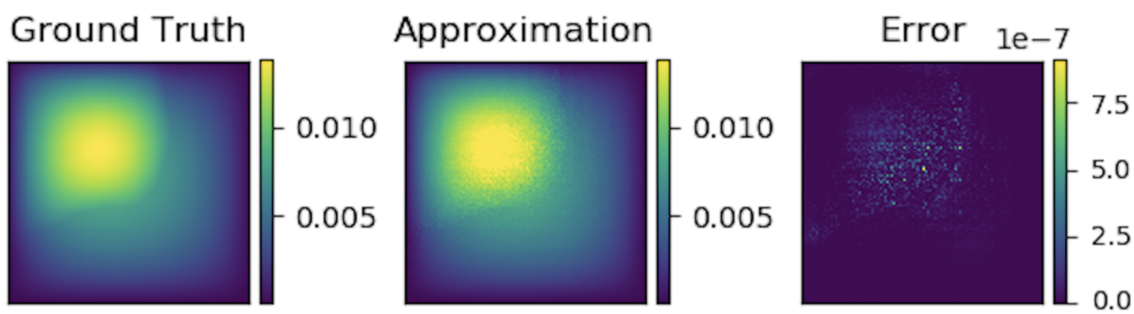
In the following section, we compare our four methods with different benchmarks on the Navier-Stokes equation 6.4

Benchmarks for time-dependent problems (Navier-Stokes):

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

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	**0.0149**	**0.0158**	**0.0160**	**0.0146**	**0.0142**	**0.0139**

Table 2: Benchmarks on 1-d Burgers' equation



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.

Figure 9: Darcy, trained on 16×16, tested on 241×241

- **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). The network is designed to model velocity. It may have a disadvantage on the Navier-Stokes equation formulated in vorticity.
- **FNO-2d:** 2-d Fourier neural operator with a RNN structure in time. We set and $k_{\max,j} = 12, d_v = 32$.
- **FNO-3d:** 3-d Fourier neural operator that directly convolves in space-time. We set and $k_{\max,j} = 12, d_v = 32$.

As shown in Table 3, the FNO-3D has the best performance when there is sufficient data ($\nu = 1\text{e}{-3}, N = 1000$ and $\nu = 1\text{e}{-4}, N = 10000$). For the configurations where the amount of data is insufficient ($\nu = 1\text{e}{-4}, N = 1000$ and $\nu = 1\text{e}{-5}, N = 1000$), all methods have $> 15\%$ error with FNO-2D achieving the lowest. Note that we only present results for spatial resolution 64×64 since all benchmarks we compare against are designed for this resolution. Increasing it degrades their performance while FNO achieves the same errors.

Table 3: Benchmarks on Navier Stokes (fixing resolution 64×64 for both training and testing)

Config	**Parameters**	**Time per epoch**	$\nu = 1\text{e}{-3}$ $T = 50$ $N = 1000$	$\nu = 1\text{e}{-4}$ $T = 30$ $N = 1000$	$\nu = 1\text{e}{-4}$ $T = 30$ $N = 10000$	$\nu = 1\text{e}{-5}$ $T = 20$ $N = 1000$
FNO-3D	$6,558,537$	$38.99s$	$\mathbf{0.0086}$	0.1918	$\mathbf{0.0820}$	0.1893
FNO-2D	$414,517$	$127.80s$	0.0128	$\mathbf{0.1559}$	0.0834	$\mathbf{0.1556}$
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

2D and 3D Convolutions. FNO-2D, U-Net, TF-Net, and ResNet all do 2D-convolution in the 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). On the other hand, FNO-3D performs convolution in space-time. It maps the initial time steps 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 3-d method to be more expressive and easier to train compared to its RNN-structured counterpart.

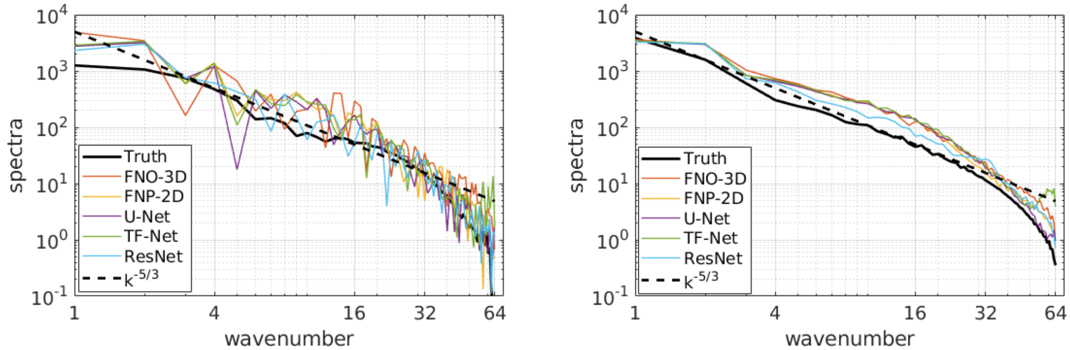
Networks	$s = 64$	$s = 128$	$s = 256$
FNO-3D	0.0098	0.0101	0.0106
FNO-2D	0.0129	0.0128	0.0126
U-Net	0.0253	0.0289	0.0344
TF-Net	0.0277	0.0278	0.0301

2D Navier-Stokes Equation with the parameter $\nu = 1\text{e}{-3}$, $N = 200$, $T = 20$.

Table 4: Resolution study on Navier-stokes equation

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 1 show an example where we train the FNO-3D model on $64 \times 64 \times 20$ resolution data in the setting above with ($\nu = 1\text{e}{-4}, N = 10000$) and transfer to $256 \times 256 \times 80$ resolution, demonstrating super-resolution in space-time. 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. And surprisingly, it can do super-resolution not only in the spatial domain but also in the temporal domain.



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 10: The spectral decay of the predictions of different methods

7.3.2 Spectral analysis.

As shown in Figure 10, all the methods are able to capture the spectral decay of the Navier-Stokes equation.

Notice, even the Fourier method truncates the higher frequency modes during the convolution, FNO can still recover the higher frequency component in the final prediction. Due to the way we parameterize R_ϕ, the function output by (**??**) 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 occur between integral operators and the final decoder network Q recover the high frequency modes. As an example, consider a solution to the Navier-Stokes equation with viscosity $\nu = 1\text{e}{-3}$. Truncating this function at 20 Fourier modes yields an error around 2% as shown in Figure 6, 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.

