Neural Operators with Localized Integral and Differential Kernels

Miguel Liu-Schiaffini [*1] **Julius Berner** [*1] **Boris Bonev** [*2] **Thorsten Kurth** [2] **Kamyar Azizzadenesheli** [2] **Anima Anandkumar** [1]

Abstract

Neural operators learn mappings between function spaces, which is practical for learning solution operators of PDEs and other scientific modeling applications. Among them, the Fourier neural operator (FNO) is a popular architecture that performs global convolutions in the Fourier space. However, such global operations are often prone to over-smoothing and may fail to capture local details. In contrast, convolutional neural networks (CNN) can capture local features but are limited to training and inference at a single resolution. In this work, we present a principled approach to operator learning that can capture local features under two frameworks by learning differential operators and integral operators with locally supported kernels. Specifically, inspired by stencil methods, we prove that we obtain differential operators under an appropriate scaling of the kernel values of CNNs. To obtain local integral operators, we utilize suitable basis representations for the kernels based on discrete-continuous convolutions. Both these approaches preserve the properties of operator learning and, hence, the ability to predict at any resolution. Adding our layers to FNOs significantly improves their performance, reducing the relative L^2-error by 34-72% in our experiments, which include a turbulent 2D Navier-Stokes and the spherical shallow water equations.

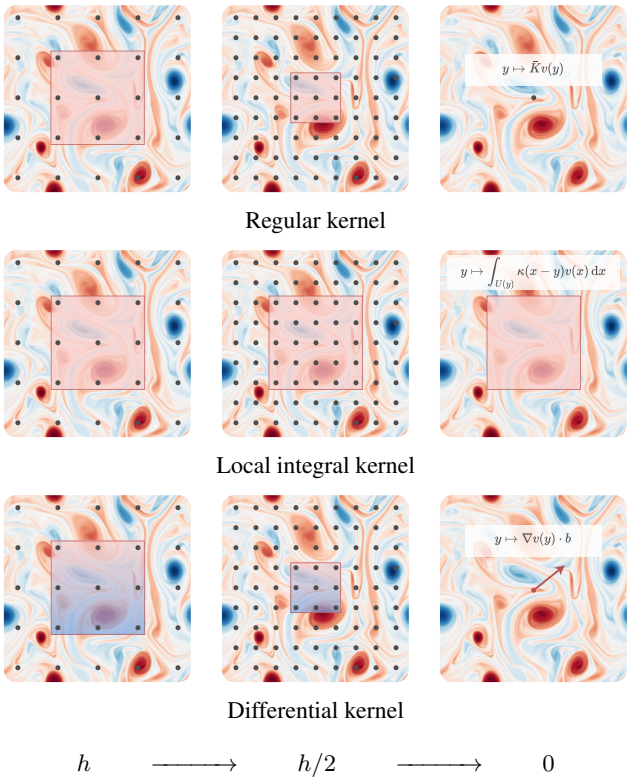


Figure 1: Visualization of different limits of a convolution with a discretized function v as the grid width h is refined, i.e., $h \to 0$. **(Top)** A regular convolution is collapsing to a pointwise linear operator. **(Middle)** Instead, we can use a kernel that can be evaluated at arbitrary resolutions and keep the receptive field unchanged, to converge to a local integral operator, see Section 3.4. **(Bottom)** Alternatively, we can let it collapse while constraining the kernel appropriately, converging to a differential operator, see Section 3.2.

1. Introduction

Deep learning holds the promise to greatly accelerate advances in computational science and engineering, which often require numerical solutions of partial differential equations (PDEs) (Azizzadenesheli et al., 2024; Zhang et al., 2023; Cuomo et al., 2022). Recent advances in deep learning have enabled applications such as weather forecasting (Pathak et al., 2022; Bonev et al., 2023; Lam et al., 2023), seismology (Sun et al., 2023; Shi et al., 2023), reservoir engineering for carbon capture (Wen et al., 2022; 2023), and many other applications with orders of magnitude speedup over traditional methods.

Many of the above results are achieved by neural operators, which learn mappings between function spaces, enabling

[*]Equal contribution [1]Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena CA 91125 [2]NVIDIA, Santa Clara, CA 95051. Correspondence to: Miguel Liu-Schiaffini <mliuschi@caltech.edu>, Julius Berner <jberner@caltech.edu>, Boris Bonev <bbonev@nvidia.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

operator learning for function-valued data (Li et al., 2021; Azizzadenesheli et al., 2024; Raonić et al., 2023). In particular, they are agnostic to the discretization of the input and output functions—a vital property in the context of PDEs where data is often provided at varying resolutions and high-resolution data is costly to generate (Kovachki et al., 2021). In contrast, standard neural networks such as convolutional neural networks (CNN) (Ronneberger et al., 2015; Gupta & Brandstetter, 2022) require the functions to be discretized at a fixed resolution on a regular grid, which is limiting.

In the past few years, various architectures of neural operators have been developed. Among them, the Fourier neural operator (FNO) (Li et al., 2020a), which performs global convolutions in Fourier space, has gained popularity and shown good performance in a number of applications. However, such global operations are often prone to over-smoothing and may fail to capture local details. Several other architectures instead learn global operators in physical space (e.g., (Li et al., 2022b)), but they likewise lack the inductive bias of local operations.

There are many applications that require a local neural operator. For instance, solution operators of several relevant PDEs are of local nature. Examples include hyperbolic PDEs, which have real-valued characteristic curves (LeVeque, 1992). As a result, a solution at a given point will only depend on the initial condition within a neighborhood of that point. As such, their solution operators only have a local receptive field and can, therefore, be efficiently learned by locally supported kernels.

Further examples of local operators are differential operators, which can be expressed in terms of pointwise multiplication with the frequency in the spectral domain. Consequently, they introduce large errors when approximated by a finite number of parameters in Fourier space. In this context, we note that the emulation of classical numerical methods to solve partial differential equations, such as finite-difference methods, relies on the usage of local stencils for differentiation (Thomas, 2013). This calls for the presence of local and differential operators in neural operator architectures.

Naturally, every local operation can also be represented by a global operation. However, this is typically vastly parameter-inefficient and does not provide a good inductive bias for learning local operations. In the context of neural operators, spectral variants of neural operators, such as the FNO (Li et al., 2020a) and Spherical FNO (SFNO) (Bonev et al., 2023), are theoretically able to approximate local convolutions. However, representing local kernels requires the approximation of a global signal in the spectral domain, in turn demanding a large number of parameters (due to the uncertainty principle (Cohen-Tannoudji et al., 1977)).

A few prior works have explored the usage of local operations in the context of neural operators. Li et al. (2020b) introduce graph neural operators (GNOs), which parameterize local integral kernels with a neural network. However, evaluating this network on all combinations of points to compute the integrals can make GNOs computationally expensive and slower than hardware-optimized convolutional kernels. Alternatively, Ye et al. (2022; 2023) propose local neural operators by combining FNOs with convolutional layers and, similarly, Wen et al. (2022) integrate U-Nets and FNOs. Moreover, Raonić et al. (2023) propose convolutional neural operators (CNOs) by leveraging U-Nets that (approximately) respect the bandlimits.

Table 1: Comparison of different architectures for the solution of PDEs. The top half enumerates architectures for planar domains and the bottom half for spherical domains. Our proposed architectures are highlighted in bold and the differential and integral kernels are detailed in Section 3.

Architecture	Efficient	Receptive field	no input downsampling
GNO	✗	local/global	✓
FNO	✓	global	✓
CNO / U-Net	✓	local	✗
FNO + integral	✓	local/global	✓
FNO + differential	✓	local/global	✓
SFNO	✓	global	✓
SFNO + integral	✓	local/global	✓

However, since all of the above approaches rely on standard convolutional layers on equidistant grids, they have the following shortcomings. First, such approaches do not allow for a natural extension to unstructured grids or other geometries, which are ubiquitous in PDE problems (Li et al., 2023). Moreover, they can only be applied to higher resolutions by downsampling of the (intermediate) inputs to the training resolution. Important high-frequency content can be lost through downsampling, which is particularly problematic for multi-scale data in the context of PDEs. In contrast, we develop convolutional layers that can be applied at any resolution without downsampling. We achieve this by appropriately scaling the receptive field or the values of the kernel (see Figure 1).

To summarize, current neural operator architectures suffer from at least one of the following limitations (see Table 1): (1) they cannot succinctly represent operations with a local receptive field, e.g., FNO, or (2) they cannot be applied to different resolutions without relying on explicit up-/downsampling which may degrade performance, e.g., CNO, or (3) they cannot be scaled to obtain sufficient expressivity, since they incur prohibitively high computationally costs, such as, for instance, GNOs.

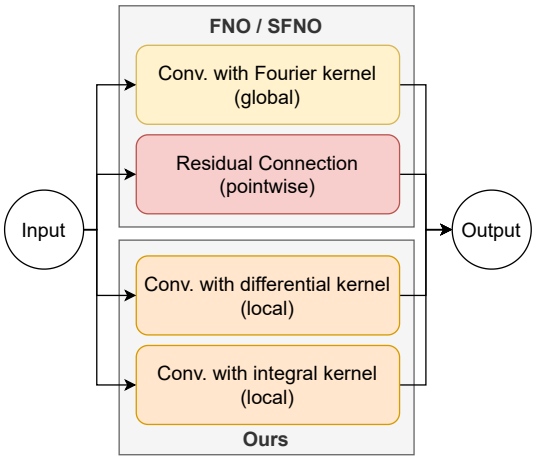


Figure 2: A single layer of our local neural operator. We add (up to) two local operations using the convolutions with differential kernel (Section 3.2) and local integral kernel (Section 3.4).

Our approach: In this work, we develop computationally efficient and principled approaches to include operations in neural operators that capture local receptive fields while retaining the ability to approximate operators and, hence, extend to multiple resolutions. We consider two kinds of localized operators: differential operators and integral kernel operators with a locally supported kernel (see Figure 1).

For the first, we draw inspiration from stencils of finite-difference methods. We derive conditions to modify convolutional layers such that they converge to a unique differential operator when the discretization is refined. For the second case of local integral operators, we adapt discrete-continuous (DISCO) convolutions (Ocampo et al., 2022) to provide an efficient, discretization-agnostic framework that can be applied to general meshes on both planar and spherical geometries. To our knowledge, we are the first to connect the DISCO framework to operator learning.

Finally, we devise efficient implementations of both layers and show that the inductive bias of local operations (see Figure 2) can significantly improve the performance of FNOs and SFNOs on three different benchmarks. In particular, we learn a differential operator in a Darcy flow setting for motivating the need for neural operators with local inductive biases, and we improve over FNO by 87%. Additionally, we improve over the baseline (S)FNO by 34% on turbulent 2D Navier-Stokes equations, by 72% on the shallow water equations on the sphere, and by 63% on the 2D diffusion-reaction equation. We also apply our method on modeling a flow past a cylinder, discretized on an irregular grid. We improve upon the best baseline by 42%.

