General Covariance Data Augmentation for Neural PDE Solvers

Vladimir Fanaskov [1]   **Tianchi Yu** [1]   **Alexander Rudikov** [1,2]   **Ivan Oseledets** [1,3]

Abstract

The growing body of research shows how to replace classical partial differential equation (PDE) integrators with neural networks. The popular strategy is to generate the input-output pairs with a PDE solver, train the neural network in the regression setting, and use the trained model as a cheap surrogate for the solver. The bottleneck in this scheme is the number of expensive queries of a PDE solver needed to generate the dataset. To alleviate the problem, we propose a computationally cheap augmentation strategy based on general covariance and simple random coordinate transformations. Our approach relies on the fact that physical laws are independent of the coordinate choice, so the change in the coordinate system preserves the type of a parametric PDE and only changes PDE's data (e.g., initial conditions, diffusion coefficient). For tried neural networks and partial differential equations, proposed augmentation improves test error by 23% on average. The worst observed result is a 17% increase in test error for multilayer perceptron, and the best case is a 80% decrease for dilated residual network.

1. Introduction

Machine learning is increasingly used to solve partial differential equations (PDEs). The especially fruitful idea is to learn a computationally cheap but sufficiently accurate surrogate for the classical solver (Hennigh, 2017), (Li et al., 2020), (Tripura & Chakraborty, 2022), (Lu et al., 2021a), (Stachenfeld et al., 2021). The most reliable training strategy is to generate input-output pairs with a classical solver and fit a neural network of choice with a standard L_2 loss (regression setting).

[1]Skoltech, Center for Artificial Intelligence Technology [2]Marchuk Institute of Numerical Mathematics of the Russian Academy of Sciences [3]Artificial Intelligence Research Institute. Correspondence to: Vladimir Fanaskov <V.Fanaskov@skoltech.ru>.

An alternative we do not consider here is to resort to a so-called physics-informed setting when the loss is a L_2 norm of PDE residual evaluated at certain points (Wang et al., 2021), (Li et al., 2021). This way, one avoids data generation by a classical solver. Arguably, it is currently recognized that the whole process is inefficient (Lu et al., 2021b), (Karnakov et al., 2022).

In the regression setting the size of the generated dataset is usually limited owing to the restrictions on the computation budget. Deep learning is data-hungry, so ways to cheaply increase the number of data points available for training are highly desirable. Numerous augmentation techniques serve this purpose in classical machine learning (Shorten & Khoshgoftaar, 2019), (Wen et al., 2020). For scientific machine learning, literature on augmentation is scarce (Brandstetter et al., 2022), (Li et al., 2022a). In this note, we contribute a new way to augment datasets for neural PDE solvers.

The central idea behind our approach is **the principle of general covariance**. General covariance states that physical phenomena do not depend on the choice of a coordinate system (Post, 1997), (Emam, 2021). Mathematically, the covariance means the physical fields are geometric objects (tensors) with particular transformation laws under the change of coordinates (Eglit et al., 1996), (Liseikin, 2017). In exceptional cases, these transformation laws leave governing equations invariant (symmetry transformation), but in most cases, it is only the form of the equations that persists. More specifically, for parametric partial differential equations, suitably chosen coordinate transformation induces the change of problem data (e.g., permeability field, convection coefficients, source term, initial or boundary conditions, e.t.c.). We use this fact to build a computationally cheap and broadly applicable augmentation strategy based on simple random coordinate transformations. To evaluate the efficiency of our approach, we perform empirical tests on the two-way wave, convection-diffusion, and stationary diffusion equations using several variants of Fourier Neural Operator (FNO) (Li et al., 2020), Deep Operator Network (DeepONet) (Lu et al., 2021a), Multilayer Perceptron (MLP) (Haykin, 1994), Dilated Residual Network (DilResNet) (Yu & Koltun, 2015), (Stachenfeld et al., 2021) and U-Net (Ronneberger et al., 2015). Both for one-dimensional and two-dimensional PDEs proposed augmentation tech-

nique improves test error by 23% on average and up to 80% in the most favorable cases.

Contributions:

1. Easily extendable, architecture-agnostic augmentation procedure based on general covariance.

2. Cheap algebraic random grids in R^n based on cumulative distribution function and transfinite interpolation.

3. Comprehensive set of experiments showing that augmentation helps to improve test error for different architectures and parametric families of PDEs.

Code and datasets are available on `https://github.com/VLSF/augmentation`.

2. Basic augmentation example

Before dwelling upon technical details, we provide a simple example of our approach for a parametric boundary-value problem

$$\frac{d}{dx}\left(a(x)\frac{d}{dx}u(x)\right) = f(x),$$
$$x \in [0,1], \ u(0) = u(1) = 0. \tag{1}$$

Suppose that a and f are chosen reasonably, so the unique solution exists.[1] The usual way to approximate this solution is to use finite-element discretization

$$u(x) = \sum_{i=1}^{N} \phi_i(x)u_i, \tag{2}$$

where $\phi_i(x)$ are piecewise linear functions such that $\phi_i(x_j) = \delta_{ij}$ for $x_j = j/(N+1)$, $j = 1, \ldots, N$, i.e., the hat functions. After that, the differential equation (1) in a weak form is equivalent to $N \times N$ linear system, and the solution is straightforward. The obtained solution is known not only on the uniform grid $\mathcal{G} = \{j/(N+1), \ j = 0, \ldots, N+1\}$ but everywhere in the domain thanks to the closed-form representation (2).

Using described procedure one can generate a dataset with features $F_i = (a_i(\mathcal{G}), f_i(\mathcal{G}))$ and targets $T_i = (u_i(\mathcal{G}))$, $i = 1, \ldots, N_{\text{samples}}$. As a rule, features, a_i and f_i in our case, are samples from some distribution (Kovachki et al., 2021) or typical inputs needed for a particular application, e.g., (Pathak et al., 2022).

Our main observation is that when the PDE is known, it is possible to extract more information from each obtained

---

[1] The formal statement on existence is available in (Evans, 2010), but it is largely irrelevant to our discussion.

solution using coordinate transformation. Suppose $y(\xi)$ is analytic strictly monotonic function from $[0,1]$ to $[0,1]$ such that $y(0) = 0$, $y(1) = 1$. We use $x \equiv y(\xi)$ as coordinate transformation and rewrite (1) in coordinates ξ as follows

$$\frac{d\xi}{dy}\frac{d}{d\xi}\left(a(y(\xi))\frac{d\xi}{dy}\frac{d}{d\xi}u(y(\xi))\right) = f(y(\xi)),$$
$$\xi \in [0,1], \ u(y(0)) = a, \ u(y(1)) = b. \tag{3}$$

As we see, transformed equation (3) has the same parametric form as the original one (1). As a consequence, if a triple of functions $a(x), u(x), f(x)$ solve (1), the triple of modified functions $a(y(x))\frac{dx}{dy}, u(y(x)), f(y(x))\frac{dy}{dx}$ also solve the same equation (1), where we rename variable ξ in (3) to x. So we can generate novel solutions from the old ones using smooth coordinate transformations and interpolation.

To complete a description of the augmentation, we need to explain how to generate smooth coordinate transformations. Since any strictly monotonic positive function that maps $[0,1]$ constitutes a valid coordinate transformation, we propose to use cumulative distribution functions with strictly positive probability density. It is easy to come up with many parametric families of probability densities. For example, we can use trigonometric series and define

$$p(x) = 1 + \sum_{k=1}^{N} \frac{(c_k \cos(2\pi kx) + d_k \sin(2\pi kx))}{c_0},$$
$$c_0 = \sum_{k=1}^{N} (|c_k| + |d_k|) + \beta, \ \beta > 0. \tag{4}$$

After integration, we obtain a cumulative distribution function that serves as a coordinate transformation

$$y(x) = x + \sum_{k=1}^{N} \frac{(c_k \sin(2\pi kx) + d_k(1 - \cos(2\pi kx)))}{2\pi k c_0}. \tag{5}$$

The whole augmentation procedure for elliptic equation (1) can be compactly written as

$$a(x), \underset{\text{solve (1)}}{u(x)}, f(x) \longrightarrow a(y(x))/\frac{dy}{dx}, u(y(x)), f(y(x))\frac{dy}{dx}. \tag{6}$$
$$\underset{y(x) \text{ from (5)}}{}$$

Figure 1 illustrates the proposed approach for elliptic equation (1) and a particular set of transformations (5).