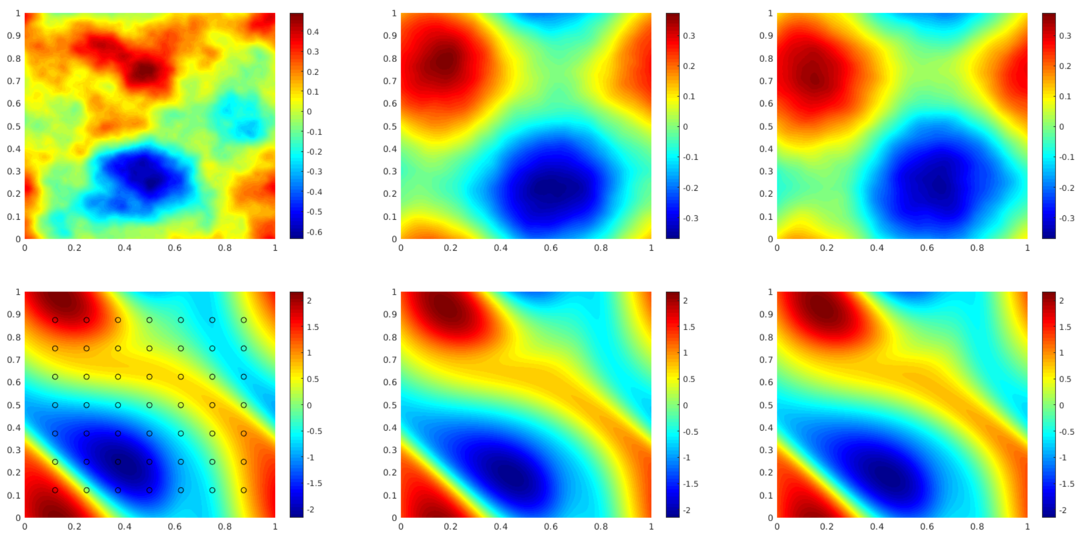
7.3.3 Non-periodic boundary condition.

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 discussion in Section 6.4.1, we use a function space Markov chain Monte Carlo (MCMC) method Cotter et al. (2013) to draw samples from the posterior distribution of the initial vorticity in Navier-Stokes 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 11, FNO and the traditional solver recover almost the same posterior mean which, when pushed forward, recovers well the late-time dynamic of Navier Stokes. 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 without the need for the adjoint method.



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.

Figure 11: Results of the Bayesian inverse problem for the Navier-Stokes equation.

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 restrict the integration to be a convolution.

GNO and MGNO are implemented using graph neural network, which helps to define sampling and integration. 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.

GNO, LNO, and MGNO all have the kernel network κ resambling the Green's function representation, while FNO doesn't. It's because FNO does the convolution on the frequency domain and it's not efficient to define the network on the frequency mode. In practice, FNO is lightweight and faster to train.

	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 5: Ingenuity

7.4.2 Expressiveness

We measure the expressiveness by the training and testing error of the method. The full $O(n^2)$ 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 have sampling. 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. But it may has limitation on non-periodic boundary condition.

7.4.3 Complexity

The complexity of the four methods are list in Table 6. GNO and MGNO have sampling. Their complexity depend on the number of nodes sampled m. When using the full nodes. They are still quadratic. LNO has the lowest complexity $O(n)$. FNO, when using the fast Fourier transform, has complexity $O(n \log n)$.

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.

7.4.4 Refinability

Refineability measures the number of parameters used in the framework. Table 7 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 of parameters. On the other hand, FNO does not have the sub-network. It needs at a larger magnitude of parameters to have a decent error rate.

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

The theoretical time complexity and the empirical time complexity. Roundup to second (on a single Nvidia V100 GPU).

Table 6: Complexity

Number of parameters	1e+3	1e+4	1e+5	1e+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

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 1e+3, 1e+4, 1e+5, 1e+6 respectively.

Table 7: Refinability

8. 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 of features, (ii) iterate a recurrent neural network on feature space, defined through composition of a sigmoid function and a nonlocal operators, and (iii) a final mapping from feature space into the output function space.

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 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 method can achieve competitive performance with other mesh-free approaches developed in the numerical analysis community, and that it beats state-of-the-art neural network approaches on large grids, which are mesh-dependent. 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.

8.1 Future directions

We foresee two 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; and secondly the development of more advanced methodologies beyond the four approximation schemes presented in the paper that are more efficient or better in specific situations; lastly it's crucial to develop an underpinning theory which captures the expressive power, and approximation error properties, of the proposed neural network.

8.1.1 New Applications

The proposed neural operator is a blackbox surrogate model for function-to-function mapping. It's naturally fit 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 board genres of scenarios. Due to its blackbox structure, the neural operator is easy to be applied on other problems. We foresee applications on more challenging turbulence flow such as climate models, sharper coefficients contrast raising in geological models, 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 math and science. For example, in computer vision, images can naturally be viewed as real-valued functions on 2-d domains and videos simply add a temporal structure. Our approach is therefore a natural choice for problems in computer vision where invariance to discretization crucial. We leave this as an interesting future direction.

8.1.2 New Methodologies

There is plenty of room for improvement upon the current methodologies given its excellent performance. The full $O(n^2)$ integration still has about 40% lead than the Fourier method. It is worthy well to develop more advanced integration quadrature 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 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 accuracy approximation. One can also combine neural operator with Physics-informed neural network (PINN) Raissi et al. (2019), using neural operators to generate a context grid that helps the PINNs.

8.1.3 Theory

Finally, the works in this paper are mostly empirical. We managed to develop a approximation bound following Lu et al. (2019) with the assumptions of compactness of the input and output space, but the general approximation power of neural operator is still unclear. For functions, we clearly know, by combining two layers of linear functions with one layer of non-linear activation function, the neural network can approximate arbitrary continuous functions. However, the approximation theory of operator is much more complex and challenging. It's important to study the class of neural operators–what space of operators or which PDEs can neural operator approximate and its efficiency. We leave this as an exciting, but, long-term goal.

Acknowledgements

Z. Li gratefully acknowledges the financial support from the Kortschak Scholars Program. A. Anandkumar is supported in part by Bren endowed chair, LwLL grants, Beyond Limits, Raytheon, Microsoft, Google, Adobe faculty fellowships, and DE Logi grant. K. Bhattacharya, N. B. Kovachki, B. Liu and A. M. Stuart gratefully acknowledge the financial support of the Army Research Laboratory through the Cooperative Agreement Number W911NF-12-0022. Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-12-2-0022. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

The computations presented here were conducted on the Caltech High Performance Cluster, partially supported by a grant from the Gordon and Betty Moore Foundation.

References

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

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`.

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.

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 pde(s). *arXiv preprint arXiv:2005.03180*, 2020.

Gilles Blanchard, Olivier Bousquet, and Laurent Zwald. Statistical properties of kernel principal component analysis. *Machine Learning*, 66(2):259–294, Mar 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.

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.

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.

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.

John B. Conway. *A Course in Functional Analysis*. Springer-Verlag New York, 1985.

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`.

F. Dadipour, M.S. Moslehian, J.M. Rassias, and S. Takahasi. Characterization of a generalized triangle inequality in normed spaces. *Nonlinear Analysis: Theory, Methods & Applications*, 75 (2):735–741, 2012.

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

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. URL `https://epubs.siam.org/doi/abs/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.

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.

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.

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.

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. *arXiv preprint arXiv:1707.03351*, 2017.

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.

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.

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.

Pierre Gilles Lemarié-Rieusset. *The Navier-Stokes problem in the 21st century.* CRC Press, 2018.

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.

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.

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.

NH Nelsen and AM Stuart. The random feature model for input-output maps between banach spaces. *arXiv preprint arXiv:2005.10224*, 2020.

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.

Ravi G Patel, Nathaniel A Trask, Mitchell A Wood, and Eric C Cyr. A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373:113500, 2021.

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.

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.

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.

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.

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.

Steven Strogatz. Loves me, loves me not (do the math). 2009.

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

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.

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.

Joseph D. Ward. Chebyshev centers in spaces of continuous functions. *Pacific Journal of Mathematics*, 52(1):283 – 287, 1974.