Outline: The remainder of the paper is organized as follows: Section 2 outlines connections to other architectures and ideas within the neural operator literature. Section 3 introduces the main ideas of the paper as well as the macro-architecture of our proposed neural operator. Section 4 discusses numerical experiments and results, and Section 5 summarizes the findings of our work.

2. Related work and connections to other frameworks

We strive to formulate local operations that are consistent with the neural operator paradigm, which stipulates that operations within the model are independent of the discretization of input functions (Kovachki et al., 2021). In principle, this allows the evaluation of the neural operator on arbitrary meshes[1]. Several discretization-independent local operations have been introduced in the deep learning literature. We outline the connections between these works in the following.

GNOs and Hypernetworks: In general, local operations can be provided by graph neural operators (GNOs) (Li et al., 2020b). However, in their general form, GNOs are typically slow, hard-to-train, and not equivariant w.r.t. the symmetry group of the underlying domain (Li et al., 2020c). Taking a convolutional kernel, GNOs subsume a series of works on hypernetwork approaches for convolutional layers (Wang et al., 2018; Shocher et al., 2020).

DISCO Convolutions: We show that DISCO convolutions (Ocampo et al., 2022) represent special cases of GNOs with a convolutional kernel, which enables generalization to different geometries and an efficient implementation, since the kernel can be pre-computed (see Section 3). In particular, DISCO convolutions are typically implemented as learnable linear combinations of fixed basis functions instead of neural networks. Also, they define maps that are equivariant up to the error resulting from the quadrature rule.

Scale-Equivariant CNNs: In the area of computer vision, there has been a series of adaptations of CNNs to be (locally) scale-equivariant by using filter dilation, filter rescalings in the discrete domain (Rahman & Yeh, 2023; Sosnovik et al., 2021; Worrall & Welling, 2019) or the continuous domain (Xu et al., 2014; Sosnovik et al., 2019; Ghosh & Gupta, 2019), or input rescalings (Marcos et al., 2018). Moreover, filter rescaling has also been explored for more general group-convolutions (Bekkers, 2019).

Neural operators and convolutional layers: While neural operators have been developed independently of the above approaches, one can leverage similar ideas. In partic-

[1] As long as they are suitable for the evaluation of the numerical operations of the neural operator, such as the Discrete Fourier Transform used in the Fourier neural operator (see Section 3).

ular, one can rescale (i.e., up- and downsample) input and output functions of a convolutional layer or the kernel itself to obtain neural operators. We note that these approaches are, in principle, also applicable to a trained network (without the need for retraining).

In combination with FNOs, interpolation of the input and output functions has been explored by Wen et al. (2022). Also, Ye et al. (2023; 2022) emphasized the need for local convolutions in FNOs, however, they do not discuss the application to different discretizations. Combined with correct treatment of the bandlimit of the functions (based on Karras et al. (2021)), the convolutional neural operator (Raonić et al., 2023) also uses up- and downsampling of the input and output functions to apply a U-Net architecture.

We note that for bandlimited functions sampled above the Nyquist frequency, the discrete convolutions have a unique correspondence to convolutions in function space, representing a special case of the DISCO framework. However, downsampling the input function introduces errors in cases where the function is not bandlimited.

3. Local Layers

In this section, we present two conceptually different types of local layers for neural operators. First, we present efficient integral transforms with local kernels. Then, we focus on the approximation of differential operators. We consider an input function

$$v\colon \mathbb{R}^d \supset D \to \mathbb{R}^n$$

that is discretized on meshes $D^h \subset D$ with width h on a domain $D \subset \mathbb{R}^d$. For notational convenience, we present most ideas for the case $d = 1$, but it is straightforward to extend them to higher-dimensional domains.

3.1. Motivation: Convolutional Layer

We take inspiration from convolutional layers since they represent the prototypical version of an efficient, local operation in neural networks. However, we will see that they are not consistent in function spaces; in particular, they converge to a pointwise linear operator when we refine the discretization of the input function v.

Let us start by recalling the definition of a convolutional[2] layer, specifically a stride-1 convolution with n input channels, a single[3] output channel, and kernel $K = (K_i)_{i=1}^S \subset \mathbb{R}^n$ of (odd) size S. Assuming a regular grid, i.e., $D^h = \{x_j\}_{j=1}^m \subset \mathbb{R}$ with $x_{j+1} - x_j = h$, we can define the output of the convolutional layer at $y \in D^h$ as

$$\begin{aligned}\mathrm{Conv}_K[v](y) &= (K \star \{v(x_j)\}_{j=1}^m)(y) \\ &= \sum_{i=1}^S K_i \cdot v(z_i + y), \end{aligned} \tag{1}$$

with

$$z_i = h\left(i - 1 - \tfrac{S-1}{2}\right), \tag{2}$$

where we use zero-padding, i.e., $v(x) = 0$ for $x \notin D$.

If we now take the same kernel K for finer discretizations, i.e., $h \to 0$, we see from (2) that $z_i \to 0$ and therefore,

$$\lim_{h\to 0} \mathrm{Conv}_K[v](y) = \bar{K} \cdot v(y) \quad \text{with} \quad \bar{K} = \sum_{i=1}^S K_i,$$

given that the function v is continuous at y. In other words, the receptive field with respect to the underlying domain D is shrinking to a point, and the convolutional layer is converging to a *pointwise linear operator*. One way to circumvent this issue would be to downsample the function v appropriately to a pre-defined grid. This is done for previous approaches mentioned in Section 2, at the cost of losing high-frequency information in the input.

In the following, we will present two ways of working on different input resolutions, while not collapsing to a pointwise operator, see also Figure 1. First, we show that rescaling (1) by the reciprocal resolution $\frac{1}{h}$ and constraining the kernel K leads to *differential operators*. Then, we define the kernel K as the evaluation of a function over a fixed input domain, leading to *integral operators*.

3.2. Differential Layer

In this section, we construct a layer that converges to a differential operator when the width h of the discretization D^h tends to zero. To prevent the operator from collapsing to a pointwise operation, we constrain and rescale the values of the kernel (according to the discretization width). Note that without rescaling, we would again recover a pointwise operator; however, due to the rescaling by a factor of $\frac{1}{h}$, the limit might not exist without constraining the values of the kernel. The next proposition shows that it is sufficient to subtract the average kernel value. See Appendix A.1 for a more general statement and a corresponding proof.

Proposition 3.1 (First-order differential layer)**.** *Let $D_h \subset \mathbb{R}^d$ be a regular grid of width h and let $v \in C^1(D, \mathbb{R}^n)$. Then, for every kernel $(K_i)_{i=1}^S \subset \mathbb{R}^n$, there exists $(b_j)_{j=1}^n \subset \mathbb{R}^d$ such that*

$$\lim_{h\to 0} \frac{1}{h} \mathrm{Conv}_{K-\bar{K}}[v](y) = \sum_{j=1}^n \nabla v_j(y) \cdot b_j$$

for every $y \in D_h$, where $\bar{K} = \sum_{i=1}^S K_i$.

[2] In line with deep learning frameworks, we consider convolution with the reflected filter, also known as *cross-correlation*.

[3] This is for notational convenience; the extension to multiple output channels is straightforward.

Proposition 3.1 shows that we can learn different directional derivatives using an appropriate adaptation of standard convolutional layers as in Section 3.1. Specifically, we center the kernel K by subtracting its mean $\bar{K}$ and scale the result by the reciprocal resolution $\frac{1}{h}$. In Appendix A.2, we empirically evaluate the convergence in resolution to a unique differnetial operator on a simple example.

Remark 3.2 (Higher-order differential operator). Similarly, we could approximate k-th order differential operator with further constraints on the elements of K and a scaling factor of $\frac{1}{h^k}$. However, we do not implement them in practice since we can also approximate higher-order derivatives by composing first-order differential layers.

3.3. Integral Kernel Layers

Instead of rescaling the kernel, we can also adapt the size of the kernel such that the receptive field stays the same, i.e., is independent of the resolution h. In this section, we begin at the general graph neural operator and demonstrate how (considering the desirable properties of translation equivariance and efficiency) we arrive at local convolutional layers with adaptive kernel sizes using a fixed number of parameters.

Graph neural operator (GNO): One of the most general instantiations of a neural operator is arguably the *graph neural operator* (GNO)

$$\mathrm{GNO}_k[v](y) = \int_{U(y)} k(x,y)\cdot v(x)\,\mathrm{d}x \tag{3}$$

$$\approx \sum_{x\in D^h\cap U(y)} k(x,y)\cdot v(x)\,q_x, \tag{4}$$

where k is a kernel (typically parametrized by a neural network) and $q_x \in \mathbb{R}$ are suitable quadrature weights. While the GNO can represent local integral operators by picking a suitably small neighborhood $U(y) \subset D$, the evaluation of the kernel and aggregation in each neighborhood $U(y)$ is slow and memory-intensive for general kernels $k\colon D \times D \to \mathbb{R}^n$. Moreover, for an arbitrary kernel, GNO is not equivariant w.r.t. the symmetry group of the underlying domain. In particular, we lose the translation equivariance of convolutional layers for planar domains.

Fourier Neural Operator (FNO): To retain translation equivariance, we can consider kernels of the form

$$k(x,y) = \kappa(x-y)$$

and $U(y) = y + \mathrm{supp}(\kappa)$. Then, we can rewrite the GNO as a convolution, i.e.,

$$\mathrm{GNO}_\kappa[v] = \kappa \star v. \tag{5}$$

If we are dealing with periodic functions on the torus D, we can leverage the convolution theorem to compute (5), i.e.,

$$\mathcal{F}[\kappa \star v] = \overline{\mathcal{F}[\kappa]}\cdot\mathcal{F}[v], \tag{6}$$

where $\mathcal{F}$ maps functions to their Fourier series coefficients. The *Fourier Neural Operator* (FNO) now directly parametrizes $\overline{\mathcal{F}[\kappa]}$ and approximates $\mathcal{F}$ using the *fast Fourier transform* given that D^h is an equidistant grid. While this leads to an efficient version of (4), it assumes that $\overline{\mathcal{F}[\kappa]}$ has only finitely many nonzero Fourier modes—or, equivalently, that the kernel κ has full support, making (5) a *global* convolution.

Localized convolutions: To construct a locally supported kernel κ, we can directly discretize the convolution in (5), i.e.,

$$\mathrm{GNO}_\kappa[v](y) \approx \sum_{x\in D^h} \kappa(x-y)\cdot v(x)\,q_x. \tag{7}$$

Note that the sum can be taken only over the $x \in D^h$ with

$$x - y \in \mathrm{supp}(\kappa).$$

We remark that for an equidistant grid and constant[4] quadrature weights, $q = q_x$, evaluating $\mathrm{GNO}_\kappa[v]$ at $y \in D^h$ corresponds to a standard convolution as in (1) with kernel

$$K_i = q\kappa(z_i), \quad i = 1,\dots,S,$$

where z_i is defined as in (2) and S is sufficiently large such that $\mathrm{supp}(\kappa) \subset [z_1, z_S]$. See Appendix B.2 for details. However, the advantage of the formulation in (7) is the fact that we can reuse the same kernel κ across different resolutions (with the same receptive field $\mathrm{supp}(\kappa)$).