To summarize, our augmentation approach consists of three stages:

1. Generate a sufficiently smooth coordinate transformation $y(\xi)$.

2. Interpolate features and targets on discrete grid $y(\xi_j)$, $\xi_j = j/(N+1)$, $j = 0, \ldots, N+1$.
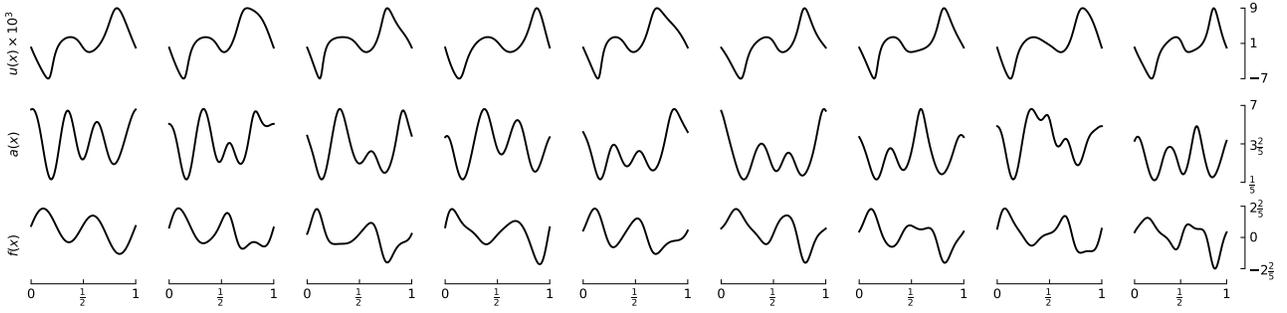
Figure 1. Example of data augmentation for elliptic equation (1). The first column on the left contains features and target that solves (1). All other columns are obtained from the first one with transformation (6). Coordinate transformations are generated according to (5) with parameters $N = 3$, $\beta = 1$ and coefficients c_k, d_k, $k = 1, 2, 3$ sampled from standard normal distribution.

3. Adjust interpolated features and targets according to the transformations law for PDE evaluated in new co-ordinates $y(\xi)$.

This procedure can be applied for as many coordinate transformations $y(\xi)$ as needed and requires only cheap interpolation, so the overall cost is $O(N)$ for each sample, where N is the number of grid points.

In the Section 3, we show how to generalize results illustrated here for other partial differential equations and higher dimensions.

3. Augmentation by General Covariance

In Section 2, we explained that two principal components of the proposed augmentation approach are grid generation and transformation law for PDE in question. Here we show how to extend results from Section 2 to a more general setting.

Everywhere in this section, Einstein's summation notation is used, e.g., $a_\alpha b^\alpha \equiv \sum_\alpha a_\alpha b^\alpha$.

3.1. How to construct coordinate transformations in the general case

We define coordinate transformations in $D \geq 1$ as one-to-one analytic mapping

$$\boldsymbol{x}(\boldsymbol{\xi}) : [0, 1]^D \longrightarrow [0, 1]^D \tag{7}$$

In Section 2, we outlined a particular scheme to construct families of coordinate transformations in $D = 1$. The general algorithm is as follows

1. Select a family of basis functions $\phi_j(\xi)$ defined on $[0, 1]$ that are easy to integrate (e.g., the indefinite integral is known).

2. Find suitable shift and scale for a series $s\left(\sum_j \phi_j(\xi)c_j\right) + c_0$ to be a valid probability density function $p(\xi)$ for all c_j.

3. Use cumulative distribution function (indefinite integral of $p(\xi)$) as a coordinate transformation.

When $D = 1$ mapping is available, it is possible to lift it to $D > 1$ by transfinite interpolation (Gordon & Hall, 1973). For example, for $D = 2$ the transformation becomes

$$\begin{aligned} x^1(\xi^1, \xi^2) &= y_1(\xi^1)(1 - \xi^2) + y_2(\xi^1)\xi^2, \\ x^2(\xi^1, \xi^2) &= y_3(\xi^2)(1 - \xi^1) + y_4(\xi^2)\xi^1, \end{aligned} \tag{8}$$

where y_i, $i = 1, \ldots, 4$ are $D = 1$ mappings, e.g., given in (5). The extension of (8) to higher dimensions is straightforward.

Note that (8) has a "low-rank" structure that decreases the diversity of possible grids. The issue can be alleviated with Hermite transfinite interpolation or with more general blending functions (Liseikin, 2017). Other more computationally involved remedies are variational and elliptic (Laplace-Beltrami) grid generators (Steinberg & Roache, 1986), (Spekreijse, 1995).

3.2. Linear PDEs under coordinate transformations

Here we remind how the most widely used differential operators change under the transformation (7).

The results we present in this section are standard (Liseikin, 2017), (Eglit et al., 1996), (Simmonds, 1994). For convenience, the proofs are also available in Appendix A.

For convenience, we define the Jacobi matrix, and its determinant

$$\mathcal{J}_{i\alpha} \equiv \frac{\partial x^i}{\partial \xi^\alpha}, \; J \equiv \det \mathcal{J}. \tag{9}$$

3

Note that for the mapping (5) determinant J vanishes nowhere in the domain due to the strict monotonicity of the mapping.

It is not hard to show that for arbitrary space-dependent fields c^j, ϕ, a^{kj}, $k, j \in 1, \ldots, D$, the following transformation laws hold

$$c^j \frac{\partial \phi}{\partial x^j} = c^j \frac{\partial \xi^\alpha}{\partial x^j} \frac{\partial \phi}{\partial \xi^\alpha},$$

$$a^{kj} \frac{\partial^2 \phi}{\partial x^j \partial x^k} = a^{kj} \frac{\partial \xi^\beta}{\partial x^j} \frac{\partial \xi^\gamma}{\partial x^k} \frac{\partial^2 \phi}{\partial \xi^\beta \partial \xi^\gamma} + a^{kj} \frac{\partial^2 \xi^\gamma}{\partial x^k \partial x^j} \frac{\partial \phi}{\partial \xi^\gamma},$$

$$\frac{\partial}{\partial x^\alpha} (c^\alpha \phi) = \frac{1}{J} \frac{\partial}{\partial \xi^k} \left(J c^\alpha \frac{\partial \xi^k}{\partial x^\alpha} \phi \right),$$

$$\frac{\partial}{\partial x^k} \left(a^{kj} \frac{\partial \phi}{\partial x^j} \right) = \frac{1}{J} \frac{\partial}{\partial \xi^k} \left(J \left(a^{\alpha j} \frac{\partial \xi^k}{\partial x^\alpha} \frac{\partial \xi^\beta}{\partial x^j} \right) \frac{\partial \phi}{\partial \xi^\beta} \right). \tag{10}$$

Results (10) allow deriving transformation laws for many practically-relevant PDEs. We are interested in the following ones:

1. Stationary diffusion equation

$$\frac{\partial}{\partial x^k} \left(a^{kj}(\boldsymbol{x}) \frac{\partial}{\partial x^j} u(\boldsymbol{x}) \right) = f(\boldsymbol{x}) \tag{11}$$
$$\boldsymbol{x} \in \Gamma \equiv [0,1]^D, \ u(\boldsymbol{x})|_{x \in \partial\Gamma} = 0,$$

where $\partial\Gamma$ is a boundary of the unit hypercube Γ, and $\boldsymbol{x} \in \mathbb{R}^D$.