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

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.

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 coefficient functions
$u \in \mathcal{U} = (D; \mathbb{R}^{d_u})$	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 time steps (layers)
$\mathcal{P}, \mathcal{Q}$	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.

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.

Table 8: Table of notations: operator learning and neural operators

Appendix B. Method of Chen and Chen

We will assume $D \subset \mathbb{R}^d$ is a compact domain and $V \subset C(D; \mathbb{R})$ is a compact set. Lemma 6 allows us to find representers for V. In particular, there exist continuous linear functionals $\{c_j\}_{j=1}^\infty \subset C(V; \mathbb{R})$ known as the *coordinate functionals* and elements $\{\varphi_j\}_{j=1}^\infty \subset V$ known as the *representers* such that any $v \in V$ may be written as

$$v = \sum_{j=1}^{\infty} c_j(v)\varphi_j. \tag{43}$$

Note that the result of Lemma 6 is stronger than (43) as the approximation is uniform in v. The first observation of Chen and Chen (1995) is that we may find representers that are scales and shifts of bounded, non-polynomial maps. In particular, fix $\sigma : \mathbb{R} \to \mathbb{R}$ as some bounded, non-polynomial map, then there exist vectors $\{w_j\}_{j=1}^\infty$ and numbers $\{b_j\}_{j=1}^\infty$ such that

$$\varphi_j(x) = \sigma(\langle x, w_j \rangle + b_j) \qquad \forall x \in D. \tag{44}$$

The weights w_j and biases b_j are independent of any particular function $v \in V$, but may depend on the set V as a whole. Combining (43) and (44) yields a universal approximation result for single layer neural networks (Chen and Chen, 1995, Theorem 3) that while being uniform in v is restricted to compact sets of $C(D;\mathbb{R})$.

With this result at hand, we now turn our attention to approximating continuous, possibly non-linear, functionals $G \in C(V;\mathbb{R})$. The main idea of Chen and Chen (1995) is to work not with the functions $v \in V$ directly but rather some appropriately defined coordinate functionals. In particular, it is shown that there exists a compact extension U of V such that any $u \in U$ may be written as

$$u = \sum_{j=1}^{\infty} u(x_j)\varphi_j \tag{45}$$

for some fixed set of points $\{x_j\}_{j=1}^{\infty} \subset D$ and appropriately defined representers $\{\varphi_j\}_{j=1}^{\infty} \subset U$. That is, the coordinate functionals c_j from (43) can be chosen as evaluation functionals for a fixed set of points by moving to the larger compact set U. Similarly to before, the result is stronger than (45) since it is uniform in u. It may be thought of as a generalization to the classical semi-discrete Fourier transform restricted to compact sets of the function space. For simplicity of the current discussion, we will not longer consider the set U and make all definitions directly on V which is possible since $V \subset U$. Once we fix the set of representers $\{\varphi_j\}_{j=1}^{\infty}$, we are free to move between the function representation of $v \in V$ and its sequence representation through the mapping $F : V \to \mathcal{V}$ which we define as

$$F(v) = (v(x_1), v(x_2), v(x_3), \dots) \qquad \forall v \in V \tag{46}$$

where $\mathcal{V} \subset \ell^{\infty}(\mathbb{N};\mathbb{R})$ is a compact set. Compactness of $\mathcal{V}$ follows from compactness of V and continuity of F, in particular, $\mathcal{V} = F(V)$. We choose to work in the induced topology of $\ell^{\infty}(\mathbb{N};\mathbb{R})$ since it is similar to the topology of $C(D;\mathbb{R})$ and results such as the continuity of F are trivial. Furthermore, we define the inverse $F^{-1} : \mathcal{V} \to V$ by

$$F^{-1}(w) = \sum_{j=1}^{\infty} w_j \varphi_j \qquad \forall w \in \mathcal{V}. \tag{47}$$

The fact that F^{-1} is indeed the inverse to F follows from uniqueness of the representation (45). We may then consider any functional $G \in C(V;\mathbb{R})$ by

$$G(v) = G(F^{-1}(F(v))) \qquad \forall v \in V. \tag{48}$$

This is especially useful as it allows us to construct approximations to G by directly working with the coordinates of v. For each $n \in \mathbb{N}$, define the spaces

$$\mathcal{V}_n = \{(v(x_1), \dots, v(x_n), 0, 0, \dots) : v \in V\} \subset \mathcal{V}. \tag{49}$$

From compactness of $\mathcal{V}$, it follows that each $\mathcal{V}_n$ is isomorphic to a compact subset of $\mathbb{R}^n$. We may thus consider the sequence of restricted functionals $G_n : V \to \mathbb{R}$ defined by

$$G_n(v) = G(F^{-1}(F_n(v))) \tag{50}$$

where $F_n : V \to \mathcal{V}_n$ is given as $F_n(v) = (v(x_1), \dots, v(x_n), 0, \dots)$. Since $F_n(V) = \mathcal{V}_n$, we have the outer mapping $G \circ F^{-1} : \mathcal{V}_n \to \mathbb{R}$. In particular, when viewed in the definition of G_n, $G \circ F^{-1}$

is a continuous function defined on a compact set of $\mathbb{R}^n$. We may therefore use the previously established universal approximation theorem for functions to conclude

$$G_n(v) \approx \sum_{j=1}^{m} c_j \varphi_j(F_n(v)) = \sum_{j=1}^{m} c_j \sigma(\langle w_j, (v(x_1), \dots, v(x_n)) \rangle + b_j).$$

The argument is finished by establishing closeness of G to G_n (Chen and Chen, 1995, Theorem 4).

Lastly, we consider approximating operators $\mathcal{G} \in C(V; C(D; \mathbb{R}))$. We note that continuity of $\mathcal{G}$, implies $\mathcal{G}(V) \subset C(D; \mathbb{R})$ is compact. Therefore we may use the universal approximation theorem for functions to conclude that

$$\mathcal{G}(v)(x) \approx \sum_{j=1}^{n} c_j(\mathcal{G}(v)) \sigma(\langle w_j, x \rangle + b_j) \qquad \forall x \in D.$$

Note that we can view each $c_j \in C(V; \mathbb{R})$ by re-defining $c_j(v) = c_j(\mathcal{G}(v))$ noting that continuity is preserved by the composition. We then repeatedly apply the universal approximation theorem for functionals to find

$$c_j(v) \approx \sum_{k=1}^{m} a_{jk} \sigma(\langle \xi_{jk}, (v(x_1), \dots, v(x_p)) \rangle + q_{jk})$$

and therefore

$$\mathcal{G}(v)(x) \approx \sum_{j=1}^{n} \sum_{k=1}^{m} a_{jk} \sigma(\langle \xi_{jk}, (v(x_1), \dots, v(x_p)) \rangle + q_{jk}) \sigma(\langle w_j, x \rangle + b_j) \qquad \forall x \in D.$$

This result is established rigorously in (Chen and Chen, 1995, Theorem 5).