The remaining question is centered around the parametrization of the kernel. One could draw inspiration from works on *hypernetworks* in computer vision and parameterize it using a neural network (Shocher et al., 2020). However, this is significantly more costly than a standard convolution. Another idea is to interpolate a fixed-sized kernel, e.g., using sinc or bi-linear interpolation. A more general version of the latter, based on a learnable linear combination of hat functions, is also known as *discrete-continuous* (DISCO) convolution and has shown to be effective in different applications (Ocampo et al., 2022). We note that this formulation is closest to the original convolutional layer typically used in computer vision. However, in the next section, we see that it also allows us to use unstructured meshes and formulate the operation on more general domains $D \subset \mathbb{R}^d$.

3.4. General discrete-continuous convolutions

The local support of the kernel in equation (7) allows us to efficiently evaluate local convolutions in subdomains

[4] This is, e.g., the case for the trapezoidal rule on a torus.

of $\mathbb{R}^d$ using sparse matrix-vector products. However, the operation $x - y$, which shifts the convolution kernel, is not well-defined on manifolds such as the sphere. To generalize the previous discussion to a Lie group G, we first replace the shift operator with the group action g and obtain the so-called group convolution

$$\begin{aligned}\text{GroupConv}_\kappa[v](g) &= (\kappa \star v)(g) \\ &= \int_G \kappa(g^{-1}x) \cdot v(x)\, \mathrm{d}\mu(x), \end{aligned} \quad (8)$$

where $g, x \in G$ and $\mathrm{d}\mu(x)$ is the invariant Haar measure on G. This formulation presents us with the challenge of being non-trivial to discretize. The framework of DISCO convolutions (Ocampo et al., 2022) achieves this by approximating the integral with a quadrature rule while evaluating the group action continuously:

$$\begin{aligned}(k \star v)(g_i) &\approx \sum_{j=1}^{m} \kappa(g_i^{-1}x_j) \cdot v(x_j)\, q_j \\ &= \sum_{j=1}^{m} K_{ij} \cdot v(x_j)\, q_j. \end{aligned} \quad (9)$$

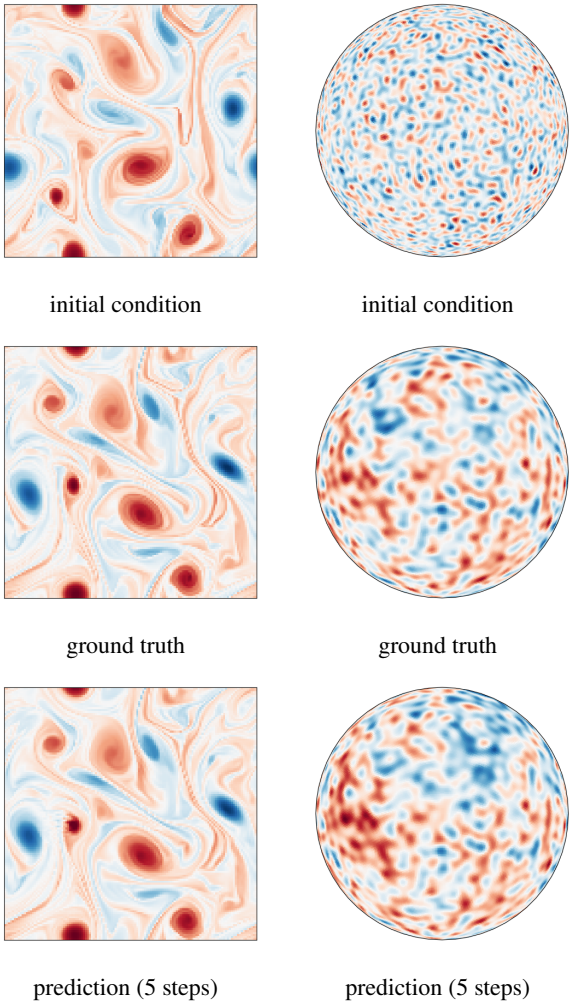
Here $(x_j)_{j=1}^m \subset G$ represent quadrature points with corresponding quadrature weights q_j. For a given set of output positions g_i, we obtain the fully discrete formulation as matrix-vector product with the matrix $K_{ij} = \kappa(g_i^{-1}x_j)$, which is sparse due to the local support of the kernel. To parameterize the kernel, we choose a linear combination of basis functions $\kappa^{(\ell)}$, such that $\kappa = \sum_{\ell=1}^{L} \theta^{(\ell)}\kappa^{(\ell)}$ with trainable parameters $\theta^{(\ell)} \in \mathbb{R}$.

Although we have presented the general idea only for Lie groups G, it is possible to construct the DISCO convolution also on manifolds with a group action acting on them, such as the 2-sphere $\mathbb{S}^2$. For a detailed discussion of DISCO convolutions and a construction in one dimension, we point the reader to Appendix B and Ocampo et al. (2022).

Remark 3.3 (Exact integration and equivariance). We note that DISCO convolutions satisfy equivariance properties for function classes that can be exactly integrated. For instance, on planar domains, these could be polynomials when using Legendre points. For equidistant grids on the torus, exact integration holds for bandlimited functions that are sampled above the Nyquist frequency.

3.5. Local neural operator architecture

To design our neural operator, we want to combine pointwise, local, and global operations. To this end, we take an FNO (or SFNO for spherical problems) as a starting point, which already features global operations in Fourier space and pointwise operations using its residual connections. Then, we augment the operators by incorporating our proposed local convolutions from the previous section as additional branches in the respective layers (see Figure 2). These four branches are summed pointwise within each local neural operator layer. The resulting architecture can be trained end-to-end with any standard operator learning training procedure and loss.

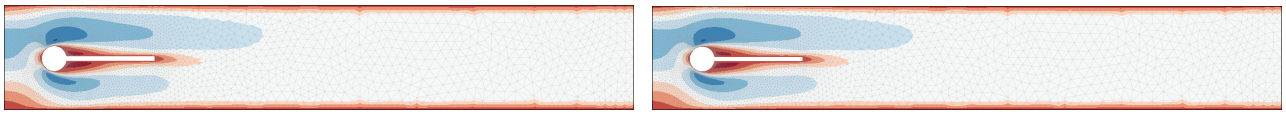


(a) **Navier-Stokes Eqns.** *(b)* **Shallow Water Eqns.**

Figure 3: Initial condition, ground truth, and corresponding autoregressive predictions of our proposed models for the Navier-Stokes problem and the shallow water equations.

4. Experiments

To validate the effectiveness of local operators, we evaluate the architecture on five PDE problems. Figure 3 and Figure 4 show samples on three of these problems. In all five cases, we incorporate our proposed local operators into existing FNO and SFNO architectures to demonstrate that a significant improvement in performance can be achieved by introducing the inductive bias of local convolutions. This



ground truth prediction

Figure 4: Ground truth and prediction of the horizontal velocity for the flow past a cylinder. The data is represented on an unstructured mesh, which is visualized in gray color. Our proposed architecture can be readily applied to data on unstructured meshes such as this one. For this figure, we learn the residual to the previous time step.

section describes the experimental setting for our problems. We provide further experiments, specific implementation details[5], and hyperparameters in Appendix C.

4.1. Darcy flow

We first consider the steady-state, two-dimensional Darcy flow equation

$$-\nabla \cdot (a \nabla u) = f, \quad u|_{\partial D} = 0, \tag{10}$$

on the domain $D = (0,1)^2$. In this problem, we motivate the need to incorporate local operators as inductive biases in neural operator architectures. To this end, we explicitly construct a problem that requires approximating the forcing function f for a given diffusion coefficient a and pressure u in (10). In other words, the task consists of learning the differential operator

$$u \mapsto -\nabla \cdot (a \nabla u). \tag{11}$$

Following the setup of Hasani & Ward (2024), we take

$$a(x) = \begin{bmatrix} x_1^2 & \sin(x_1 x_2) \\ x_1 + x_2 & x_2 \end{bmatrix}$$

for $x \in D$. We then compare the performance of our proposed models: FNO in parallel with differential kernels, FNO with integral kernels, and FNO with both kernels. We also compare with a baseline FNO and the U-Net architecture from Gupta & Brandstetter (2022). Moreover, we perform zero-shot super-resolution on this problem; these results can be found in Appendix C.6.

4.2. Navier-Stokes equations

Next, we consider the two-dimensional Navier-Stokes equation on the torus. In particular, we consider Kolmogorov flows, which can be described by

$$\begin{aligned} \partial_t u + u \cdot \nabla u - \frac{1}{\mathrm{Re}} \Delta u &= -\nabla p + \sin(my)\hat{x} \\ \nabla \cdot u &= 0 \end{aligned} \tag{12}$$

on the spatial domain $D = (0, 2\pi)^2$ equipped with periodic boundary conditions. In the above, u and p denote the velocity and pressure, and Re denotes the Reynolds number of the flow. We want to learn the solution operator mapping initial conditions $u(\cdot, 0) = u_0$ to the time-evolved solution in vorticity form $w(\cdot, \tau)$ at time $\tau \in (0, \infty)$, where the vorticity is given by

$$w = (\nabla \times u)\hat{z} \tag{13}$$

with $\hat{z}$ being a unit vector normal to the plane.

We consider $m = 4$ and $\mathrm{Re} = 5000$, and we emphasize that learning the solution operator for such a high Reynolds number is very challenging due to the turbulent nature and small-scale features of the flow. We conjecture that the baseline FNO is prone to over-smoothing over the finer scales, consequently leading to a degradation of performance.

To validate this conjecture, we compare the performance of the local neural operator with a baseline FNO and U-Net and evaluate the effectiveness of our proposed differential and integral kernels, respectively.

4.3. Diffusion-Reaction equation

Moreover, we consider the 2D Diffusion-Reaction equation from Takamoto et al. (2022) which can, for instance, be used for modeling biological pattern formation. In particular, we want to predict the time-evolution of two non-linearly coupled variables, i.e., the activator u and the inhibitor v, solving the equation

$$\partial_t u = c_u \Delta u + R_u(u, v), \tag{14}$$
$$\partial_t v = c_v \Delta v + R_v(u, v), \tag{15}$$

for the spatial domain $D = (-1, 1)^2$ with no-flow Neumann boundary condition. The reaction functions R_u and R_v are given by the Fitzhugh-Nagumo equation

$$R_u(u, v) = u - u^3 - k - v, \quad R_v(u, v) = u - v \tag{16}$$

and we consider $c_u = 10^{-3}$, $c_v = 5 \cdot 10^{-3}$, and $k = 5 \cdot 10^{-3}$. The task is to learn the operator mapping the state (u, v) at consecutive time-steps to the time-evolved state and we can compare against the baselines from Takamoto et al. (2022).