2. Convection-diffusion equation

$$\frac{\partial}{\partial t} \phi(\boldsymbol{x}, t) + \frac{\partial}{\partial x^i} \left(v^i(\boldsymbol{x}) \phi(\boldsymbol{x}, t) \right) =$$
$$\frac{\partial}{\partial x^k} \left(a^{kj}(\boldsymbol{x}) \frac{\partial}{\partial x^j} \phi(\boldsymbol{x}, t) \right),$$
$$\boldsymbol{x} \in \Gamma \equiv [0,1]^D, \ \phi(\boldsymbol{x}, t)|_{x \in \partial\Gamma} = 0, \ \phi(\boldsymbol{x}, 0) = f(\boldsymbol{x}). \tag{12}$$

3. Two-way wave equation

$$\frac{\partial^2 \rho(\boldsymbol{x}, t)}{\partial t^2} + v^i(\boldsymbol{x}) \frac{\partial \rho(\boldsymbol{x}, t)}{\partial x^i} = c^{ij}(\boldsymbol{x}) \frac{\partial^2 \rho(\boldsymbol{x}, t)}{\partial x^i \partial x^j}$$
$$+ e(\boldsymbol{x}) \rho(\boldsymbol{x}, t),$$
$$\boldsymbol{x} \in \Gamma \equiv [0,1]^D, \ \rho(\boldsymbol{x}, t)|_{x \in \partial\Gamma} = 0, \ \rho(\boldsymbol{x}, 0) = f(\boldsymbol{x}). \tag{13}$$

For all of the equations above, the transformed form easily follows from (10). However, for wave and convection-diffusion equations, additional steps are required to ensure that the transformed equation has the same parametric form as the original one. Table 1 contains the results for selected PDEs.

Results summarized in Table 1 along with the coordinate transformations described in Section 3.1 are sufficient to perform general covariance augmentation for equations (11), (12), (13).

3.3. Navier-Stokes equation

To show that our scheme applies to nonlinear PDEs and more general boundary conditions, we consider lid-driven cavity flow of incompressible fluid with deformed cavities. The system of equations in the physical space reads

$$\frac{\partial v^i}{\partial t} = \frac{\partial}{\partial x^k} \left(-v^k v^i - p + \nu \frac{\partial v^i}{\partial x^k} \right), \ \frac{\partial v^k}{\partial x^k} = 0,$$

$$p(t, x)|_{x \in L} = 0, \ \left. \frac{\partial p(t, x)}{\partial x^i} n^i \right|_{x \in \Gamma} = 0,$$

$$v^1(t, x)|_{x \in L} = 1, \ v^2(t, x)|_{x \in C} = v^1(t, x)|_{x \in \Gamma} = 0, \tag{14}$$

where x belong to the interior of the curve $C(x^1, x^2) = L(x^1, x^2) \cup \Gamma(x^1, x^2)$, $L(x^1, x^2)$ represents the lid and $\Gamma(x^1, x^2)$ — the rest of the cavity's boundary, $t \in [0, T]$.

For Equation (14) we explicitly specify the form of cavity $x^i(\xi^1, \xi^2)$, $i = 1, 2$ using curvilinear coordinates $\xi^1, \xi^2 \in [0, 1]^2$ (see Section 4 for the description of the cavities used). When solution at $t = T$ is obtained, the only parameter of the PDE is the geometry itself which is fully specified by $x^i(\xi^1, \xi^2)$, $i = 1, 2$.

In these circumstances general covariance augmentation simplifies. Namely, we use random coordinate transformations Equation (8) to form additional mapping $\xi^i(\widetilde{\xi^i})$ and reinterpolate obtained solution $v^i(\xi^1, \xi^2, T)$ on the grid $\widetilde{\xi^i}$, $i = 1, 2$.

Although transformation low of the equation Navier-Stokes equation Equation (14) is not directly used for augmentation, we still need it to obtain a solution in the computational domain ξ^i, $i = 1, 2$ (see Appendix E for details).

In Section 4, we show the empirical performance of the proposed augmentation scheme for both linear and nonlinear PDEs.

4. Experiments

Here we present an empirical evaluation of augmentation by general covariance. For that purpose, we design several experiments in $D = 1$ and $D = 2$. We start with a description of the shared experiments' setup.

Table 1. Transformation of PDEs parameters under the change of coordinates $\boldsymbol{x} \to \boldsymbol{x}(\boldsymbol{\xi})$.

Equation	Fields	Transformed fields
Stationary diffusion (11)	$u(\boldsymbol{x})$	$u(\boldsymbol{x}(\boldsymbol{\xi}))$
	$a^{k\beta}(\boldsymbol{x})$	$Ja^{\alpha j}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial \xi^k}{\partial x^\alpha}\frac{\partial \xi^\beta}{\partial x^j}$
	$f(\boldsymbol{x})$	$Jf(\boldsymbol{x}(\boldsymbol{\xi}))$
Convection-diffusion (12)	$\phi(\boldsymbol{x},t)$	$J\phi(\boldsymbol{x}(\boldsymbol{\xi}),t)$
	$a^{k\beta}(\boldsymbol{x})$	$a^{\alpha j}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial \xi^k}{\partial x^\alpha}\frac{\partial \xi^\beta}{\partial x^j}$
	$v^i(\boldsymbol{x})$	$v^k(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial \xi^i}{\partial x^k} + a^{\alpha j}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial \xi^k}{\partial x^\alpha}\frac{\partial \xi^\beta}{\partial x^j}\frac{\partial \xi^\gamma}{\partial x^\rho}\frac{\partial^2 x^\rho}{\partial \xi^\gamma \partial \xi^\beta}$
	$f(\boldsymbol{x})$	$Jf(\boldsymbol{x}(\boldsymbol{\xi}))$
Wave (13)	$\rho(\boldsymbol{x},t)$	$\rho(\boldsymbol{x}(\boldsymbol{\xi}),t)$
	$c^{\gamma\beta}(\boldsymbol{x})$	$c^{kj}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial \xi^\gamma}{\partial x^k}\frac{\partial \xi^\beta}{\partial x^j}$
	$v^\alpha(\boldsymbol{x})$	$v^i(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial \xi^\alpha}{\partial x^i} - c^{kj}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial^2 \xi^\alpha}{\partial x^k \partial x^j}$
	$f(\boldsymbol{x})$	$f(\boldsymbol{x}(\boldsymbol{\xi}))$
	$e(\boldsymbol{x})$	$e(\boldsymbol{x}(\boldsymbol{\xi}))$

4.1. Setup

4.1.1. NEURAL NETWORKS

As a rule, neural PDE solvers are either Neural Operators or classical architectures used for image processing.[2] Since our approach is architecture-agnostic, we include results for both types of neural networks.

On the side of neural operators, we include original versions of DeepONet (Lu et al., 2021a) and FNO (Li et al., 2020), implemented in `https://github.com/lu-group/deeponet-fno` and `https://github.com/neural-operator/fourier_neural_operator`, respectively. Besides, for the $D = 1$ case, we also implemented an FNO-like operator dubbed rFNO with FFT replaced by pure real transform based on a complete trigonometric family and trapezoidal rule. In addition, for $D = 2$ we also use Spectral Neural Operator described in (Fanaskov & Oseledets, 2022). Roughly speaking, the architecture has the same structure as FNO but with Discrete Cosine Transform in place of FFT.

Classical machine-learning architectures include DilResNet (Yu & Koltun, 2015), (Stachenfeld et al., 2021), U-Net (Ronneberger et al., 2015), and MLP (Haykin, 1994).

A detailed description of neural networks is available in Appendix C.

---

[2]There are also hybrid methods (e.g., (Bar-Sinai et al., 2019)), but we do not consider them here.

4.1.2. PARTIAL DIFFERENTIAL EQUATIONS AND DATASETS

We evaluate augmentation on stationary diffusion (11), convection-diffusion (12), and wave (13) equations.