B.1 Supporting Results

Lemma 6 *Let $\mathcal{X}$ be a Banach space and $V \subseteq \mathcal{X}$ a compact set. Then, for any $\epsilon > 0$, there exists a number $n = n(\epsilon) \in \mathbb{N}$, continuous, linear, functionals $G_1, \dots, G_n \in C(V; \mathbb{R})$, and elements $\varphi_1, \dots, \varphi_n \in V$ such that such that*

$$\sup_{v \in V} \Big\| v - \sum_{j=1}^{n} G_j(v) \varphi_j \Big\|_{\mathcal{X}} < \epsilon.$$

Proof Since V is compact, we may find nested finite dimensional spaces $V_1 \subset V_2 \subset \dots$ with $\dim(V_n) = n$ for $n = 1, 2, \dots$ such that

$$\lim_{n \to \infty} \sup_{u \in V} \min_{v \in V_n} \|u - v\|_{\mathcal{X}} = 0. \tag{51}$$

Since the spaces V_n are finite dimensional, they admit a Schauder basis, in particular, there are sequences $\{(\varphi_{j,n})_{j=1}^n\}_{n=1}^\infty$ of elements $\varphi_{j,n} \in V$ and $\{(G_{j,n})_{j=1}^n\}_{n=1}^\infty$ of functionals $G_{j,n} \in C(V; \mathbb{R})$ such that for any $n \in \mathbb{N}$, any $v \in V_n$ can be uniquely written as

$$v = \sum_{j=1}^{n} G_{j,n}(v) \varphi_{j,n}.$$

Let $v \in V$ be arbitrary and define $v_n \in V_n$ to be the best approximation of v from V_n for $n = 1, 2, \dots$ and set $v_0 = 0$. Then, by (51), for any $\epsilon > 0$, we can always find $n = n(\epsilon) \in \mathbb{N}$ that is independent of v such that

$$\Big\| v - \sum_{j=1}^{n} (v_j - v_{j-1}) \Big\|_{\mathcal{X}} < \epsilon.$$

Since $v_j - v_{j-1} \in V_j$ for $j = 1, \dots, n$, it may be written as a linear combination of elements $\varphi_{k,j}$ with coefficients $G_{k,j}(v_j - v_{j-1})$ for $k = 1, \dots, j$ and the result follows. ■

Lemma 7 *Let $\mathcal{X}$ be a Banach space, $V \subseteq \mathcal{X}$ a compact set, and $U \subset \mathcal{X}$ a dense set. Then, for any $\epsilon > 0$, there exists a number $n = n(\epsilon) \in \mathbb{N}$, continuous, linear, functionals $G_1, \dots, G_n \in C(V; \mathbb{R})$, and elements $\varphi_1, \dots, \varphi_n \in U$ such that such that*

$$\sup_{v \in V} \Big\| v - \sum_{j=1}^{n} G_j(v) \varphi_j \Big\|_{\mathcal{X}} < \epsilon.$$

Proof Apply Lemma 6 to find a number $n \in \mathbb{N}$, continuous, linear, functionals $G_1, \dots, G_n \in C(V; \mathbb{R})$, and elements $\psi_1, \dots, \psi_n \in V$ such that

$$\sup_{v \in V} \Big\| v - \sum_{j=1}^{n} G_j(v) \psi_j \Big\|_{\mathcal{X}} < \frac{\epsilon}{2}.$$

By continuity, the sets $G_j(V) \subset \mathbb{R}$ are compact hence we can find a number $M > 0$ such that

$$\sup_{v \in V} \max_{j=1,\dots,n} |G_j(v)| < M.$$

By density of U, we can find elements $\varphi_1, \dots, \varphi_n \in U$ such that

$$\|\varphi_j - \psi_j\|_{\mathcal{X}} < \frac{\epsilon}{2nM}, \qquad \forall j \in \{1, \dots, n\}.$$

By triangle inequality,

$$\begin{aligned}
\sup_{v \in V} \Big\| v - \sum_{j=1}^{n} G_j(v) \varphi_j \Big\|_{\mathcal{X}} &\leq \sup_{v \in V} \Big\| v - \sum_{j=1}^{n} G_j(v) \psi_j \Big\|_{\mathcal{X}} + \sup_{v \in V} \Big\| \sum_{j=1}^{n} G_j(v) \psi_j - \sum_{j=1}^{n} G_j(v) \varphi_j \Big\|_{\mathcal{X}} \\
&\leq \frac{\epsilon}{2} + \sup_{v \in V} \sum_{j=1}^{n} |G_j(v)| \|\psi_j - \varphi_j\|_{\mathcal{X}} \\
&< \frac{\epsilon}{2} + \frac{\epsilon}{2nM} nM \\
&= \epsilon
\end{aligned}$$

as desired. ■

Lemma 8 *Let $D \subseteq \mathbb{R}^d$ be a compact domain, and $\kappa \in C(D \times D;\mathbb{R})$ and $b \in C(D;\mathbb{R})$ be continuous functions. Furthermore let $\sigma : \mathbb{R} \to \mathbb{R}$ be α-Hölder continuous for some $\alpha > 0$ then the operator $P : C(D;\mathbb{R}) \to C(D;\mathbb{R})$ defined by*

$$P(v)(x) = \sigma\left(\int_D \kappa(x,y)v(y)\,dy + b(x)\right) \qquad \forall x \in D, \quad \forall v \in C(D;\mathbb{R})$$

is α-Hölder continuous. In particular, there exists a constant $C > 0$ such that

$$\|P(v) - P(u)\|_{C(D;\mathbb{R})} \le C\|v-u\|^\alpha_{C(D;\mathbb{R})}.$$

Proof Since $D \times D$ is compact and κ is continuous, there is a number $M > 0$ such that

$$\sup_{x,y\in D} |\kappa(x,y)| \le M.$$

Then for any $u, v \in C(D;\mathbb{R})$, we have that there exists a constant $L > 0$ such that

$$\begin{aligned}\|P(v) - P(u)\|_{C(D;\mathbb{R})} &\le L \sup_{x\in D} \int_D |\kappa(x,y)|^\alpha |v(y) - u(y)|^\alpha \,\mathrm{d}y \\ &\le LM^\alpha |D|\, \|v-u\|^\alpha_{C(D;\mathbb{R})}\end{aligned}$$

as desired. ∎

Lemma 9 *Let $D \subseteq \mathbb{R}^d$ be a domain, $f_1, \dots, f_n \in C(D;\mathbb{R})$ a finite collection of continuous functions, and $x_1, \dots, x_n \in D$ a finite collection of distinct points. Then there exists a function $\kappa \in C(D \times D;\mathbb{R})$ such that*

$$\kappa(x_j, y) = f_j(y) \qquad \forall y \in D, \quad j = 1, \dots, n.$$

Furthermore if $f_1 \in C^{\alpha_1}(D;\mathbb{R}), \dots, f_n \in C^{\alpha_n}(D;\mathbb{R})$ then $\kappa \in C^\alpha(D \times D;\mathbb{R})$ where $\alpha = \min_{j\in\{1,\dots,n\}} \alpha_j$.