[5]Code is available in the `neuraloperator` library at github.com/neuraloperator/neuraloperator and the `torch-harmonics` library at github.com/NVIDIA/torch-harmonics.

Table 2: Results for the 2D diffusion-reaction equation from PDEBench (Takamoto et al., 2022). We present their U-Net baseline results and metrics. In particular, we report the RMSE errors for the low, medium, and high frequency ranges (fRMSE low/med./high), and the (absolute) RMSE, the maximum error, RMSE errors at the boundary (bRMSE), as well as RMSE for the conserved values (cRMSE).

Model	# Params	Rel. L^2-error	fRMSE low	fRMSE med.	fRMSE high	RMSE	max. error	bRMSE	cRMSE
U-Net	$7.8 \cdot 10^5$	$8.4 \cdot 10^{-1}$	$1.7 \cdot 10^{-2}$	$8.2 \cdot 10^{-4}$	$5.7 \cdot 10^{-2}$	$6.1 \cdot 10^{-2}$	$1.9 \cdot 10^{-1}$	$7.8 \cdot 10^{-2}$	$3.9 \cdot 10^{-2}$
FNO	$9.3 \cdot 10^5$	$8.3 \cdot 10^{-2}$	$6.2 \cdot 10^{-4}$	$5.6 \cdot 10^{-4}$	$2.4 \cdot 10^{-4}$	$5.2 \cdot 10^{-3}$	$7.3 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$1.2 \cdot 10^{-3}$
FNO + loc. int (ours)	$8.8 \cdot 10^5$	$6.3 \cdot 10^{-2}$	$4.0 \cdot 10^{-4}$	$4.6 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$	$3.6 \cdot 10^{-3}$	$5.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$\mathbf{4.8 \cdot 10^{-4}}$
FNO + diff. (ours)	$8.8 \cdot 10^5$	$3.4 \cdot 10^{-2}$	$4.4 \cdot 10^{-4}$	$1.9 \cdot 10^{-4}$	$\mathbf{6.1 \cdot 10^{-5}}$	$1.9 \cdot 10^{-3}$	$3.5 \cdot 10^{-2}$	$4.4 \cdot 10^{-3}$	$1.1 \cdot 10^{-3}$
FNO + loc. int. + diff. (ours)	$8.9 \cdot 10^5$	$\mathbf{3.1 \cdot 10^{-2}}$	$\mathbf{3.2 \cdot 10^{-4}}$	$\mathbf{1.8 \cdot 10^{-4}}$	$6.2 \cdot 10^{-5}$	$\mathbf{1.7 \cdot 10^{-3}}$	$\mathbf{3.3 \cdot 10^{-2}}$	$\mathbf{4.2 \cdot 10^{-3}}$	$7.4 \cdot 10^{-4}$

4.4. Shallow water equations

To test the approach on the sphere, we consider the shallow water equations on the rotating sphere. They represent a system of hyperbolic partial differential equations used to model a variety of geophysical flow phenomena, such as atmospheric flows, tsunamis, and storm surges. They can be formulated in terms of the evolution of two state variables φ and u (geopotential height and the tangential velocity of the fluid column), governed by the equations

$$\begin{aligned} \partial_t \varphi + \nabla \cdot (\varphi u) &= 0, \\ \partial_t (\varphi u) + \nabla \cdot F &= f, \end{aligned} \tag{17}$$

on the sphere $D = \mathbb{S}^2$ with suitable initial conditions $\varphi(\cdot, 0) = \varphi_0$ and $u(\cdot, 0) = u_0$, a momentum flux tensor

$$F_{ij} = \varphi u_i u_j + \frac{1}{2}\varphi^2, \tag{18}$$

and a source term

$$f = -2\Omega x \times (\varphi u), \tag{19}$$

which models the Coriolis force due to the rotation of the sphere with angular velocity Ω.

As baselines, we use a planar U-Net architecture, a spherical U-Net, where convolutions are performed using local integral kernels on the sphere, and an SFNO architecture. On the sphere, we only consider the impact of the integral kernel on the SFNO architecture, as it allows for a natural extension to the sphere (see Section 3.4).

4.5. Flow past a cylinder

Due to the formulation of the discrete-continuous convolutions, our approach can be readily generalized to unstructured meshes. To demonstrate this, we consider the Navier-Stokes equations in a two-dimensional channel with a cylindrical structure and a membrane attached to it (see Rahman et al., 2024). Figure 4 depicts a sample from the validation dataset alongside predictions from our model.

The dataset considers the flow past the cylinder with Reynolds number $\mathrm{Re} = 2000$. Rahman et al. (2024) provide several baselines for a low-data regime (250 samples),

Table 3: Results for the cylinder flow problem in a low-data regime with 250 training samples. Baseline results are taken from Rahman et al. (2024) and all architecture including ours perform a direct prediction (no residual prediction) of the velocity and pressure at the next time step.

Model	# Parameters	MSE error
GINO	$6 \cdot 10^7$	$2.09 \cdot 10^{-2}$
DeepONet	$6 \cdot 10^6$	$1.39 \cdot 10^{-1}$
GNN	$6 \cdot 10^5$	$5.00 \cdot 10^{-3}$
ViT	$3 \cdot 10^7$	$1.19 \cdot 10^{-2}$
U-net	$3 \cdot 10^7$	$9.34 \cdot 10^{-2}$
FNO + local integral kernel (ours)	$10 \cdot 10^6$	$\mathbf{2.88 \cdot 10^{-3}}$

where the task is to predict the velocity and pressure after a given time step. To deal with the unstructured grid, we adapt the architecture to utilize local integral operators as encoders and decoders to transform the data into a latent representation on an equidistant grid. Then, we can use an FNO with our local integration layers in the latent space.

4.6. Results and discussion

The results of our numerical experiments are reported[6] in Tables 2 to 4. We observe significant performance gains over the baselines in all five problem settings. In particular, we notice that the best performance in the reported relative L^2-errors is achieved with the inclusion of both global and local operations, despite an overall reduction in parameter count with respect to their corresponding FNO/SFNO baselines. This is particularly pronounced for the Navier-Stokes and shallow water equations, where errors can accumulate in autoregressive roll-outs. We attribute this to our models' inductive bias, allowing it to better capture the fine-grained scales and thus achieve better performance. In Table 2, we also show that our local layers achieve lower errors at high frequencies for the diffusion-reaction equation, indicating that local, high-frequency features are correctly captured.

The Navier-Stokes problem demonstrates the efficacy of

[6] In all settings, hyperparameters are chosen to result in macro-architectures with similar parameter counts for the purpose of comparability. The experimental setup, including the choice of hyperparameters, is outlined in Appendix C in the Appendix.

Table 4: Experimental results for Darcy flow, Navier-Stokes, and the spherical shallow water problems. For all three problems, the test error is reported in terms of the relative L^2-error after a single step. For the time-dependent Navier-Stokes and shallow water equations, we also predict the error after 5 autoregressive steps.

Model	Parameters				Relative L^2-Error	
	# Layers	# Modes	Embedding	# Parameters	1 step	5 steps
Darcy Flow						
U-Net	17	-	18	$2.850 \cdot 10^6$	$1.380 \cdot 10^{-2}$	-
FNO	4	20	41	$2.715 \cdot 10^6$	$5.867 \cdot 10^{-2}$	-
FNO + diff. kernel (ours)	4	12	65	$2.638 \cdot 10^6$	$\mathbf{7.357 \cdot 10^{-3}}$	-
FNO + local integral kernel (ours)	4	20	40	$2.617 \cdot 10^6$	$6.034 \cdot 10^{-2}$	-
FNO + local integral + diff. kernel (ours)	4	12	64	$2.639 \cdot 10^6$	$9.032 \cdot 10^{-3}$	-
Navier-Stokes Equations						
U-Net	17	-	56	$2.758 \cdot 10^7$	$1.674 \cdot 10^{-1}$	$5.115 \cdot 10^{-1}$
FNO	4	40	65	$2.711 \cdot 10^7$	$1.381 \cdot 10^{-1}$	$2.360 \cdot 10^{-1}$
FNO + diff. kernel (ours)	4	40	65	$2.726 \cdot 10^7$	$1.073 \cdot 10^{-1}$	$2.129 \cdot 10^{-1}$
FNO + local integral kernel (ours)	4	20	129	$2.716 \cdot 10^7$	$1.110 \cdot 10^{-1}$	$2.183 \cdot 10^{-1}$
FNO + local integral + diff. kernel (ours)	4	20	127	$2.691 \cdot 10^7$	$\mathbf{9.022 \cdot 10^{-2}}$	$\mathbf{1.956 \cdot 10^{-1}}$
Spherical Shallow Water Equations						
U-Net	17	-	32	$2.898 \cdot 10^6$	$1.341 \cdot 10^{-3}$	$1.226 \cdot 10^{-2}$
Spherical U-Net (with local integral kernel)	17	-	32	$1.639 \cdot 10^6$	$6.160 \cdot 10^{-4}$	$3.265 \cdot 10^{-3}$
SFNO	4	128	32	$1.066 \cdot 10^6$	$9.220 \cdot 10^{-4}$	$3.185 \cdot 10^{-3}$
SFNO + local integral kernel (ours)	4	128	31	$1.019 \cdot 10^6$	$\mathbf{2.624 \cdot 10^{-4}}$	$\mathbf{5.392 \cdot 10^{-4}}$

combining both differential and local integral kernels with the global convolution of the FNO. Our proposed model with these three components together outperforms models with only two of these three operations. While hyperbolic PDEs likely require only local receptive fields (due to finite information propagation (LeVeque, 1992)) and elliptic problems instead require global information, we observe that many practical problems involving a mixture of operators benefit from a hybrid approach such as ours.

We note that the Darcy flow problem is meant to motivate the need for our proposed differential kernels (and indeed, our best-performing model achieves 87% lower relative L^2-error than FNO). Since the ground truth operator in (11) is a differential operator, we expect (and observe) that for high accuracies, very local (i.e., differential) kernels are needed. In particular, we see that the baseline FNO performs poorly, and the local integral kernels do not provide additional benefits. However, we include them in Table 4 for completeness.

Finally, we perform super-resolution experiments, in which we train the model at a given resolution and then evaluate it at another resolution without fine-tuning. Figure 8 depicts super-resolution results at twice the training resolution for both the Darcy and shallow water equations. The corresponding numerical results are listed in Table 5. We observe that our approach generalizes well across resolutions and outperforms the baselines across all tested resolutions. For a detailed discussion of the super-resolution experiments, we refer the reader to Appendix C.6.