To produce PDE data, for linear PDEs, we sampled all needed functions from random trigonometric series

$$f(x) = \sum_{k=0}^{N-1} \left(c_k \cos(2\pi k x) + s_k \sin(2\pi k x)\right), \quad (15)$$

with c_k sampled from the standard normal distribution and scaled/shifted appropriately to ensure needed boundary conditions or make $f(x)$ uniformly positive. For $D = 2$, the procedure is the same, but a direct product of one-dimensional bases is used.

Afterward, equations with randomly generated data are discretized either with finite-difference or finite-element methods.

For $D = 1$, we generated one dataset per equation and an additional dataset for the wave equation. Results for the extra wave dataset are available in Appendix D. For $D = 2$, we produced two datasets per equation that differ by complexity (more rough targets or more diverse feature-target pairs).

Also, for the purposes explained later, we use two distinct elliptic datasets in $D = 2$. In the first one, named "Elliptic alpha", diffusion coefficients $a^{i\beta}$ form a symmetric positive definite matrix for each point of the domain. In the second one, named "Elliptic beta", the matrix $a^{i\beta}$ is the identity

matrix multiplied by a single uniformly positive diffusion coefficient.

For Navier-Stokes equation Equation (14) we generate cavities defined with a transfinite interpolation Equation (8) from randomly generated boundary curves

$$
\begin{aligned}
& x^1(\xi^1, 0) = \xi^1, \ x^1(\xi^1, 1) = \xi^1, \\
& x^2(0, \xi^2) = \xi^2, \ x^2(1, \xi^2) = \xi^2, \ x^2(\xi^1, 1) = 1, \\
& x^1(0, \xi^2) = \sum_{i=1}^{m} \sin(\pi \xi^2 (k+1) c^k / (10(k+1)^2)), \\
& x^1(1, \xi^2) = 1 + \xi^2 (1 - \xi^2) \alpha / 2, \\
& x^2(\xi^1, 0) = \sum_{i=1}^{m} \sin(\pi \xi^1 (k+1) d^k / (10(k+1)^2)),
\end{aligned}
\tag{16}
$$

where d^k, c^k, α are sampled from standard normal distribution and $\xi^1, \xi^2 \in [0,1]^2$. We also use $T = 10^{-4}$ and $\nu = 10^{-2}$.

More details on the dataset generation process are available in Appendix D. Links to the datasets are available in the repository `https://github.com/VLSF/augmentation`.

4.1.3. COORDINATE TRANSFORMATIONS

To generate coordinate transformation for $D = 1$, we use a cumulative distribution function constructed from unnormalize probability density

$$
p(x) = \beta + \sum_{i=1}^{N} \left(c_i \cos(2n\pi x + p_i) \right), \tag{17}
$$

where c_i and p_i are samples from the standard normal distribution, and $N = 5, \beta = 1$.

For $D = 2$ we use four $D = 1$ coordinate transformations constructed with (5), where coefficients $i = 1, \ldots, 5$ are from standards normal distribution, $c_0 = 1$, $N = 6$, and $\beta = 10^{-5}$. To obtain $D = 2$ mappings from the four unidimensional, we apply transformation using transfinite interpolation (8).

4.1.4. METRICS

As a main measure of performance, we use average relative L_2 test error

$$
E_{\text{test}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\|\mathcal{N}(f_i) - t_i\|_2}{\|t_i\|_2}, \tag{18}
$$

where $\mathcal{N}$ is a neural network, f_i and t_i are features and targets from the test set.

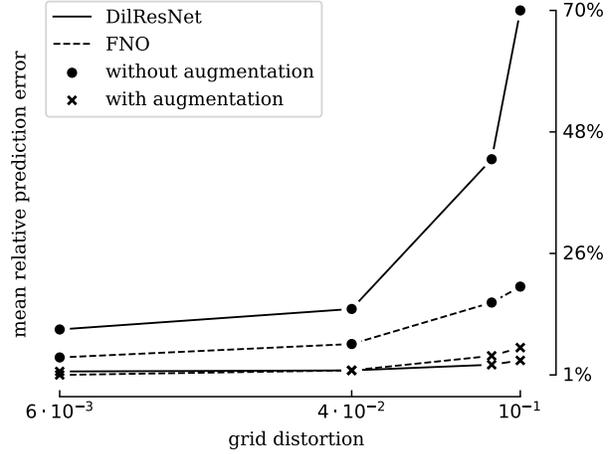To evaluate the impact of augmentation, we consider the

Figure 2. Sensitivity to grid distortion for DilResNet and FNO with and without augmentation. The distortion here refers to the maximal difference between the unperturbed x and perturbed $x(\xi)$ grids averaged over 1000 grids used to augment dataset.

relative gain

$$
g = \left(1 - \frac{E_{\text{test}}^{\text{aug}}}{E_{\text{test}}} \right) \times 100\%, \tag{19}
$$

where $E_{\text{test}}^{\text{aug}}$, and $E_{\text{test}} > 0$ are relative errors of neural network trained with and without augmentation respectively. Since $E_{\text{test}}^{\text{aug}} = (1 - g) E_{\text{test}}$, the larger g the better.

4.2. Sensitivity to grid distortions

Before training of augmented dataset, it is instructive to evaluate the network trained without augmentation on the augmented train set. This way, we can estimate the degree of equivariance current neural networks have.

More specifically, for this experiment, we take a dataset for stationary-diffusion equation (11) (Elliptic alpha), and train DilResNet and FNO. After that, we generate a set of augmented datasets using increasingly distorted grids and evaluate neural networks on them. Results are reported in Figure 2.

Table 2. Relative gain for averaging with respect to equations and networks. Augmentation factor m means that additional mN_{train} augmented samples are appended to the training dataset.

$m \backslash N_{\text{train}}$	500	1000	1500	2000
1	11%	18%	21%	21%
2	16%	25%	27%	31%
3	15%	23%	28%	28%
4	19%	25%	28%	30%

As we can see, if distortion is comparable with 10^{-2}, which is a spacing of the original grid, the neural network can handle the modified dataset quite well. However, the further distortion increase leads to substantial performance deterioration. So for distortion of about 10 original grid spacing, the trained network is unusable. Networks trained with augmentation retain good relative error even on distorted grids.

Interestingly, FNO can handle distortion slightly better — four-fold increase against seven-fold for DilResNet. Qualitatively, FNO starts with 4% and ends with 18%, whereas DilResNet starts with 10% and ends with 70%

We stress that for equations other than elliptic the equivariance is not observed (see results in Table 6). Besides, for the elliptic equation, equivariance is confirmed only for the specific distortions of the grid. It is not obvious that the same result holds for other grid transformations, so we do not claim that we achieved general covariance.

4.3. Statistical study for $D = 1$ problems

Given that neural networks fail to produce correct predictions for augmented dataset, the next natural step is to introduce augmented samples on the training stage. Here we describe relevant experiments for $D = 1$ datasets.

In this section we use m to denote augmentation factor. If original train set consists of N_{train} points, after augmentation with factor m the modified train set has $(1+m)N_{\text{train}}$ points.

For each equation we consider the following parameters: $N_{\text{train}} = 500, 1000, 1500, 2000$; $N_{\text{test}} = 1000$; augmentation factor $m = 1, 2, 3, 4$. For each set of parameters we perform five runs with different seeds controlling network initialization and random grids generated for the augmentation. In Appendix F one can find relative test errors averaged with respect to these five runs. Here we present only aggregated results.

First, Table 3 contain results averaged with respect to sizes of train set and augmentation factors. We can see that gain is positive on average for all networks and equations. Note, however, that we observe negative gain, i.e., the augmentation fail to improve test error. This occurres mainly for weak models such as DeepONet and MLP that are othen fail to achieve reasonable test error.

Table 3. Relative gain for averaging wrt different training scenarios.

	DeepONet	FNO	DilResNet	rFNO	MLP
elliptic	16%	22%	28%	19%	8%
conv-dif	32%	36%	22%	24%	28%
wave	14%	36%	22%	18%	16%

Second, Table 2 contain gains averaged over equations and networks. Generally, we observe that augmentation is more helpful for larger datasets. Our best current explanation is that our augmentation procedure is poorly-calibrated, i.e., the augmented data is completely out of distribution. If this is the case, the increase of the train set may improve the overlap between the train data distribution and augmented data distribution.