Proof Recursively define

$$\kappa_j(x,y) = t_j(x)\kappa_{j-1}(x,y) + (1 - t_j(x))f_{j+1}(y) \qquad \forall x, y \in D, \quad j \in \{1, \dots, n-1\}$$

where $\kappa_0(x,y) = f_1(y)$ and $t_j : D \to \mathbb{R}$ is the unique polynomial of degree at most j such that $t_j(x_{j+1}) = 0, t_j(x_j) = 1, t_j(x_{j-1}) = 1, \dots, t_j(x_1) = 1$. Existence and uniqueness of the polynomials $t_1, \dots, t_{n-1}$ is guaranteed by the interpolation theorem for polynomials since the points $x_1, \dots, x_n$ are assumed to be distinct. Furthermore since sums and products of continuous functions are continuous, $\kappa_j \in C(D \times D;\mathbb{R})$ for $j = 1, \dots, n-1$. Notice that, for any $j \in \{1, \dots, n\}$

$$\kappa_{n-1}(x_j, y) = t_{n-1}(x_j)\kappa_{n-2}(x_j, y) + (1 - t_{n-1}(x_j))f_n(y).$$

If $j = n$, then by definition, $t_{n-1}(x_j) = 0$ hence $\kappa_{n-1}(x_j, y) = f_n(y)$. If $j = n-1$ then by definition, $t_{n-1}(x_j) = 1$ hence

$$\kappa_{n-1}(x_j, y) = \kappa_{n-2}(x_j, y) = t_{n-2}(x_j)\kappa_{n-3}(x_j, y) + (1 - t_{n-2}(x_j))f_{n-1}(y) = f_{n-1}(y)$$

since, by definition, $t_{n-2}(x_j) = 0$. Continuing this by induction shows that setting

$$\kappa(x, y) = \kappa_{n-1}(x, y)$$

gives the desired construction. The fact that $\kappa \in C^{\alpha}(D \times D; \mathbb{R})$ follows immediately since $t_1, \dots, t_{n-1} \in C^{\infty}(D; \mathbb{R})$ by construction. ■

Lemma 10 *Let $D \subset \mathbb{R}^d$ be a domain and $V \subset C(D; \mathbb{R})$ be a compact set. Furthermore let $c_1, \dots, c_n \in \mathbb{R}$ be a finite collection of points, and $x_1, \dots, x_n \in \text{int}(D)$ be a finite collection of distinct points. Then, for any $\epsilon > 0$, there exists a smooth function $w \in C_c^{\infty}(D; \mathbb{R})$ such that*

$$\Big| \int_D w(x) v(x) \, dx - \sum_{j=1}^{n} c_j v(x_j) \Big| < \epsilon \qquad \forall v \in V.$$

Proof Define

$$c = \sum_{j=1}^{n} |c_j|.$$

Since V is compact, we can find a number $p = p(\epsilon, c) \in \mathbb{N}$ as well as functions $v_1, \dots, v_p \in U$ such that, for any $v \in V$, there exists a number $l \in \{1, \dots, p\}$ such that

$$\|v_l - v\|_{C(D;\mathbb{R})} = \sup_{x \in D} |v_l(x) - v(x)| < \frac{\epsilon}{3c}. \tag{52}$$

Pick $\gamma > 0$ so that $\bar{B}_{\gamma}(x_j) \cap \bar{B}_{\gamma}(x_k) = \emptyset$ whenever $j \neq k$ for all $j, k \in \{1, \dots, n\}$ and so that and $\bar{B}_{\gamma}(x_j) \subset D$ for $j = 1, \dots, n$. Let $\varphi \in C_c^{\infty}(\mathbb{R}^d; \mathbb{R})$ be the standard mollifier supported on $\bar{B}_{\gamma}(0)$. Since φ is a mollifier, we may find numbers $\alpha_1, \dots, \alpha_p \in \mathbb{R}_+$ such that, for any $l \in \{1, \dots, p\}$ we have

$$|\alpha^{-d} \int_D \varphi\left(\frac{x - x_j}{\alpha}\right) v_l(x) \, dx - v_l(x_j)| < \frac{\epsilon}{3c} \qquad \forall \alpha \leq \alpha_l, \quad j = 1, \dots, n. \tag{53}$$

Define $\alpha = \min_{l \in \{1,\dots,p\}} \alpha_l$ and notice that by applying triangle inequality and combining (52) and (53), we find that for any $v \in V$,

$$\begin{aligned}
\Big|\alpha^{-d} \int_D \varphi\left(\frac{x - x_j}{\alpha}\right) v(x) \, dx - v(x_j)\Big| &\leq \Big|v_l(x_j) - v(x_j)\Big| + \Big|\alpha^{-d} \int_D \varphi\left(\frac{x - x_j}{\alpha}\right) v(x) \, dx - v_l(x_j)\Big| \\
&< \frac{\epsilon}{3c} + \Big|\alpha^{-d} \int_D \varphi\left(\frac{x - x_j}{\alpha}\right) v_l(x) \, dx - v_l(x_j)\Big| \\
&\quad + \Big|\alpha^{-d} \int_D \varphi\left(\frac{x - x_j}{\alpha}\right) v(x) \, dx - \alpha^{-d} \int_D \varphi\left(\frac{x - x_j}{\alpha}\right) v_l(x) \, dx\Big| \\
&< \frac{2\epsilon}{3c} + \alpha^{-d} \int_D \varphi\left(\frac{x - x_j}{\alpha}\right) \, dx \|v - v_l\|_{C(D;\mathbb{R})} \\
&< \frac{\epsilon}{c}
\end{aligned} \tag{54}$$

Define

$$w(x) = \alpha^{-d} \sum_{j=1}^{n} c_j \varphi\left(\frac{x - x_j}{\alpha}\right) \qquad \forall x \in D$$

noting that, by construction, $w \in C_c^\infty(D;\mathbb{R})$. Finally, applying triangle inequality and using (54), we find that for any $v \in V$

$$\begin{aligned}
\Big| \int_D w(x)v(x)\, dx - \sum_{j=1}^{n} c_j v(x_j) \Big| &\leq \sum_{j=1}^{n} |c_j| \Big| \alpha^{-d} \int_D \varphi\left(\frac{x - x_j}{\alpha}\right) v(x)\, dx - v(x_j) \Big| \\
&< \sum_{j=1}^{n} |c_j| \cdot \frac{\epsilon}{c} \\
&= \epsilon
\end{aligned}$$

as desired. ■

Theorem 11 *Let $D \subset \mathbb{R}^d$ be a compact domain and $V \subset C(D;\mathbb{R})$ be a compact set. Let $\mathcal{G}^\dagger \in C(V;\mathbb{R})$ be a continuous functional and let $\sigma : \mathbb{R} \to \mathbb{R}$ be α-Hölder continuous for some $\alpha > 0$ and of the Tauber-Wiener class. Then, for any $\epsilon > 0$, there exists a smooth kernel $\kappa \in C^\infty(D \times D;\mathbb{R})$ and smooth functions $w, b \in C^\infty(D;\mathbb{R})$ such that*

$$\Big| \mathcal{G}^\dagger(v) - \int_D w(x)\sigma\left(\int_D \kappa(x,y)v(y)\, dy + b(x)\right)\, dx \Big| < \epsilon \qquad \forall v \in V.$$

Proof Applying (Chen and Chen, 1995, Theorem 4), we find integers $n, m \in \mathbb{N}$, distinct points $x_1, \dots, x_m \in \text{int}(D)$, as well as constants $w_j, b_j, \xi_{jk} \in \mathbb{R}$ for $j = 1, \dots, n$ and $k = 1, \dots, m$ such that

$$\Big| \mathcal{G}^\dagger(v) - \sum_{j=1}^{n} w_j \sigma\left(\sum_{k=1}^{m} \xi_{jk} v(x_k) + b_j\right) \Big| < \frac{\epsilon}{3} \qquad \forall v \in V. \tag{55}$$