5. Conclusion

In this paper, we have demonstrated a novel framework for local neural operators. We have shown how convolutional layers can be constrained to realize neural operators that approximate differential operators in the continuous limit. Moreover, we have derived convolutions with local integral kernels from the general notion of an integral transform and the related graph neural operator. Finally, we have constructed localized neural operators on the sphere by using discrete-continuous convolutions (Ocampo et al., 2022).

The resulting neural operators introduce a strong inductive bias for learning operators with local receptive fields. In particular, their formulation ensures the same local operation everywhere in the domain. This equivariance (w.r.t. the underlying symmetry group) reduces the required number of learnable parameters and improves generalization.

Our numerical experiments demonstrate consistent improvements when existing neural operators with global receptive fields are augmented with the proposed localized convolutions, resulting in reductions in relative L^2-error of up to 72% over the corresponding baselines. We thus expect local neural operators to play an important role in solving real-world scientific computing problems with machine learning.

Acknowledgements

The authors thank Md Ashiqur Rahman for useful discussions about benchmarks and baselines. M. Liu-Schiaffini is grateful for support from the Mellon Mays Undergraduate Fellowship. J. Berner acknowledges support from the Wally Baer and Jeri Weiss Postdoctoral Fellowship. A. Anandkumar is supported in part by Bren endowed chair and by the AI2050 senior fellow program at Schmidt Sciences.

Impact Statement

The aim of this work is to advance the field of machine learning and scientific computing. While there are many potential societal consequences, none of them are immediate to require specifically being highlight here.

References

Azizzadenesheli, K., Kovachki, N., Li, Z., Liu-Schiaffini, M., Kossaifi, J., and Anandkumar, A. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics*, pp. 1–9, 2024.

Bekkers, E. J. B-spline CNNs on Lie groups. *arXiv preprint arXiv:1909.12057*, 2019.

Bonev, B., Kurth, T., Hundt, C., Pathak, J., Baust, M., Kashinath, K., and Anandkumar, A. Spherical Fourier neural operators: Learning stable dynamics on the sphere. *arXiv preprint arXiv:2306.03838*, 2023.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In *International conference on machine learning*, pp. 2990–2999. PMLR, 2016.

Cohen-Tannoudji, C., Diu, B., and Laloë, F. *Quantum mechanics; 1st ed.* Wiley, New York, NY, 1977. Trans. of : Mécanique quantique. Paris : Hermann, 1973.

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

Driscoll, J. and Healy, D. Computing fourier transforms and convolutions on the 2-sphere. *Advances in Applied Mathematics*, 15:202–250, 6 1994.

Ghosh, R. and Gupta, A. K. Scale steerable filters for locally scale-invariant convolutional neural networks. *arXiv preprint arXiv:1906.03861*, 2019.

Gupta, J. K. and Brandstetter, J. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

Hasani, E. and Ward, R. A. Generating synthetic data for neural operators. *arXiv preprint arXiv:2401.02398*, 2024.

Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J., and Aila, T. Alias-free generative adversarial networks. *Advances in Neural Information Processing Systems*, 34:852–863, 2021.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., et al. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023.

LeVeque, R. J. *Numerical Methods for Conservation Laws*. Birkhäuser Basel, 1992.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

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

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

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021.

Li, Z., Liu-Schiaffini, M., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Learning chaotic dynamics in dissipative systems. *Advances in Neural Information Processing Systems*, 35: 16768–16781, 2022a.

Li, Z., Meidani, K., and Farimani, A. B. Transformer for partial differential equations' operator learning. *arXiv preprint arXiv:2205.13671*, 2022b.

Li, Z., Kovachki, N. B., Choy, C., Li, B., Kossaifi, J., Otta, S. P., Nabian, M. A., Stadler, M., Hundt, C., Azizzadenesheli, K., et al. Geometry-informed neural operator for large-scale 3d pdes. *arXiv preprint arXiv:2309.00583*, 2023.

Marcos, D., Kellenberger, B., Lobry, S., and Tuia, D. Scale equivariance in cnns with vector fields. *arXiv preprint arXiv:1807.11783*, 2018.

Ocampo, J., Price, M. A., and McEwen, J. D. Scalable and equivariant spherical CNNs by discrete-continuous (DISCO) convolutions. *arXiv preprint arXiv:2209.13603*, 9 2022.

Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K., Hassanzadeh, P., Kashinath, K., and Anandkumar, A. FourCastNet: A global data-driven high-resolution weather model using adaptive Fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2 2022.

Rahman, M. A. and Yeh, R. A. Truly scale-equivariant deep nets with Fourier layers. *arXiv preprint arXiv:2311.02922*, 2023.

Rahman, M. A., George, R. J., Elleithy, M., Leibovici, D., Li, Z., Bonev, B., White, C., Berner, J., Yeh, R. A., Kossaifi, J., et al. Pretraining codomain attention neural operators for solving multiphysics PDEs. *arXiv preprint arXiv:2403.12553*, 2024.

Raonić, B., Molinaro, R., Rohner, T., Mishra, S., and de Bezenac, E. Convolutional neural operators. *arXiv preprint arXiv:2302.01178*, 2023.

Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241. Springer, 2015.

Shi, Y., Lavrentiadis, G., Asimaki, D., Ross, Z. E., and Azizzadenesheli, K. Broadband ground motion synthesis via generative adversarial neural operators: Development and validation. *arXiv preprint arXiv:2309.03447*, 2023.

Shocher, A., Feinstein, B., Haim, N., and Irani, M. From discrete to continuous convolution layers. *arXiv preprint arXiv:2006.11120*, 2020.

Sosnovik, I., Szmaja, M., and Smeulders, A. Scale-equivariant steerable networks. *arXiv preprint arXiv:1910.11093*, 2019.

Sosnovik, I., Moskalev, A., and Smeulders, A. Disco: accurate discrete scale convolutions. *arXiv preprint arXiv:2106.02733*, 2021.

Sun, H., Ross, Z. E., Zhu, W., and Azizzadenesheli, K. Phase neural operator for multi-station picking of seismic arrivals. *Geophysical Research Letters*, 50(24): e2023GL106434, 2023.

Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. PDEBench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35: 1596–1611, 2022.

Thomas, J. W. *Numerical partial differential equations: finite difference methods*, volume 22. Springer Science & Business Media, 2013.

Wang, S., Suo, S., Ma, W.-C., Pokrovsky, A., and Urtasun, R. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2589–2597, 2018.

Wen, G., Li, Z., Azizzadenesheli, K., Anandkumar, A., and Benson, S. M. U-FNO—an enhanced Fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

Wen, G., Li, Z., Long, Q., Azizzadenesheli, K., Anandkumar, A., and Benson, S. M. Real-time high-resolution CO_2 geological storage prediction using nested Fourier neural operators. *Energy & Environmental Science*, 16 (4):1732–1741, 2023.

Worrall, D. and Welling, M. Deep scale-spaces: Equivariance over scale. *Advances in Neural Information Processing Systems*, 32, 2019.

Xu, Y., Xiao, T., Zhang, J., Yang, K., and Zhang, Z. Scale-invariant convolutional neural networks. *arXiv preprint arXiv:1411.6369*, 2014.

Ye, X., Li, H., Jiang, P., Wang, T., and Qin, G. Learning transient partial differential equations with local neural operators. *arXiv preprint arXiv:2203.08145*, 2022.

Ye, X., Li, H., Huang, J., and Qin, G. On the locality of local neural operator in learning fluid dynamics. *arXiv preprint arXiv:2312.09820*, 2023.

Zhang, X., Wang, L., Helwig, J., Luo, Y., Fu, C., Xie, Y., Liu, M., Lin, Y., Xu, Z., Yan, K., et al. Artificial intelligence for science in quantum, atomistic, and continuum systems. *arXiv preprint arXiv:2307.08423*, 2023.

A. General differential kernels

A.1. Theoretical construction

In the following, we present the general idea for irregular grids, from which Proposition 3.1 follows as a special case. Let us first define the assumptions on our grids.

Regular discrete refinement: Let $\|\cdot\|$ be a norm on $\mathbb{R}^d$ and denote by $B_h(x) \subset \mathbb{R}^d$ the ball of radius $h \in (0,\infty)$ around $x \in \mathbb{R}^d$ w.r.t. $\|\cdot\|$. Further, let $D \subset \mathbb{R}^d$ be a domain and let $(h_\ell)_{\ell\in\mathbb{Z}}$ be a sequence that converges to zero. We call $(D_\ell)_{\ell\in\mathbb{Z}} \subset D$ a *regular discrete refinement* with widths $(h_\ell)_{\ell\in\mathbb{Z}}$ if there exists $N \in \mathbb{N}$ such that for all $\ell \in \mathbb{Z}$ and $x \in \mathbb{R}^d$ we have that

$$|B_{h_\ell}(x) \cap D_\ell| \leq N \quad \text{and} \quad \operatorname{span}\left(\left\{\begin{bmatrix}1\\y\end{bmatrix} - \begin{bmatrix}1\\x\end{bmatrix} : y \in B_{h_\ell}(x) \cap D_\ell\right\}\right) = \mathbb{R}^{d+1}.$$

The second assumption states that we can find an *affinely independent* subset in each ball. Note that, for instance, equidistant grids satisfy these assumptions.

First-order differential operator: We want to find bounded kernels $k(x,y)\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ with the following property: There exist $c \in \mathbb{R}$ and $b \in \mathbb{R}^d$ such that for all $v \in C^1(D,\mathbb{R})$, all $y \in \mathbb{R}^d$, and any regular discrete refinement $(D_\ell)_{\ell\in\mathbb{Z}} \subset \mathbb{R}^d$ with widths $(h_\ell)_{\ell\in\mathbb{Z}}$ it holds that

$$\lim_{\ell\to\infty} \sum_{x\in B_{h_\ell}(y)\cap D_\ell} k(x,y)v(x) = cv(y) + \nabla v(y)\cdot b. \tag{20}$$

We will now investigate which additional assumptions are needed. Let us fix $y \in \mathbb{R}^d$ and enumerate

$$(x_j)_{j=1}^m \coloneqq B_{h_\ell}(y) \cap D_\ell.$$

Then, we can use Taylor's theorem to show that

$$\sum_{x\in B_{h_\ell}(y)\cap D_\ell} k(x,y)v(x) = \sum_{j=1}^m k(x_j,y)\left(v(y) + \nabla v(y)\cdot(x_j-y) + \mathcal{O}(h_\ell)\right) \tag{21}$$

$$= v(y)\sum_{j=1}^m k(x_j,y) + \nabla v(y)\cdot\left(\sum_{j=1}^m k(x_j,y)(x_j-y)\right) + \mathcal{O}(m\|k\|_{L^\infty}h_\ell). \tag{22}$$