4.4. Augmentation for $D = 2$ problems

We report results for a single run for two-dimensional problems in Table 4.

One can see that, augmentation reduces relative test error for all cases when the network can generalize. It also helps DeepONet to reach test errors smaller than one for the Elliptic alpha and Elliptic beta datasets.

The most intriguing part are the results for the stationary diffusion equation. As we explained before, in $D = 2$ we consider two datasets for the elliptic equation: Elliptic alpha and Elliptic beta. Elliptic alpha has a complete set of distinct diffusion coefficients that form a symmetric positive definite matrix.

Contrary to that, Elliptic beta has a single positive diffusion coefficient, so the diffusion matrix a^{ij} in (11) is proportional to the diagonal matrix. On the other hand, after the augmentation, i.e., for the transformed equation, the Elliptic beta train set has nonzero off-diagonal contributions to the diffusion matrix (see Table 1). At the same time, we still have a diffusion matrix proportional to the diagonal for the test set. Despite this discrepancy, augmentation still improves the test error.

This result suggests that it can be beneficial to embed a given family of equations into a larger parametric family and perform augmentation for that extended set.

For the Navier-Stokes equation, we can also see that augmentation improves test error for both components of speed v^1, v^2. This provides evidence that the method is applicable without difficulties to complex geometries. We also tried to train DeepONet, but failed to obtain relative errors comparable to other networks, additional results are available in Appendix F.

5. Related research

We know of two articles directly related to the augmentation techniques for neural PDE solvers (Brandstetter et al., 2022), (Li et al., 2022a).

The article (Li et al., 2022a) is secondary since it is only a minor extension of (Brandstetter et al., 2022). We do not discuss it further.

Equation	Model	simple datasets			complex datasets		
		$\times$	$\checkmark$	g	$\times$	$\checkmark$	g
Convection-diffusion	FNO	0.067	0.048	28%	0.510	0.418	18%
	DeepONet	0.675	0.567	16%	—	—	
	DilResNet	0.023	0.010	56%	0.312	0.225	28%
	MLP	0.094	0.050	49%	0.566	0.496	12%
	U-Net	0.069	0.031	55%	0.419	0.364	13%
	SNO	0.086	0.066	23%	0.416	0.373	10%
Elliptic alpha	FNO	0.066	0.036	46%	0.306	0.207	32%
	DeepONet	—	0.826		—	—	
	DilResNet	0.105	0.021	80%	0.160	0.133	17%
	MLP	0.088	0.053	40%	0.322	0.253	21%
	U-Net	0.093	0.070	25%	0.386	0.194	50%
	SNO	0.082	0.050	39%	0.251	0.209	17%
Elliptic beta	FNO	0.034	0.021	38%	0.181	0.126	30%
	DeepONet	—	0.832		—	0.946	
	DilResNet	0.099	0.022	78%	0.089	0.062	30%
	MLP	0.069	0.035	50%	0.238	0.138	42%
	U-Net	0.070	0.067	4%	0.170	0.143	16%
	SNO	0.068	0.038	44%	0.187	0.144	23%
Wave	FNO	0.200	0.159	21%	0.650	0.628	3%
	DeepONet	—	—		—	—	
	DilResNet	0.053	0.048	9%	0.43	0.38	12%
	MLP	0.313	0.295	6%	—	0.99	
	U-Net	—	—		0.57	0.52	9%
	SNO	0.37	0.37	0%	—	—	
		v^1			v^2		
Navier-Stokes	FNO	0.005	0.003	40%	0.022	0.010	55%
	UNet	0.019	0.09	53%	0.069	0.037	46%
	DilResNet	0.021	0.015	29%	0.073	0.045	38%
	MLP	0.082	0.066	38%	0.082	0.066	20%
	SNO	0.004	0.003	25%	0.013	0.008	38%

Table 4. Relative test errors and gain for $D = 2$ datasets. Symbols $\checkmark$ and $\times$ mark results with and without augmentation respectively. We put — when network fails to reach test error below 1.0.

In (Brandstetter et al., 2022), authors consider Lie point symmetries. Namely, to perform augmentation, they use smooth transformations that preserve a solution set of PDE (map a given solution to the other one) and form a group with a structure of the continuous manifold (Lie group). As the authors explain, Lie point symmetries, in a certain sense, provide an exhaustive set of possible transformations suitable for augmentation. Given that, it is appropriate to highlight what distinguishes our research from (Brandstetter et al., 2022).

In (Brandstetter et al., 2022) symmetries of a *fixed PDE* are used. In place of that, we consider mappings that leave us within *a particular family of PDEs*. Such transformations are more abundant, easier to find, and more suitable for physical systems with spatiotemporal dependencies, e.g., Maxwell equations in macroscopic media (Jackson, 1975), wave propagation in non-homogeneous media (Brekhovskikh, 1980), and fluid flow through porous media (Alt & Di Benedetto, 1985), e.t.c. In particular, the local distortion of coordinates is a safe choice for a large set of PDEs because of the general covariance principle (Post, 1997).

Among other contributions indirectly related to our approach, we can mention articles on deep learning for PDE that deal with complex geometries (Gao et al., 2021), (Li et al., 2022b). The end of this line of research is to generalize physics-informed neural networks and neural operators from rectangular to more general domains. These works

also contain particular equations in a transformed form, but for an entirely different reason.

Similar coordinate transformations are abundant in classical scientific computing (Liseikin, 2017), (Knupp & Steinberg, 1994). On the one hand, the grid defines the geometry of the domain. On the other hand, the grid is refined (h-refinement (Li & Bettess, 1997), (Baker, 1997)) or transported (r-refinement (Baker, 1997)) to improve accuracy (e.g., equidistribution principle (Chen, 1994)). One can implement both approaches using partial differential equations (elliptic and hyperbolic equations) or analytic mappings (algebraic methods (Smith, 1982), (Gordon & Hall, 1973)). In the present research, we gravitate toward the latter because it is computationally cheap, and derivatives are readily available.

From the broader perspective, the entire subfield of geometric machine learning (Bronstein et al., 2021) deals with related issues. Typically, the quest is to design a neural network for which the invariance or equivariance holds for a chosen set of transformations. The most relevant works of this sort are (Cheng et al., 2019), (Weiler et al., 2021), (Wang et al., 2020). In the first two articles, the authors develop gauge-invariant convolutions (general covariance), but PDEs are not in question, and the generalization to neural operators is not yet available. In the third article, the authors design neural networks that respect selected symmetries of the Navier-Stokes equation.

We want to point out that invariance and equivariance principles are often of no use for PDE problems. First, transformations of the physical fields can fail to be covariant or contravariant, as shown by the example of convection-diffusion equation Table 1. Second, the symmetries are often not apparent when PDE in question has spatial dependence or is defined in complex geometry. For example, lid-driven cavity flow Equation (14) breaks all symmetries of the Navier-Stokes equation listed in (Wang et al., 2020).

6. Conclusion and further research

We demonstrated how to construct augmentation based on general covariance. The essence of the approach is the observation that it is possible to use change of coordinates to produce novel solutions from the old ones. This is possible because physical phenomena do not depend on the choice of coordinates, so in the new coordinate system, the type of the equation persists, but parameters change. These new parameters along with the solution in new coordinate system can be used as additional train samples.

The proposed augmentation systematically improves test error for all considered architectures. Besides that, it is architecture-agnostic and generalizes well on other equations, especially defined in complex geometries suitable for body-fitted meshes.

A lot of improvements to the proposed approach are possible. The list below sums up a few possibilities:

1. *More complex structured grids.* As we showed in the Navier-Stokes example, it is straightforward to extend our approach to situations where there is an intermediate mapping from the physical domain to the computational domain. The considered example is elementary, so it is desirable to test augmentation on more challenging problems. It is also interesting to consider our augmentation approach with the architectures that already contain mapping as part of the network, e.g., (Gao et al., 2021), (Li et al., 2022b).