Fix $z_1, \dots, z_n \in D$ to be arbitrary distinct points. Using the interpolation theorem for polynomials, we define $b \in C^\infty(D;\mathbb{R})$ to be the unique polynomial of degree at most $n-1$ such that $b_j = b(z_j)$ for $j = 1, \dots, n$. Let $L > 0$ be the Hölder constant of σ and suppose we can find a kernel $\kappa \in C^\infty(D \times D;\mathbb{R})$ so that for every $j \in \{1, \dots, n\}$ we have

$$\Big| \sum_{k=1}^{m} \xi_{jk} v(x_k) - \int_D \kappa(z_j, y)v(y)\, dy \Big| < \left(\frac{\epsilon}{3nL|w_j|}\right)^{1/\alpha}. \tag{56}$$

Define the operator $P : V \to C(D;\mathbb{R})$ by

$$P(v)(x) = \sigma\left(\int_D \kappa(x,y)v(y)\, dy + b(x)\right) \qquad \forall v \in V.$$

Applying Lemma 8, we find that P is continuous therefore $P(V)$ is compact since V is compact. We can therefore apply Lemma 10 to find $w \in C^\infty(D;\mathbb{R})$ such that for any $v \in V$

$$\Big| \int_D w(x)\sigma\left(\int_D \kappa(x,y)v(y)\, dy + b(x)\right)\, dx - \sum_{j=1}^{n} w_j \sigma\left(\int_D \kappa(z_j,y)v(y)\, dy + b(z_j)\right) \Big| < \frac{\epsilon}{3}. \tag{57}$$

Applying triangle inequality and combining (55), (56), and (57), we find

$$
\begin{aligned}
\Big|\mathcal{G}^\dagger(v) - \int_D w(x)\sigma\left(\int_D \kappa(x,y)v(y)\,dy + b(x)\right)\,dx\Big| \\
&\leq \Big|\mathcal{G}^\dagger(v) - \sum_{j=1}^n w_j\sigma\left(\sum_{k=1}^m \xi_{jk}v(x_k) + b_j\right)\Big| \\
&\quad + \Big|\sum_{j=1}^n w_j\sigma\left(\sum_{k=1}^m \xi_{jk}v(x_k) + b_j\right) \\
&\quad - \int_D w(x)\sigma\left(\int_D \kappa(x,y)v(y)\,dy + b(x)\right)\,dx\Big| \\
&< \frac{\epsilon}{3} + \Big|\sum_{j=1}^n w_j\sigma\left(\sum_{k=1}^m \xi_{jk}v(x_k) + b_j\right) \\
&\quad - \sum_{j=1}^n w_j\sigma\left(\int_D \kappa(z_j,y)v(y)\,dy + b(z_j)\right)\Big| \\
&\quad + \Big|\sum_{j=1}^n w_j\sigma\left(\int_D \kappa(z_j,y)v(y)\,dy + b(z_j)\right) \\
&\quad - \int_D w(x)\sigma\left(\int_D \kappa(x,y)v(y)\,dy + b(x)\right)\,dx\Big| \\
&< \frac{2\epsilon}{3} + L\sum_{j=1}^n |w_j|\Big|\sum_{k=1}^m \xi_{jk}v(x_k) - \int_D \kappa(z_j,y)v(y)\,dy\Big|^\alpha \\
&< \epsilon.
\end{aligned}
$$

All that is left to do is find a smooth kernel κ satisfying (56). By repeatedly applying Lemma 10, we can find functions $\kappa_1, \dots, \kappa_n \in C^\infty(D;\mathbb{R})$ such that for $j = 1, \dots, n$, we have

$$
\Big|\sum_{k=1}^m \xi_{jk}v(x_k) - \int_D \kappa_j(y)v(y)\,dy\Big| < \left(\frac{\epsilon}{3nL|w_j|}\right)^{1/\alpha}.
$$

Applying Lemma 9 to the functions $\kappa_1, \dots, \kappa_n$ we find $\kappa \in C^\infty(D \times D;\mathbb{R})$ such that

$$
\kappa(z_j, y) = \kappa_j(y), \qquad \forall y \in D, \quad j = 1, \dots, n
$$

therefore (56) holds by construction and the proof is complete. ∎

Appendix C. Proof of Theorem 4

Proof (of Theorem 4) Apply Theorem 5 to find a number $n \in \mathbb{N}$ and functions $\tilde{\kappa}_1^{(1)}, \dots, \tilde{\kappa}_n^{(1)} \in C(D' \times D;\mathbb{R})$, $\tilde{\kappa}_1^{(0)}, \dots, \tilde{\kappa}_n^{(0)} \in C(D \times D;\mathbb{R})$, $\tilde{b}_1, \dots, \tilde{b}_n \in C(D;\mathbb{R})$ such that

$$
\sup_{a\in K}\sup_{x\in D'}\Big|\mathcal{G}^\dagger(a)(x) - \sum_{j=1}^n \int_D \tilde{\kappa}_j^{(1)}(x,y)\sigma\left(\int_D \tilde{\kappa}_j^{(0)}(y,z)a(z)\,\mathrm{d}z + \tilde{b}_j(y)\right)\,\mathrm{d}y\Big| < \epsilon.
$$

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

$$\sup_{a \in K} \|a\|_{C(D;\mathbb{R})} = \sup_{a \in K} \sup_{x \in D} |a(x)| < M.$$

We can find a neural network $\mathcal{P} : \mathbb{R} \to \mathbb{R}^n$ such that

$$\sup_{x \in [-M,M]} \|\mathcal{P}(x) - \tilde{\mathcal{P}}(x)\|_{\mathbb{R}^n} < \epsilon$$

where $\tilde{\mathcal{P}}(x) = (x, \dots, x) \in \mathbb{R}^n$. Since the domains $D' \times D$, $D \times D$, and D are compact, we can find neural networks $\kappa^{(1)} : \mathbb{R}^{d'} \times \mathbb{R}^d \to \mathbb{R}^{n \times n}$, $\kappa^{(0)} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{n \times n}$, $b : \mathbb{R}^d \to \mathbb{R}^n$ such that

$$\begin{aligned}
&\sup_{(x,y) \in D' \times D} |\kappa^{(1)}_{jj}(x,y) - \tilde{\kappa}^{(1)}_j(x,y)| < \epsilon, \quad j = 1, \dots, n, \\
&\sup_{(x,y) \in D' \times D} |\kappa^{(1)}_{jk}(x,y)| < \frac{\epsilon}{n}, \quad j,k = 1, \dots, n, \ j \neq k
\end{aligned}$$

and

$$\begin{aligned}
&\sup_{(y,z) \in D \times D} |\kappa^{(0)}_{jj}(y,z) - \tilde{\kappa}^{(0)}_j(y,z)| < \epsilon, \quad j = 1, \dots, n, \\
&\sup_{(y,z) \in D \times D} |\kappa^{(0)}_{jk}(y,z)| < \frac{\epsilon}{n^\beta}, \quad j,k = 1, \dots, n, \ j \neq k
\end{aligned}$$