Since the kernel and m are bounded (uniformly over $\ell \in \mathbb{N}$), we have that $\mathcal{O}(m\|k\|_{L^\infty}h_\ell) = \mathcal{O}(h_\ell)$. As we want to guarantee convergence to $cv(y) + \nabla v(y)\cdot b$ for suitable $c \in \mathbb{R}$ and $b \in \mathbb{R}^d$ (independent of y), we need to satisfy that

$$c = \sum_{j=1}^m k(x_j,y) \tag{23}$$

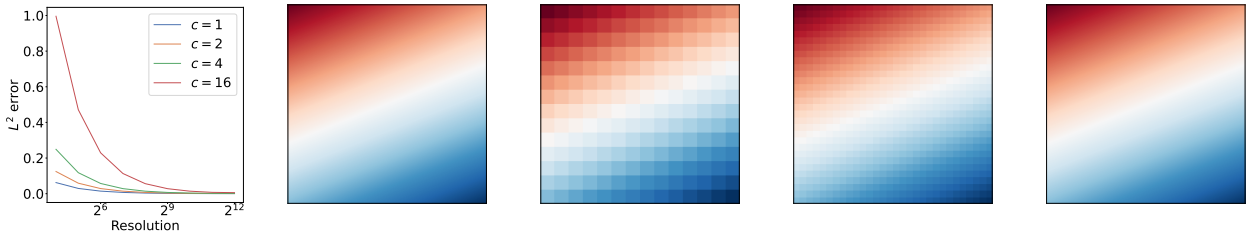
and that

$$b = \sum_{j=1}^m k(x_j,y)(x_j-y) \tag{24}$$

This yields the linear[7] system

$$\underbrace{\begin{bmatrix}1 & \dots & 1\\ x_1-y & \dots & x_m-y\end{bmatrix}}_{\in\mathbb{R}^{(d+1)\times n}} \begin{bmatrix}k(x_1,y)\\ \vdots \\ k(x_m,y)\end{bmatrix} = \begin{bmatrix}c\\b\end{bmatrix}. \tag{25}$$

Our assumptions on the refinement guarantee that we can find linearly independent columns such that we can solve the system. However, generally and by abuse of notation, the value of $k(x_j,y)$ depends on all the points $(x_j)_{j=1}^m$. However, for an equidistant grid, one can directly see that the convolutional kernel given in Proposition 3.1 satisfies (25) with $c = 0$.



(a) Convergence *(b)* Ground truth *(c)* Low-res prediction *(d)* Medium-res prediction *(e)* High-res prediction

Figure 5: Empirical evaluation of the proposed differential kernel. (a) L^2 errors at various resolutions and quadratic coefficient scales c. (b) True differential operator for $c = 1$. (c) Output of the differential kernel at a resolution of 32×32. (d) Output of the differential kernel at a resolution of 64×64. (e) Output of the differential kernel at a resolution of 4096×4096.

A.2. Empirical evaluation

In this section, we empirically evaluate the convergence of our proposed differential kernels to a differential operator as the resolution increases. In particular, we consider a randomly-initialized differential kernel satisfying the constraints discussed in Section 3.2 (mean zero and scaled by the resolution $\frac{1}{h}$, where h is the width of each grid cell in an equidistant grid).

To be able to compute the theoretical differential operator in closed form, we consider a simple parabola $v\colon [0,1]^2 \to \mathbb{R}^n$, given by

$$x \mapsto \|x\|^2 \left[c_1, \dots, c_n\right]^\top, \tag{26}$$

with coefficients $c_1, \dots, c_n$.

We define a randomly-initialized first-order differential kernel $K = (K_{ij})_{i,j=1}^{3,3} \subset \mathbb{R}^n$ (i.e., one differential kernel for each of the n input channels) subject to the constraints described in Section 3.2. Let $b_j \in \mathbb{R}^n$ denote the direction corresponding to the directional derivative corresponding to the differential kernel for channel j, see Appendix A.1. With the parabola as defined in (26), the output function of the true differential operator corresponding to K is given by

$$x \mapsto 2 \sum_{i=j}^{n} x \cdot b_j. \tag{27}$$

In this simple experiment, we set $n = 10$ and generate $c_1, \dots, c_n$ uniformly on the unit interval, multiplied by some scaling factor $c \in \{1, 2, 4, 16\}$. We discretize the parabola in (26) over different resolutions, convolve the kernel K with each of these discretizations, and compute the L^2 error with respect to the true differential operator. Figure 5 shows the L^2 error for these values of c and resolutions, as well as visualizations of the true differential operator and the one outputted by the differential kernel at three different resolutions.

From Figure 5, we observe a clear convergence to the differential operator as the resolution increases. For a fixed resolution, convergence is slower for greater magnitudes of the second derivative (i.e., larger c). This can be theoretically shown by estimating the remainder of the Taylor expansion in Appendix A.1.

B. Discrete-continuous convolutions

B.1. General Ideas

The following section outlines the basic ideas behind discrete-continuous convolutions as introduced by (Ocampo et al., 2022). To generalize the (continuous) convolution (5) to Lie groups and quotient spaces of Lie Groups, we consider the group convolution (see e.g. Cohen & Welling (2016)).

Definition B.1 (Group Convolution)**.** Let $\kappa, v : G \to \mathbb{R}$ be functions defined on the group G. The group convolution is given by

$$(\kappa \star v)(g) = \int \kappa(g^{-1}x) \cdot v(x) \, \mathrm{d}\mu(x), \tag{8}$$

[7] For fixed y.

where $g, x \in G$ and $\mathrm{d}\mu(x)$ is the invariant Haar measure on G.

Remark B.2. In some cases, signals are not defined on a group but rather on a quotient space G/H, where H is a subgroup of G. In such cases, a convolution may still be defined by taking $g \in G/H$. For an example, see spherical convolutions (Driscoll & Healy, 1994; Ocampo et al., 2022).

While group convolutions can typically be computed by generalized Fourier transforms on the corresponding manifolds, their usage is generally preferred if the convolutions are non-local operators, i.e., the convolution kernel κ is not compactly supported. On the periodic domain $\mathbb{T}^d$ (i.e., Euclidean space with periodic boundaries), such convolutions are typically computed discretely by directly sliding the kernel.

Definition B.3 (DISCO convolutions)**.** Given a quadrature rule with quadrature points $x_j \in G$ and quadrature weights q_j, we approximate the group convolution (8) with the discrete sum

$$(\kappa \star v)(g) = \int \kappa(g^{-1}x) \cdot v(x)\, \mathrm{d}\mu(x) \approx \sum_{j=1}^{m} \kappa(g^{-1}x_j) \cdot v(x_j)\, q_j. \tag{28}$$

In particular, the group action g is applied analytically to the kernel function κ, whereas the integral is approximated using the quadrature rule.

For a discrete set of output locations g_i, this becomes a straight-forward matrix-vector multiplication

$$\sum_{j=1}^{m} \kappa(g_i^{-1}x_j) \cdot v(x_j)\, q_j = \sum_{j=1}^{m} K_{ij} \cdot v(x_j)\, q_j \tag{29}$$

with $K_{ij} = \kappa(g_i^{-1}x_j)$. In the case where κ is compactly supported, K_{ij} is a sparse matrix with the number of non-zero entries per row depending on the resolution of the grid x_j and the support of κ. To obtain a learnable filter, κ is parametrized as a linear combination of a chosen set of basis functions.

B.2. DISCO convolutions in one dimension

For the sake of simplicity, we discuss the simple one-dimensional case on $D = [0, 1]$ with periodic boundary conditions. We note that this corresponds to the circle group (or the torus) $\mathbb{T}$. For any element $y \in D$, the corresponding group operation T_y is the translation $T_y : D \to D$, $x \mapsto x \oplus y$, where we denote by $x \oplus y$ a modular shift such that the result remains in D.

Then, the DISCO convolution in one dimension, for Lebesgue square-integrable functions v and κ becomes

$$(\kappa \star v)(y) = \int_{[0,1]} \kappa(T_y^{-1}x)v(x)\, \mathrm{d}x = \int_{[0,1]} \kappa(x - y)v(x)\, \mathrm{d}x \approx \sum_{j=1}^{m} \kappa(x_j - y)\, v(x_j)\, q_j, \tag{30}$$

for suitable quadrature points $D^h = \{x_j\}_{j=1}^m$ with corresponding quadrature weights $\{q_j\}_{j=1}^m$.

To parameterize the filter κ, we pick a finite support $[0, x_{\text{cutoff}}]$ with L equidistant collocation points $\xi^{(\ell)} \in [0, x_{\text{cutoff}}]$ and corresponding hat functions. The ℓ-th hat function is then defined as

$$\kappa^{(\ell)}(x) = \begin{cases} \frac{x - \xi^{(\ell-1)}}{\xi^{(\ell)} - \xi^{(\ell-1)}} & \text{for } \xi^{(\ell-1)} \le x < \xi^{(\ell)} \\ \frac{\xi^{(\ell+1)} - x}{\xi^{(\ell+1)} - \xi^{(\ell)}} & \text{for } \xi^{(\ell)} \le x < \xi^{(\ell+1)} \\ 0 & \text{else,} \end{cases} \tag{31}$$

where $\xi^{(0)}, \xi^{(L+1)} \in [0, x_{\text{cutoff}}]$ are suitable boundary points. The resulting filter is obtained as a linear combination $\kappa = \sum_{\ell=1}^{L} \theta^{(\ell)} k^{(\ell)}$ with trainable parameters $\theta^{(\ell)}$. Plugging this into (29), we obtain the trainable DISCO convolution

$$\sum_{j=1}^{m} \kappa(x_j - y_i)\, v(x_j)\, q_j = \sum_{\ell=1}^{L} \sum_{j=1}^{m} \theta^{(\ell)}\, \kappa^{(\ell)}(x_j - y_i)\, v(x_j)\, q_j = \sum_{\ell=1}^{L} \sum_{j=1}^{m} \theta^{(\ell)} K_{ij}^{(\ell)}\, v(x_j)\, q_j, \tag{32}$$

where $K_{ij}^{(\ell)} = \kappa^{(\ell)}(x_j - y_i)$ are the shifted filter functions.