2. *Adaptive augmentation.* In the present research, we generate random grids to perform augmentation. It should be more advantageous to actively select grids based on the neural network performance.

3. *Unstructured grids.* Our approach, as described here, is, in principle, applicable to the unstructured grids. The main problem is that it is not obvious how to construct a mapping and generate a deformed grid such that it is still acceptable from the computational perspective.

4. *Covariant neural operators.* It would be interesting to adapt or generalize results from (Weiler et al., 2021), (Wang et al., 2020) to construct covariant neural operators. Less ambitious, it is possible to improve the training protocols for neural networks using the Jacobi matrix and determinant as input features. This way, one escapes the need to introduce higher derivatives in transformed equations Table 1.

5. *Transformations between parametric families.* A very interesting feature that we observe is that augmentation still helps even when it is performed for a larger parametric family of equations than needed. One possible extension is to dispense with coordinate covariance and consider more general transformations that map one family of PDEs to another family.

6. *Time-dependent coordinate transformations.* We only consider spatial transformations. It is possible to use time-dependent transformations. For example, one can construct a grid, that is deformed at $t = 0$ and approaches its non-deformed state when t increases.

7. Acknowledgements

References

Alt, H. W. and Di Benedetto, E. Nonsteady flow of water and oil through inhomogeneous porous media. *Annali della Scuola Normale Superiore di Pisa-Classe di Scienze*, 12 (3):335–392, 1985.

Baker, T. J. Mesh adaptation strategies for problems in fluid dynamics. *Finite Elements in Analysis and Design*, 25 (3-4):243–273, 1997.

Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

Brandstetter, J., Welling, M., and Worrall, D. E. Lie point symmetry data augmentation for neural pde solvers. *arXiv preprint arXiv:2202.07643*, 2022.

Brekhovskikh, L. M. *Waves in layered media*. Applied Mathematics and Mechanics, Vol. 16. Academic Press, Inc. [Harcourt Brace Jovanovich, Publishers], New York-London, 1980. ISBN 0-12-130560-0. Translated from the second Russian edition by Robert T. Beyer.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Chen, K. Error equidistribution and mesh adaptation. *SIAM Journal on Scientific Computing*, 15(4):798–818, 1994.

Cheng, M. C., Anagiannis, V., Weiler, M., de Haan, P., Cohen, T. S., and Welling, M. Covariance in physics and convolutional neural networks. *arXiv preprint arXiv:1906.02481*, 2019.

Eglit, M. E., Golubiatnikov, A. N., Kamenjarzh, J. A., Karlikov, V. P., Kulikovsky, A. G., Petrov, A. G., Shikina, I. S., and Sveshnikova, E. I. *Continuum mechanics via problems and exercises. Part I*, volume 19 of *World Scientific Series on Nonlinear Science. Series A: Monographs and Treatises*. World Scientific Publishing Co., Inc., River Edge, NJ, 1996. ISBN 981-02-2962-3. doi: 10.1142/3010. URL https://doi.org/10.1142/3010. Theory and problems, Translated from the Russian by A. N. Tiatiushkin, Translation edited by Eglit and Dewey H. Hodges.

Emam, M. H. *Covariant Physics: From Classical Mechanics to General Relativity and Beyond*. Oxford University Press, 2021.

Evans, L. C. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

Fanaskov, V. and Oseledets, I. Spectral neural operators. *arXiv preprint arXiv:2205.10573*, 2022.

Gao, H., Sun, L., and Wang, J.-X. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428: 110079, 2021.

Golberg, M. A. The derivative of a determinant. *Amer. Math. Monthly*, 79:1124–1126, 1972. ISSN 0002-9890. doi: 10.2307/2317435. URL https://doi.org/10.2307/2317435.

Gordon, W. J. and Hall, C. A. Construction of curvilinear co-ordinate systems and applications to mesh generation. *International Journal for Numerical Methods in Engineering*, 7(4):461–477, 1973.

Haykin, S. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

Hennigh, O. Lat-net: compressing lattice boltzmann flow simulations using deep neural networks. *arXiv preprint arXiv:1705.09036*, 2017.

Jackson, J. D. *Classical electrodynamics*. John Wiley & Sons, Inc., New York-London-Sydney, second edition, 1975.

Karnakov, P., Litvinov, S., and Koumoutsakos, P. Optimizing a discrete loss (odil) to solve forward and inverse problems for partial differential equations using machine learning tools. *arXiv preprint arXiv:2205.04611*, 2022.

Knupp, P. and Steinberg, S. *Fundamentals of grid generation*. CRC Press, Boca Raton, FL, 1994. ISBN 0-8493-8987-9. With 1 IBM-PC floppy disk (3.5 inch; HD).

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.

Li, L.-y. and Bettess, P. Adaptive Finite Element Methods: A Review. *Applied Mechanics Reviews*, 50(10):581–591, 10 1997. ISSN 0003-6900. doi: 10.1115/1.3101670. URL https://doi.org/10.1115/1.3101670.

Li, Y., Pang, Y., and Shan, B. Physics-guided data augmentation for learning the solution operator of linear differential equations. *arXiv preprint arXiv:2212.04100*, 2022a.

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*, 2020.

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., Huang, D. Z., Liu, B., and Anandkumar, A. Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, 2022b.

Liseikin, V. D. *Grid generation methods*. Scientific Computation. Springer, Cham, third edition, 2017. ISBN 978-3-319-57845-3; 978-3-319-57846-0. doi: 10.1007/978-3-319-57846-0. URL https://doi.org/10.1007/978-3-319-57846-0.

Lu, L., Jin, P., and Karniadakis, G. E. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021a.

Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021b.

Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K., et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

Post, E. J. *Formal structure of electromagnetics: general covariance and electromagnetics*. Courier Corporation, 1997.

Rippel, O., Snoek, J., and Adams, R. P. Spectral representations for convolutional neural networks. *Advances in neural information processing systems*, 28, 2015.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Shorten, C. and Khoshgoftaar, T. M. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

Simmonds, J. G. *A brief on tensor analysis*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, second edition, 1994. ISBN 0-387-94088-X. doi: 10.1007/

978-1-4419-8522-4. URL https://doi.org/10.1007/978-1-4419-8522-4.

Smith, R. E. Algebraic grid generation. *Applied Mathematics and Computation*, 10:137–170, 1982.

Spekreijse, S. P. Elliptic grid generation based on laplace equations and algebraic transformations. *Journal of Computational Physics*, 118(1):38–61, 1995.

Stachenfeld, K., Fielding, D. B., Kochkov, D., Cranmer, M., Pfaff, T., Godwin, J., Cui, C., Ho, S., Battaglia, P., and Sanchez-Gonzalez, A. Learned coarse models for efficient turbulence simulation. *arXiv preprint arXiv:2112.15275*, 2021.

Steinberg, S. and Roache, P. J. Variational grid generation. *Numerical Methods for Partial Differential Equations*, 2 (1):71–96, 1986.

Tripura, T. and Chakraborty, S. Wavelet neural operator: a neural operator for parametric partial differential equations. *arXiv preprint arXiv:2205.02191*, 2022.

Wang, R., Walters, R., and Yu, R. Incorporating symmetry into deep dynamics models for improved generalization. *arXiv preprint arXiv:2002.03061*, 2020.

Wang, S., Wang, H., and Perdikaris, P. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40): eabi8605, 2021.

Weiler, M., Forré, P., Verlinde, E., and Welling, M. Coordinate independent convolutional networks–isometry and gauge equivariant convolutions on riemannian manifolds. *arXiv preprint arXiv:2106.06020*, 2021.

Wen, Q., Sun, L., Yang, F., Song, X., Gao, J., Wang, X., and Xu, H. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

Yu, F. and Koltun, V. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

A. Coordinate transformations

In this appendix we collect standard material related to coordinate transformations and transformation laws for differential operators.