and

$$\sup_{y \in D} |b_j(y) - \tilde{b}_j(y)| < \epsilon, \quad j = 1, \dots, n$$

where $\beta > 0$ is to be determined later. Define the operator $S : C(D;\mathbb{R}^n) \to C(D';\mathbb{R}^n)$ by

$$S(f)_k(x) = \sum_{l=1}^n \int_{D'} \kappa^{(1)}_{kl}(x,y) \sigma \left(\int_D \sum_{j=1}^n \kappa^{(0)}_{lj}(y,z) f_j(z) \, \mathrm{d}z + b_l(y) \right) \mathrm{d}y, \quad \forall x \in D'$$

for $k = 1, \dots, n$ and any $f \in C(D;\mathbb{R}^n)$ where $f = (f_1, \dots, f_n)$. Likewise, define $\tilde{S} : C(D;\mathbb{R}^n) \to C(D';\mathbb{R}^n)$ by

$$\tilde{S}(f)_k(x) = \int_D \tilde{\kappa}^{(1)}_k(x,y) \sigma \left(\int_D \tilde{\kappa}^{(0)}_k(y,z) f_k(z) \, \mathrm{d}z + \tilde{b}_k(y) \right) \mathrm{d}y.$$

Since $\mathcal{P}, \kappa^{(0)}$, and b are continuous functions on compact domains and are therefore bounded, and K is a bounded set, there exists a constant $C_1 > 0$ such that

$$\sup_{a \in K} \max_{l=1,\dots,n} \sup_{y \in D} \left| \int_D \sum_{j=1}^n \kappa^{(0)}_{lj}(y,z) \mathcal{P}_j(a)(z) \, \mathrm{d}z + b_l(y) \right| < C_1.$$

Since σ is continuous on $\mathbb{R}$, it is bounded on $[-C_1, C_1]$. Using this and the fact that $\kappa^{(1)}$ is continuous on a compact domain, we can find a constant $C_2 > 0$ such that

$$\sup_{a \in K} \max_{k=1,\dots,n} \sup_{x \in D'} |S(\mathcal{P}(a))_k(x)| < C_2.$$

We can find a neural network $\mathcal{Q} : \mathbb{R}^n \to \mathbb{R}$ such that

$$\sup_{x \in [-C_2, C_2]^n} |\mathcal{Q}(x) - \tilde{\mathcal{Q}}(x)| < \epsilon$$

where $\tilde{\mathcal{Q}}(x) = x_1 + \dots x_n$ for any $x \in \mathbb{R}^n$. We can now define our neural operator approximation $\mathcal{G}_\theta : K \to C(D'; \mathbb{R})$ by

$$\mathcal{G}_\theta(a) = \mathcal{Q}(S(\mathcal{P}(a))), \qquad \forall a \in K$$

where θ denotes the concatenation of the parameters of all involved neural networks. It is not hard to see that $\mathcal{G}_\theta$ has the form (18) and that there exists a constant $C_3 > 0$ such hat

$$\sup_{a \in K} \|\mathcal{G}_\theta(a)\|_{C(D;\mathbb{R})} < C_3.$$

Notice that to show boundedness, we only used the fact that K is bounded. Hence to obtain boundedness on B, we simply extend the domain of approximation of the neural networks $\mathcal{P}$, $\mathcal{Q}$, using, instead of the constant $M > 0$, the constant $M' \geq M$ defined to be a number such that

$$\sup_{a \in B} \|a\|_{C(D;\mathbb{R})} = \sup_{a \in B} \sup_{x \in D} |a(x)| < M'.$$

We now complete the proof by extensive use of the triangle inequality. Let $a \in K$. We begin by noting that

$$\begin{aligned}
\sup_{x \in D'} |\mathcal{G}^\dagger(a)(x) - \mathcal{G}_\theta(a)(x)| &\leq \sup_{x \in D'} \Big(|G^\dagger(a)(x) - \tilde{\mathcal{Q}}(\tilde{S}(\tilde{\mathcal{P}}(a)))(x)| + |\tilde{\mathcal{Q}}(\tilde{S}(\tilde{\mathcal{P}}(a)))(x) - \mathcal{G}_\theta(a)(x)| \Big) \\
&\leq \epsilon + \sup_{x \in D'} |\tilde{\mathcal{Q}}(\tilde{S}(\tilde{\mathcal{P}}(a)))(x) - \mathcal{Q}(S(\mathcal{P}(a)))(x)|.
\end{aligned}$$

Suppose now that σ is α-Hölder for some $\alpha > 0$. We pick

$$\beta = \begin{cases} 1, & \alpha \geq \frac{1}{2} \\ \frac{1}{2\alpha}, & 0 < \alpha < \frac{1}{2} \end{cases}$$

recalling that β controls how well the off-diagonal entries of the neural network $\kappa^{(0)}$ approximate the zero function. This choice ensures that, since $n \geq 1$, we have

$$\frac{1}{n^\beta} \leq \frac{1}{n}, \qquad \frac{1}{n^{2\alpha\beta}} \leq \frac{1}{n}.$$

Since $\tilde{\mathcal{Q}}$ is linear hence 1-Hölder, and the only non-linearity in $\tilde{S}$ is due to σ, we have that $\tilde{\mathcal{Q}} \circ \tilde{S}$ is α-Hölder and therefore there is a constant $C_4 > 0$ such that

$$\begin{aligned}
\sup_{x \in D'} |\tilde{\mathcal{Q}}(\tilde{S}(\tilde{\mathcal{P}}(a)))(x) - \mathcal{Q}(S(\mathcal{P}(a)))(x)| &\leq C_4 \sup_{x \in D} \|\tilde{\mathcal{P}}(a)(x) - \mathcal{P}(a)(x)\|_{\mathbb{R}^n}^\alpha \\
&\quad + \sup_{x \in D'} |\tilde{\mathcal{Q}}(\tilde{S}(\mathcal{P}(a)))(x) - \mathcal{Q}(S(\mathcal{P}(a)))(x)| \\
&\leq C_4 \sup_{x \in [-M, M]} \|\tilde{\mathcal{P}}(x) - \mathcal{P}(x)\|_{\mathbb{R}^n}^\alpha \\
&\quad + \sup_{x \in D'} |\tilde{\mathcal{Q}}(\tilde{S}(\mathcal{P}(a)))(x) - \mathcal{Q}(S(\mathcal{P}(a)))(x)| \\
&\leq C_4 \epsilon^\alpha + \sup_{x \in D'} |\tilde{\mathcal{Q}}(\tilde{S}(\mathcal{P}(a)))(x) - \mathcal{Q}(S(\mathcal{P}(a)))(x)|.
\end{aligned}$$

Similarly,

$$\begin{aligned}
\sup_{x\in D'} |\tilde{\mathcal{Q}}(\tilde{S}(\mathcal{P}(a)))(x) - \mathcal{Q}(S(\mathcal{P}(a)))(x)| &\leq \|\tilde{S}(\mathcal{P}(a)) - S(\mathcal{P}(a))\|_{C(D';\mathbb{R}^n)} \\
&\quad + \sup_{x\in D'} |\tilde{\mathcal{Q}}(S(\mathcal{P}(a)))(x) - \mathcal{Q}(S(\mathcal{P}(a)))(x)| \\
&\leq \|\tilde{S}(\mathcal{P}(a)) - S(\mathcal{P}(a))\|_{C(D';\mathbb{R}^n)} \\
&\quad + \sup_{x\in[-C_2,C_2]^n} |\tilde{\mathcal{Q}}(x) - \mathcal{Q}(x)| \\
&\leq \epsilon + \|\tilde{S}(\mathcal{P}(a)) - S(\mathcal{P}(a))\|_{C(D';\mathbb{R}^n)}.
\end{aligned}$$