Let us now consider the special case of an equidistant grid D^h, i.e., $x_{j+1} - x_j = h$, and a trapezoidal quadrature rule $q_j = h$. Let us further assume that the output points y_i coincide with the grid points and that the collocation points $\xi^{(\ell)}$ are given as the first L grid points. Then, due to the property of the hat functions, $K_{ij}^{(\ell)}$ can only contain either 0 or 1 and we obtain the circulant convolution matrices given by

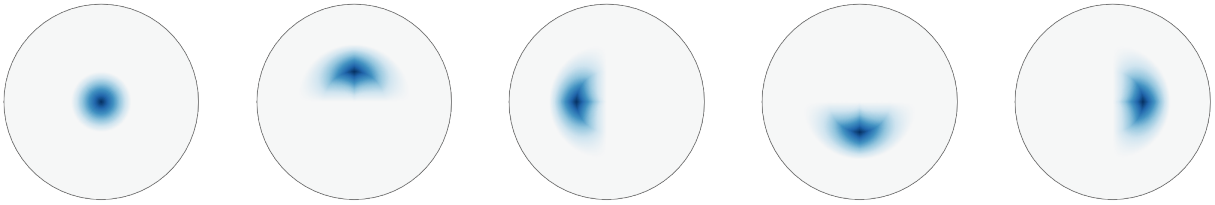
$$K^{(1)} = (e_1, e_2, e_3, \dots, e_m), \qquad K^{(2)} = (e_2, e_3, \dots, e_m, e_1), \qquad K^{(3)} = (e_3, \dots, e_m, e_1, e_2), \quad \dots$$

where $e_i \in \mathbb{R}^m$ is the i-th standard basis vector. For regular grids on planar geometries, we can thus efficiently implement the DISCO convolution in (32) in terms of highly-optimized CUDA kernels based on common-place convolutional layers. This observation also generalizes to higher dimensions, where the same discrete kernel is obtained when the kernel is continuously shifted on the grid.

B.3. DISCO convolutions on the sphere

For general group actions $g \in SO(3)$, the outcome of the group convolution (8) will be a function defined on $SO(3)$. This is due to $\mathbb{S}^2$ not being a group but rather a manifold on which $SO(3)$ acts. We can see this by fixing the north-pole $n = [0, 0, 1]^\top$ and applying any rotation $g \in SO(3)$ to it. This will trace out the whole sphere despite the north pole eliminating one of the Eulerian rotation angles. Therefore, to ensure that the result of the convolution is still a function defined on $\mathbb{S}^2$, we can simply restrict g in (8) to rotations in $SO(3)/SO(2)$, which is isomorphic to $\mathbb{S}^2$. Formally, this can be achieved by fixing the first of the three Euler angles parameterizing g to 0.

As basis functions, we pick a set of piecewise linear basis functions as in (31). To accommodate anisotropic kernels, collocation points are distributed in an equidistant manner along both radius and circumference. More precisely, the first collocation point lies at the center, and for each consecutive ring, a fixed amount of collocation points is distributed along the circumference. The resulting basis functions are illustrated in Figure 6, for a cutoff radius of $r_{\text{cutoff}} = 0.1\pi$.



(a) $\ell = 1$ *(b)* $\ell = 2$ *(c)* $\ell = 3$ *(d)* $\ell = 4$ *(e)* $\ell = 5$

Figure 6: Radial, piecewise linear basis functions for the approximation of anisotropic filters on the sphere.

C. Implementation details

A numerical comparison of our methods with baseline FNO and U-Net architectures can be found in Table 4. In this section, we outline the implementation details for our numerical experiments.

For Darcy flow, Navier-Stokes, and the shallow water equations, training is conducted by minimizing the squared L^2-loss until convergence is reached. Our U-Net baseline is adapted from the model and code of the PDE Arena benchmark (Gupta & Brandstetter, 2022). For the FNOs (Li et al., 2020a) and SFNOs (Bonev et al., 2023), we use the implementation in the `neuraloperator` and `torch-harmonics` libraries. Moreover, for all experiments, GELU activation functions and the Adam optimizer are used.

For these three problems, we trained the FNO/SFNO-based models with varying widths and modes, while keeping the overall number of parameters approximately constant. We present the best results for each problem and macro-architecture in Table 4. For the models with local layers, we found that a larger embedding dimension and fewer modes can often improve performance. We conjecture that the increased embedding dimension confers additional expressivity to the local kernels. This also suggests that local operators are an important inductive bias for these problems. When supplementing the FNO/SFNO blocks with additional branches, we also observed improved convergence by scaling the initial parameters or the output by $n^{-1/2}$, where n is the number of branches. A detailed experimental setup is outlined for these three datasets in the

following subsections. We also present results for the 2D diffusion-reaction equation and for super-resolution experiments on Darcy flow and the shallow water equations.

C.1. 2D Darcy flow equation

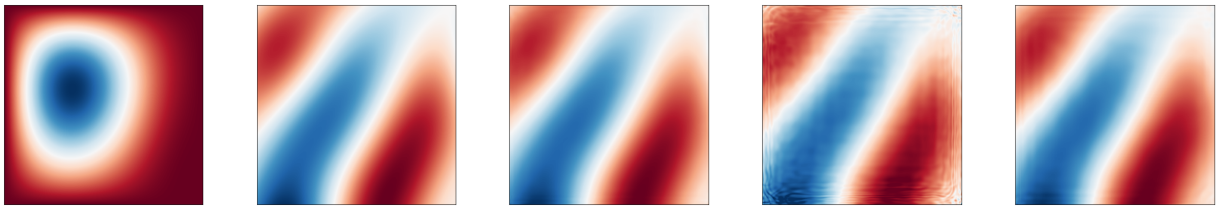
In the 2D Darcy flow setting, we generate our data as described in Section 4.1. For the input functions, we consider random linear combinations of eigenfunctions of the Laplace operator with zero Dirichlet boundary conditions, i.e.,

$$u(x) = \sum_{i,j=1}^{20} \frac{c_{ij}}{\sqrt{(i\pi)^2 + (j\pi)^2}} \sin(i\pi x_1) \sin(j\pi x_2), \quad x \in D,$$

with i.i.d. $c_{ij} \sim \mathcal{N}(0, 1/(i+j))$. We train and test our models and baselines with data discretized onto a 256×256 regular grid. We use 10000 training samples and 2000 testing samples. As a baseline, we compare our proposed models with FNO (Li et al., 2020a) and the U-Net architecture of Gupta & Brandstetter (2022). We note that this U-Net architecture is not agnostic to the discretization (Kovachki et al., 2021), see also Section 3.1. We compare these baselines against our proposed models: the architecture using convolutions with Fourier (i.e., FNO), differential, and integral kernels (Figure 2), as well as architectures using only FNO and differential kernels or only FNO with the proposed integral kernels. Figure 7 compares the predictions of each model.

We choose all hyperparameters such that the overall number of parameters of all compared models is similar. The number of layers, number of Fourier modes, and embedding dimension are shown in Table 4. Models using convolutions with differential and local integral kernels use these layers in parallel to the Fourier layers and pointwise skip connection on all layers. We use reflective padding for the convolutional operations in the differential and local integral kernels. For all relevant models, the local integral kernels use a radius cutoff of 0.007 on $[-1, 1]^2$, and they are parameterized by five radial, piecewise linear basis functions for the approximation of anisotropic filters (analogous to Figure 6 on the plane). The differential kernels are parameterized as 3×3 convolutional kernels over the regular grid. For our U-Net baseline, we use 3×3 convolutional kernels with 2 residual blocks for each resolution (two downsampling and two upsampling) and three layers within each block, with channel multipliers of $1, 2, 4$ for each layer within a block.

Training is conducted by minimizing the squared L^2-loss for 70 epochs on a single NVIDIA P100 GPU, which is sufficient to achieve convergence on all models. We use a step learning rate decay scheduler, starting at 10^{-3} and halving every 10 epochs. The results are shown in Table 4.



(a) Input *(b)* Ground truth *(c)* Ours *(d)* FNO *(e)* U-Net

Figure 7: Comparison of models from Table 4: The outputs of our best-performing model, FNO, and U-Net on a randomly selected input pressure function from the Darcy flow problem. Edge artifacts are very prevalent in the FNO predictions, and they are less dominant in the U-Net predictions.

C.2. 2D Navier-Stokes Equations

For the 2D Navier-Stokes equations (Kolmogorov flows), we use the same experimental setup and dataset as Li et al. (2022a), which sets $m = 4$ and $\text{Re} = 5000$ in (12) and uses a temporal discretization of $\tau = 1$ on a 128×128 regular grid. The initial conditions are sampled from a Gaussian measure as described in Li et al. (2022a), and the equation is solved with a pseudo-spectral solver. We compare the same five models as in the Darcy experiment.

As in the Darcy setting, we choose all hyperparameters such that the overall number of parameters for all the models is similar. The number of layers, number of Fourier modes, and embedding dimension are shown in Table 4. Models using convolutions with differential and local integral kernels use these layers in parallel to the Fourier layers and pointwise skip

connection on all layers. We enforce periodic boundary conditions during padding for the convolutional operations in the differential and local integral kernels. For all relevant models, the local integral kernels use a radius cutoff of 0.05π on the torus, and they are parameterized by five radial, piecewise linear basis functions for the approximation of anisotropic filters (analogous to Figure 6 on the plane). The differential kernels are parameterized as 3×3 convolutional kernels over the regular grid. Our U-Net baseline is set up in the same way as in the Darcy experiment.

Training was performed for 136 epochs on a single NVIDIA RTX 4090 GPU with an exponentially decaying learning rate, starting at 10^{-3} and halving every 33 epochs. The results are shown in Table 4.

C.3. Diffusion-Reaction equation

For the 2D Diffusion-Reaction equation, we use the same experimental setup and dataset as Takamoto et al. (2022). The reference solution is computed using a finite-volume method in space and a fourth-order Runge-Kutta method in time. The dataset consists of 900 training samples and 100 validation samples discretized on a 128×128 regular grid with 100 equidistant time-steps in the interval $[0, 5]$ and Gaussian initial conditions. The task is to predict the state of the variables (u, v) at the next time-step from the states at the previous 10 steps. The errors are measured for the full autoregressive roll-out as in the implementation of PDEBench at github.com/pdebench/PDEBench.

We use the same experimental setup and implementation as Takamoto et al. (2022) and only adapt the learning rate, the number of modes, and the embedding dimension. We then add our proposed layers to (a subset of) the FNO blocks and experiment with both reflective and replicate padding. In particular, we parametrize the local integral kernels by five radial, piecewise linear basis functions and use 3×3 convolutional kernels for the differential kernels.

We train on a single NVIDIA RTX 4090 GPU for 500 epochs with early stopping (using the same criteria as Takamoto et al. (2022)). Moreover, we use an exponentially decaying learning rate, starting at 10^{-4} and halving every 100 epochs. We present our best results in Table 2 and refer to Takamoto et al. (2022) for details on the metrics. We compare against the baselines by Takamoto et al. (2022) and note that we also improve their FNO baseline results. However, our local integral and differential kernels still provide a significant improvement with fewer parameters. Specifically, we reduced the modes from 24 to 16 and increased the embedding dimension from 20 to 29.