The material of this section is available in many sources (Liseikin, 2017), (Eglit et al., 1996), (Simmonds, 1994) and presented here merely for convenience of the reader.

Everywhere in this section the Einstein's summation notation is used, e.g., $a_\alpha b^\alpha \equiv \sum_\alpha a_\alpha b^\alpha$, and as coordinate transformations we consider (7).

A.1. Some relations for the first and the second derivatives

In this section we explain several identities we find useful for expressing equations in the covariant form. The main problem is that when we have a mapping $\boldsymbol{x}(\boldsymbol{\xi})$ (defined below) given in some explicit form and have no closed-form expression for the inverse mapping, it is inconvenient to use derivatives with respect to $\boldsymbol{x}$. Since such derivatives appear in plenitude in PDEs after coordinate transformation, our chief goal is to find relations that allows us to rewrite them using derivatives with respect to $\boldsymbol{\xi}$.

We use Jacobi's formula for differentiable matrix-valued function $\boldsymbol{A}(t)$ without a proof:

$$\frac{d}{dt} \det \boldsymbol{A}(t) = \det \boldsymbol{A}(t) \, \text{tr} \left(\boldsymbol{A}^{-1}(t) \frac{d\boldsymbol{A}(t)}{dt} \right). \tag{20}$$

The short note (Golberg, 1972) contains a concise derivation.

We start with the relation between first derivatives

$$\frac{\partial x^i}{\partial \xi^\alpha} \frac{\partial \xi^\alpha}{\partial x^j} = \delta_{ij} = \begin{cases} 1, \, i = j; \\ 0, \, i \neq j, \end{cases} \tag{21}$$

which follows from chain rule applied to the function $\boldsymbol{x}(\boldsymbol{\xi}(\boldsymbol{x})) = \boldsymbol{x}$:

$$\delta_{ij} = \frac{\partial x^i}{\partial x^j} = \frac{\partial x^i(\xi^1(x^1, \ldots, x^D), \ldots, \xi^D(x^1, \ldots, x^D))}{\partial x^j} = \frac{\partial x^i}{\partial \xi^\alpha} \frac{\partial \xi^\alpha}{\partial x^j}. \tag{22}$$

It is also convenient to rewrite identity Equation (21) in a matrix form

$$\mathcal{J}_{i\alpha} \equiv \frac{\partial x^i}{\partial \xi^\alpha}, \, \boldsymbol{\mathcal{J}} \boldsymbol{\mathcal{J}}^{-1} = \boldsymbol{I}, \tag{23}$$

where $\boldsymbol{\mathcal{J}}$ is Jacobi matrix and $\boldsymbol{I}$ is the identity matrix.

Next identity we consider is the equation for the derivative of $J = \det \boldsymbol{\mathcal{J}}$, i.e., the determinant of Jacobi matrix:

$$\frac{\partial}{\partial \xi^k} J = J \frac{\partial \xi^m}{\partial x^i} \frac{\partial^2 x^i}{\partial \xi^m \partial \xi^k}. \tag{24}$$

The equation immediately follows from Jacobi's equation (20) and the definition of inverse (21), (23) for Jacobi matrix.

Another relation useful in derivations reads

$$\frac{\partial^2 \xi^\alpha}{\partial x^i \partial x^j} = -\frac{\partial \xi^\gamma}{\partial x^i} \frac{\partial \xi^\theta}{\partial x^j} \frac{\partial \xi^\alpha}{\partial x^k} \frac{\partial^2 x^k}{\partial \xi^\gamma \partial \xi^\theta}. \tag{25}$$

To prove the relation, we find a derivative of (21) as follows

$$0 = \frac{\partial}{\partial x^k} \left(\frac{\partial x^i}{\partial \xi^\alpha} \frac{\partial \xi^\alpha}{\partial x^j} \right) = \frac{\partial}{\partial x^k} \left(\frac{\partial x^i}{\partial \xi^\alpha} \right) \frac{\partial \xi^\alpha}{\partial x^j} + \frac{\partial x^i}{\partial \xi^\alpha} \frac{\partial^2 \xi^\alpha}{\partial x^j \partial x^k} = \frac{\partial \xi^\beta}{\partial x^k} \frac{\partial^2 x^i}{\partial \xi^\alpha \partial \xi^\beta} \frac{\partial \xi^\alpha}{\partial x^j} + \frac{\partial x^i}{\partial \xi^\alpha} \frac{\partial^2 \xi^\alpha}{\partial x^j \partial x^k}, \tag{26}$$

and multiply by the inverse Jacobi matrix.

The last identity we need reads

$$\frac{1}{J}\frac{\partial}{\partial \xi^j}\left(J\frac{\partial \xi^j}{\partial x^i}\right) = 0. \tag{27}$$

To prove (27) we apply (24) (Jacobi formula) and obtain

$$\frac{1}{J}\frac{\partial}{\partial \xi^\alpha}\left(J\frac{\partial \xi^\alpha}{\partial x^j}\right) = \frac{\partial \xi^\beta}{\partial x^k}\frac{\partial^2 x^k}{\partial \xi^\alpha \partial \xi^\beta}\frac{\partial \xi^\alpha}{\partial x^j} + \frac{\partial}{\partial \xi^\alpha}\frac{\partial \xi^\alpha}{\partial x^j} = \frac{\partial \xi^\beta}{\partial x^k}\frac{\partial^2 x^k}{\partial \xi^\alpha \partial \xi^\beta}\frac{\partial \xi^\alpha}{\partial x^j} + \frac{\partial x^k}{\partial \xi^\alpha}\frac{\partial^2 \xi^\alpha}{\partial x^j \partial x^k}. \tag{28}$$

The last expression is zero since it is a particular form of more general identity (26) with $i = k$.

A.2. Selected differential operators under coordinate transformation

Using results from Appendix A.1 we derive transformation laws for particular differential operators.

Two equations for the first derivative

$$c^j \frac{\partial \phi}{\partial x^j} = c^j \frac{\partial \xi^\alpha}{\partial x^j}\frac{\partial \phi}{\partial \xi^\alpha} \tag{29}$$

and for the second

$$a^{kj}\frac{\partial^2 \phi}{\partial x^j \partial x^k} = a^{kj}\frac{\partial \xi^\beta}{\partial x^j}\frac{\partial}{\partial \xi^\beta}\left(\frac{\partial \xi^\gamma}{\partial x^k}\frac{\partial \phi}{\partial \xi^\gamma}\right) = a^{kj}\frac{\partial \xi^\beta}{\partial x^j}\frac{\partial \xi^\gamma}{\partial x^k}\frac{\partial^2 \phi}{\partial \xi^\beta \partial \xi^\gamma} + a^{kj}\frac{\partial^2 \xi^\gamma}{\partial x^k \partial x^j}\frac{\partial \phi}{\partial \xi^\gamma} \tag{30}$$

follow simply from the chain rule. To apply these equations when only a mapping $x(\xi)$ is known one needs to use relations (21) and (25) for (30).

Conservative forms of the equations above read

$$\frac{\partial}{\partial x^\alpha}(c^\alpha \phi) = \frac{1}{J}\frac{\partial}{\partial \xi^k}\left(Jc^\alpha \frac{\partial \xi^k}{\partial x^\alpha}\phi\right) \tag{31}$$

and

$$\frac{\partial}{\partial x^k}\left(a^{kj}\frac{\partial \phi}{\partial x^j}\right) = \frac{1}{J}\frac{\partial}{\partial \xi^k}\left(J\left(a^{\alpha j}\frac{\partial \xi^k}{\partial x^\alpha}\frac{\partial \xi^\beta}{\partial x^j}\right)\frac{\partial \phi}{\partial \xi^\beta}\right) \tag{32}$$

respectively.