Note that since $\mathcal{P}$ is continuous, $\mathcal{P}(K)$ is compact, therefore there is a number $M' > 0$ such that

$$\sup_{a\in K} \|\mathcal{P}(a)\|_{C(D;\mathbb{R}^n)} < M'.$$

Let $f = \mathcal{P}(a)$ then for any $k \in \{1, \dots, n\}$ and $x \in D'$, we have

$$\begin{aligned}
|\sum_{l=1}^n \int_D \kappa_{kl}^{(1)}(x,y)\sigma(A_l)\,\mathrm{d}y - \int_D \tilde{\kappa}_k^{(1)}(x,y)\sigma(\tilde{A})\,\mathrm{d}y| &\leq |\sum_{l\neq k} \int_D \kappa_{kl}^{(1)}(x,y)\sigma(A_l)\,\mathrm{d}y| \\
&\quad + |\int_D \kappa_{kk}^{(1)}(x,y)\sigma(A_k)\,\mathrm{d}y - \int_D \tilde{\kappa}_k^{(1)}(x,y)\sigma(\tilde{A})\,\mathrm{d}y|
\end{aligned}$$

where

$$\begin{aligned}
A_l(y) &= \int_D \sum_{j=1}^n \kappa_{lj}^{(0)}(y,z) f_j(z)\,\mathrm{d}z + b_l(y) \\
\tilde{A}(y) &= \int_D \tilde{\kappa}_k^{(0)}(y,z) f_k(z)\,\mathrm{d}z + \tilde{b}_k(y).
\end{aligned}$$

By Hölder's inequality,

$$\begin{aligned}
\sup_{x\in D'} |\sum_{l\neq k} \int_D \kappa_{kl}^{(1)}(x,y)\sigma(A_l)\,\mathrm{d}y| &\leq \sup_{x\in D'} \sum_{l\neq k} \|\kappa_{kl}^{(1)}(x,\cdot)\|_{L^2(D;\mathbb{R})} \|\sigma(A_l)\|_{L^2(D;\mathbb{R})} \\
&\leq \epsilon |D|^{1/2} C_5
\end{aligned}$$

for some constant $C_5 > 0$ found by using boundedness of $\mathcal{P}(K)$ and continuity of σ. To see this, note that

$$\begin{aligned}
\sup_{x\in D'} \|\kappa_{kl}^{(1)}(x,\cdot)\|_{L^2(D;\mathbb{R})} &\leq \sup_{x\in D'} \left(|D| \sup_{y\in D} |\kappa_{kl}^{(1)}(x,y)|^2 \right)^{1/2} \\
&\leq \left(|D| \frac{\epsilon^2}{n^2} \right)^{1/2} \\
&= |D|^{1/2} \frac{\epsilon}{n}.
\end{aligned}$$

Furthermore,

$$\sup_{y\in D}|A_l(y)| \leq \sup_{y\in D}\left(\|\kappa_{ll}^{(0)}(y,\cdot)\|_{L^2}\|f_l\|_{L^2} + \sum_{j\neq l}\|\kappa_{lj}^{(0)}(y,\cdot)\|_{L^2}\|f_j\|_{L^2} + |b_l(y)|\right)$$

and

$$\sup_{y\in D}\sum_{j\neq l}\|\kappa_{lj}^{(0)}(y,\cdot)\|_{L^2}\|f_j\|_{L^2} \leq \sum_{j\neq l}|D|M'\frac{\epsilon}{n^\beta} \leq \epsilon|D|M'$$

hence A_l is uniformly bounded for all $a \in K$ and existence of C_5 follows. Using Hölder again, we have

$$\begin{aligned}
|\int_D \kappa_{kk}^{(1)}(x,y)\sigma(A_k)\,\mathrm{d}y - \int_D \tilde{\kappa}_k^{(1)}(x,y)\sigma(\tilde{A})\,\mathrm{d}y| &\leq |\int_D \kappa_{kk}^{(1)}(x,y)(\sigma(A_k)-\sigma(\tilde{A}))\,\mathrm{d}y \\
&\quad + |\int_D (\kappa_{kk}^{(1)}(x,y)-\tilde{\kappa}_k^{(1)}(x,y))\sigma(\tilde{A})\,\mathrm{d}y| \\
&\leq \|\kappa_{kk}^{(1)}(x,\cdot)\|_{L^2}\|\sigma(A_k)-\sigma(\tilde{A})\|_{L^2} \\
&\quad + \|\kappa_{kk}^{(1)}(x,\cdot)-\tilde{\kappa}_k^{(1)}\|_{L^2}\|\sigma(\tilde{A})\|_{L^2}.
\end{aligned}$$

Clear $\tilde{A}$ is uniformly bounded for all $a \in K$ hence there exists a constant $C_6 > 0$ such that

$$\sup_{x\in D'}\|\kappa_{kk}^{(1)}(x,\cdot)-\tilde{\kappa}_k^{(1)}\|_{L^2}\|\sigma(\tilde{A})\|_{L^2} \leq \epsilon|D|^{1/2}C_6.$$

Using α-Hölder continuity of σ and the generalized triangle inequality, we find a constant $C_7 > 0$ such that

$$\begin{aligned}
\|\sigma(A_k)-\sigma(\tilde{A})\|_{L^2}^2 &\leq C_7\Big(\int_D \|\kappa_{kk}^{(0)}(y,\cdot)-\tilde{\kappa}_k^{(0)}(y,\cdot)\|_{L^2}^{2\alpha}\|f_k\|_{L^2}^{2\alpha}\,\mathrm{d}y + \sum_{j\neq k}\int_D \|\kappa_{kj}^{(0)}\|_{L^2}^{2\alpha}\|f_j\|_{L^2}^{2\alpha}\,\mathrm{d}y \\
&\quad + \int_D (b_k(y)-\tilde{b}_k(y))^{2\alpha}\,\mathrm{d}y\Big) \\
&\leq C_7\left(|D|^{2\alpha}M'^{2\alpha}\epsilon^{2\alpha} + \sum_{j\neq k}|D|^{2\alpha}M'^{2\alpha}\frac{\epsilon^{2\alpha}}{n^{2\alpha\beta}} + |D|\epsilon^{2\alpha}\right) \\
&\leq \epsilon^{2\alpha}C_7\left(|D|^{2\alpha}M'^{2\alpha} + |D|^{2\alpha}M'^{2\alpha} + |D|\right).
\end{aligned}$$

Since all of our estimates are uniform over $k = 1\ldots,n$, we conclude that there exists a constant $C_8 > 0$ such that

$$\|\tilde{S}(\mathcal{P}(a)) - S(\mathcal{P}(a))\|_{C(D';\mathbb{R}^n)} \leq (\epsilon+\epsilon^\alpha)C_8.$$

Since ϵ is arbitrary, the proof is complete. ■