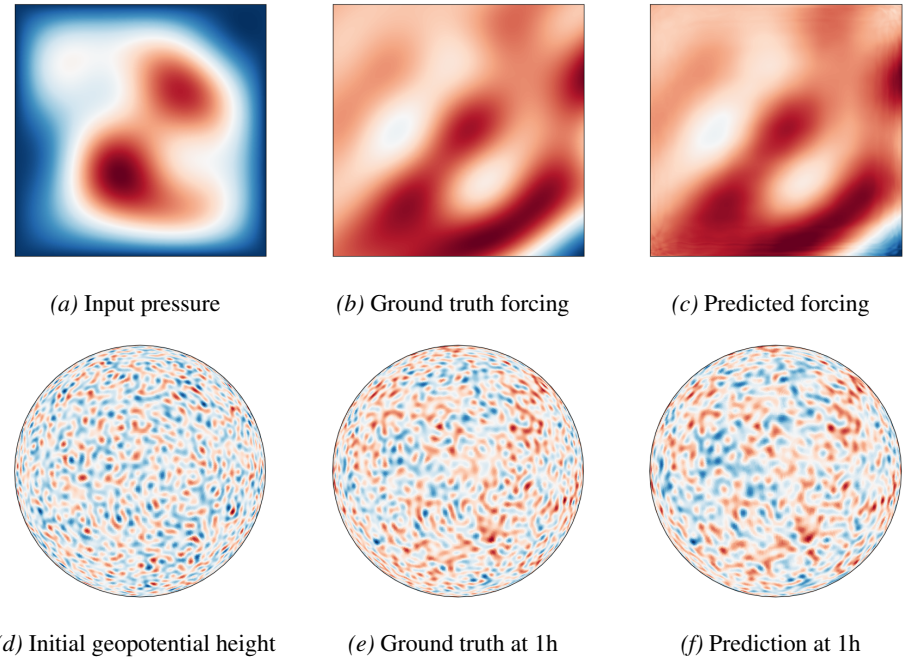
C.4. Shallow water Equations

For the shallow water equations, we use the dataset presented by Bonev et al. (2023), which uses a Gaussian random field to generate initial conditions on an equiangular lat-lon-grid on the sphere at a resolution of 256×512 and solves for φ, u at a lead time of one hour. The target solution is computed using a spectral solver, which takes 24 Euler steps[8]. Physical constants such as the sphere's radius or the Coriolis force are set to match those of Earth. The numerical solver uses 150 explicit Euler steps to advance the solution 1 hour in time. The right-hand side is discretized using the spectral basis provided by the Spherical Harmonics. For a detailed description of the dataset, we refer the reader to Bonev et al. (2023) and the corresponding implementation in the `torch-harmonics` package.

As a baseline for our experiments, we use the SFNO architecture as presented by Bonev et al. (2023), where the embedding dimension is adjusted to 32 to obtain a manageable parameter count. Moreover, we adapt the U-Net architecture by Gupta & Brandstetter (2022) to the spherical domain by replacing all spatial (i.e., not the 1×1) convolutions with DISCO convolutions on the sphere. Therefore, the resulting architecture is a spherical U-Net similar to the architecture presented by Ocampo et al. (2022). Moreover, due to the DISCO convolutions' discretization-agnostic nature, this architecture can be interpreted as a neural operator. Finally, we augment the SFNO architecture with local DISCO convolutions to obtain the proposed architecture; see Section 3.5.

For all three architectures, hyperparameters were chosen to achieve roughly similar parameter counts. The learning rates for each architecture were determined with a quick parameter sweep, resulting in $3 \cdot 10^{-4}$ for the spherical U-Net and $3 \cdot 10^{-3}$ for both neural operators. As a learning rate scheduler, we use the policy of halving the learning rate upon a plateauing of the loss. Training was performed for 100 epochs on a single NVIDIA RTX A6000 GPU, which was sufficient to achieve convergence on all considered models.

[8]Dataset and solver are taken from the `torch-harmonics` package at github.com/NVIDIA/torch-harmonics.



(a) Input pressure *(b)* Ground truth forcing *(c)* Predicted forcing

(d) Initial geopotential height *(e)* Ground truth at 1h *(f)* Prediction at 1h

Figure 8: Randomly selected super-resolution samples for the Darcy flow (top row) and shallow water (bottom row) problems.

C.5. Flow past a cylinder

To demonstrate the capability of dealing with unstructured representations, we use the dataset provided by Rahman et al. (2024) to train a model for predicting velocity and pressure fields for a Navier-Stokes problem in a channel with a suspended cylinder and attached membrane (see Figure 4). We use the dataset with a viscosity of $\mu = 1.0$, which corresponds to a Reynolds number of $\mathrm{Re} = 2000$. To deal with the unstructured data, we employ a local integral convolution layer in both the encoder and decoder with a cutoff radius of $r_{\text{cutoff}} = 0.052$ and 5 piecewise-linear basis functions. This layer transforms the unstructured data to a regularly spaced grid of 48×192 in the internal representation. The processor part of the architecture then consists of 4 FNO blocks with local integral kernels, which have the same filter basis as the convolutions used in the encoder/decoder. The embedding dimension is fixed at 16 and all Fourier modes are kept in the internal representation. Overall, this leads to a parameter count of $9.6 \cdot 10^6$. The architecture is trained on a dataset of 250 samples for 50 epochs using the Adam optimizer and a learning rate of $2 \cdot 10^{-3}$. The results alongside baselines from Rahman et al. (2024) are reported in Table 3.

C.6. Zero-shot super-resolution results

In this paper, we propose two methods to embed the inductive bias of locality into neural operator architectures. The key distinction between our proposed methods and CNN-based architectures is that our methods are agnostic to the discretization of the input function. In this section, we present and discuss the super-resolution capabilities of our proposed models. In particular, we focus on two examples: (1) the Darcy flow equation to showcase our differential layers on a regular Cartesian grid and (2) the shallow water equation to demonstrate super-resolution for our local integration on the sphere. The experimental setting that we consider is that of *zero-shot super-resolution*. In particular, suppose that the model has been trained at a particular training resolution (or at multiple different resolutions). Given input at a higher resolution than the training resolution, the task of zero-shot super-resolution is to then predict the output function at this higher resolution and evaluate the resulting model error.

In the Darcy flow setting, we use the same setup and dataset as described in Section 4.1. We train two models on data sampled at a 256×256 regular grid and evaluate on a 128×128 regular grid ($\frac{1}{2}$x resolution and $\frac{1}{4}$x the number of points), 512×512 grid (2x super-resolution), and 1024×1024 grid (4x super-resolution). For the shallow water equations, a similar approach is taken. We take the models in Section 4.4 which were trained at a resolution of 256×512 and apply them to data generated at a resolution of 128×256, 512×1024, and 1024×2048.

Table 5: Zero-shot super-resolution results for Darcy flow and the spherical shallow water problems. The validation error is reported in terms of relative L^2-error at various resolutions. Autoregressive rollouts are super-resolved in the sense that the rollout is performed at the high resolution.

Model	Parameters				Relative L^2-Error			
					resolution (relative to training resolution)			
	# Layers	# Modes	Embedding	# Parameters	$1/2\times$	$1\times$	$2\times$	$4\times$
Darcy Flow								
FNO	4	20	41	$2.715 \cdot 10^6$	$1.475 \cdot 10^{-1}$	$5.867 \cdot 10^{-2}$	$8.646 \cdot 10^{-2}$	$7.731 \cdot 10^{-2}$
FNO + diff. (ours)	4	20	40	$2.599 \cdot 10^6$	$\mathbf{1.174 \cdot 10^{-1}}$	$\mathbf{5.851 \cdot 10^{-2}}$	$\mathbf{7.774 \cdot 10^{-2}}$	$\mathbf{6.681 \cdot 10^{-2}}$
Spherical Shallow Water Equations								
Spherical U-Net	17	-	32	$1.639 \cdot 10^6$	$5.586 \cdot 10^{-3}$	$6.160 \cdot 10^{-4}$	$3.386 \cdot 10^{-3}$	$4.102 \cdot 10^{-3}$
SFNO	4	128	32	$1.066 \cdot 10^6$	$1.342 \cdot 10^{-3}$	$9.220 \cdot 10^{-4}$	$3.830 \cdot 10^{-3}$	$4.419 \cdot 10^{-3}$
SFNO + loc. int. (ours)	4	128	31	$1.019 \cdot 10^6$	$\mathbf{8.673 \cdot 10^{-4}}$	$\mathbf{2.624 \cdot 10^{-4}}$	$\mathbf{3.341 \cdot 10^{-3}}$	$\mathbf{4.097 \cdot 10^{-3}}$

Table 5 provides the details of the models we used for these experiments as well as the results of our super-resolution experiments with our proposed models. In the Darcy setting, we compare the baseline FNO to the FNO augmented with the differential kernel. On the sphere, we make use of the fact that the spherical U-Net presented in (Ocampo et al., 2022) is already a neural operator due to its discretization-independence. As such, we can use it alongside the SFNO as a baseline for zero-shot super-resolution on the sphere.

As with all neural operator architectures, during training there is the possibility of overfitting to the training resolution. For instance, FNO may learn features in Fourier space that are intrinsically tied to the resolution of the input function. Similarly, it is possible that our proposed differential and integral convolutional operators will learn a function of the training discretization. This effect can be remedied by using high-resolution training data where the local details are fully resolved. This minimum required resolution of the training data is thus a function of the smoothness of the input function. For this reason, we decided to exclude the 2D Navier-Stokes problem from our super-resolution experiments, since training at a resolution that sufficiently resolves the local dynamics would be prohibitvely expensive.

In our experiments, we noticed that our differential layers tend to incur some discretization errors when trained on data that is not of sufficiently high resolution. If differential convolutions are present in consecutive layers in the model, this error can propagate quickly. As such, we found that the best-performing model on zero-shot super-resolution for the Darcy flow problem is a model with a differential convolution in only the first layer. Using fewer differential layers in our model reduces the expressivity, but we find that the super-resolution capabilities are still better than the baseline FNO.

Lastly, we would like to note that in our experiments, FNO had a larger error near the boundary in non-periodic problems, as the FFT used in FNO assumes periodic boundary conditions. We note that our proposed differential layer can help reduce the error at the boundary caused by FNO, but some of these effects may still be present (see Figure 8).

C.7. Computational efficiency of differential and local integral kernels

On equidistant grids, our differential and local integral kernels can be implemented as standard convolutional kernels, which are heavily optimized for GPU performance. The computational complexity of our differential kernels is linear in the number of grid points since the size of the convolutional kernel remains constant regardless of resolution. In contrast, the size of the discretized local integral kernels does increase with input resolution. While this may be expensive for some high-resolution datasets, in our experiments, we found that the heavily-optimized convolutions in deep learning libraries can help reduce this computational burden. As a consequence, local integral kernels are more efficient than GNO (Li et al., 2020b) and other graph-based approaches on equidistant grids.

Moreover, the local integral kernels can also be applied to irregular grids, where they are implemented as sparse matrix multiplications. This tends to outperform GNOs (Li et al., 2020b), which need to evaluate a neural network on the graph. The complexity of the local integral layer is linear with a constant that depends on the sparsity of the matrix. This, in turn, depends on the support of the local kernel; choosing a fixed support will scale quadratically with the resolution, whereas a support that matches the resolution will scale linearly in practice.