Equation (31) is straightforward to confirm using (27). Indeed,

$$\frac{1}{J}\frac{\partial}{\partial \xi^k}\left(Jc^\alpha \frac{\partial \xi^k}{\partial x^\alpha}\phi\right) = \underbrace{\frac{1}{J}\frac{\partial}{\partial \xi^k}\left(J\frac{\partial \xi^k}{\partial x^\alpha}\right)c^\alpha \phi}_{=0} + \frac{\partial \xi^k}{\partial x^\alpha}\phi\frac{\partial c^\alpha}{\partial \xi^k} + \frac{\partial \xi^k}{\partial x^\alpha}c^\alpha\frac{\partial \phi}{\partial \xi^k} = \frac{\partial}{\partial x^\alpha}(c^\alpha \phi). \tag{33}$$

Equation (32) is easier to derive from the analogous result for the divergence of a vectors field

$$\frac{\partial}{\partial x^k}f^k(x) = \frac{1}{J}\frac{\partial}{\partial \xi^j}\left(Jf^i(x(\xi))\frac{\partial \xi^j}{\partial x^i}\right). \tag{34}$$

Equation (34) itself trivially follows from (27).

Having (34) we derive (32) as follows

$$\frac{\partial}{\partial x^k}\left(a^{kj}\frac{\partial \phi}{\partial x^j}\right) = \frac{1}{J}\frac{\partial}{\partial \xi^k}\left(Ja^{\alpha j}\frac{\partial \phi}{\partial x^j}\frac{\partial \xi^k}{\partial x^\alpha}\right) = \frac{1}{J}\frac{\partial}{\partial \xi^k}\left(J\left(a^{\alpha j}\frac{\partial \xi^k}{\partial x^\alpha}\frac{\partial \xi^\beta}{\partial x^j}\right)\frac{\partial \phi}{\partial \xi^\beta}\right). \tag{35}$$

A.3. Selected PDEs under coordinate transformation

Results from Appendix A.2 allows to derive transformed form of a large set of equations. We confine our attention to stationary diffusion (11), convection-diffusion (12) and wave equations (13).

We start with the derivation of the transformation law for (11). According to (32), after the transformation (7) equation becomes

$$\frac{\partial}{\partial\xi^k}\left(J\left(a^{\alpha j}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial\xi^k}{\partial x^\alpha}\frac{\partial\xi^\beta}{\partial x^j}\right)\frac{\partial u(\boldsymbol{x}(\boldsymbol{\xi}))}{\partial\xi^\beta}\right) = Jf(\boldsymbol{x}(\boldsymbol{\xi})), \tag{36}$$

with the same boundary conditions. So, it is exactly the same parametric equation as (11) but with different parameters.

Next, we consider the convection-diffusion equation (12). Using again (32) and (31) we obtain a transformed form

$$\frac{\partial}{\partial t}\phi(\boldsymbol{x}(\boldsymbol{\xi}),t) + \frac{1}{J}\frac{\partial}{\partial\xi^i}\left(Jv^k(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial\xi^i}{\partial x^k}\phi(\boldsymbol{x}(\boldsymbol{\xi}),t)\right) = \frac{1}{J}\frac{\partial}{\partial\xi^k}\left(J\left(a^{\alpha j}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial\xi^k}{\partial x^\alpha}\frac{\partial\xi^\beta}{\partial x^j}\right)\frac{\partial\phi(\boldsymbol{x}(\boldsymbol{\xi}),t)}{\partial\xi^\beta}\right). \tag{37}$$

This time we can see that (37) does not have the same parametric form as (12). To resolve the problem we define a new field $\psi(\boldsymbol{\xi},t) \equiv J\phi(\boldsymbol{x}(\boldsymbol{\xi}),t)$. Multiplying both sides of Equation (37) by J we observe that left hand side has a desired parametric form. For the right hand side we proceed as follows

$$\frac{\partial}{\partial\xi^k}\left(Ja^{\alpha j}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial\xi^k}{\partial x^\alpha}\frac{\partial\xi^\beta}{\partial x^j}\frac{\partial}{\partial\xi^\beta}\left(\frac{\psi(\boldsymbol{x}(\boldsymbol{\xi}),t)}{J}\right)\right) = \frac{\partial}{\partial\xi^k}\left(a^{\alpha j}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial\xi^k}{\partial x^\alpha}\frac{\partial\xi^\beta}{\partial x^j}\frac{\partial\psi(\boldsymbol{x}(\boldsymbol{\xi}),t)}{\partial\xi^\beta}\right)$$
$$-\frac{\partial}{\partial\xi^k}\left(a^{\alpha j}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial\xi^k}{\partial x^\alpha}\frac{\partial\xi^\beta}{\partial x^j}\frac{\partial\xi^\gamma}{\partial x^\rho}\frac{\partial^2 x^\rho}{\partial\xi^\gamma\partial\xi^\beta}\psi(\boldsymbol{x}(\boldsymbol{\xi}))\right), \tag{38}$$

where we used (24). As we can see the right hand side of (37) introduces additional contribution to the convection term, and with this contribution the final equation has the same parametric form as the original convection-diffusion equation (12).

In the case of two-way wave equation (13) we apply (29) and (30) to obtain

$$\frac{\partial^2\rho(\boldsymbol{x}(\boldsymbol{\xi}),t)}{\partial t^2} + v^i(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial\xi^\alpha}{\partial x^i}\frac{\partial\rho(\boldsymbol{x}(\boldsymbol{\xi}),t)}{\partial\xi^\alpha} = c^{kj}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial\xi^\beta}{\partial x^j}\frac{\partial\xi^\gamma}{\partial x^k}\frac{\partial^2\rho(\boldsymbol{x}(\boldsymbol{\xi}),t)}{\partial\xi^\beta\partial\xi^\gamma} + c^{kj}(\boldsymbol{x}(\boldsymbol{\xi}))\frac{\partial^2\xi^\gamma}{\partial x^k\partial x^j}\frac{\partial\rho(\boldsymbol{x}(\boldsymbol{\xi}),t)}{\partial\xi^\gamma}$$
$$+ e(\boldsymbol{x}(\boldsymbol{\xi}))\rho(\boldsymbol{x}(\boldsymbol{\xi})). \tag{39}$$

All transformation laws we derived can be found in Table 1.

B. Coordinate derivatives for transfinite interpolation

For the case of transfinite interpolation Equation (8), Jacobi matrix and higher order derivatives slightly simplify. Here we provide explicit expressions for the $D = 2$ case:

1. Jacobi matrix

$$\mathcal{J}_{ij} = \frac{\partial x^i}{\partial\xi^j} = \begin{pmatrix} y_1'(\xi^1)(1-\xi^2) + y_2'(\xi^1)\xi^2 & y_2(\xi^1) - y_1(\xi^1) \\ y_4(\xi^2) - y_3(\xi^2) & (1-\xi^1)y_3'(\xi^2) + \xi^1 y_4'(\xi^2) \end{pmatrix}. \tag{40}$$

2. Inverse Jacobi matrix

$$\left(\mathcal{J}^{-1}\right)_{ij} = \frac{\partial\xi^i}{\partial x^j} = \frac{1}{J}\begin{pmatrix} (1-\xi^1)y_3'(\xi^2) + \xi^1 y_4'(\xi^2) & y_1(\xi^1) - y_2(\xi^1) \\ y_3(\xi^2) - y_4(\xi^2) & y_1'(\xi^1)(1-\xi^2) + y_2'(\xi^1)\xi^2 \end{pmatrix}, \quad J = \det\mathcal{J}. \tag{41}$$

3. Second derivatives

$$\frac{\partial^2 x^1}{\partial\xi^i\partial\xi^j} = \begin{pmatrix} y_1''(\xi^1)(1-\xi^2) + y_2''(\xi^1)\xi^2 & P_2'(\xi^1) - P_1'(\xi^1) \\ P_2'(\xi^1) - P_1'(\xi^1) & 0 \end{pmatrix},$$
$$\frac{\partial^2 x^2}{\partial\xi^i\partial\xi^j} = \begin{pmatrix} 0 & P_4'(\xi^2) - P_3'(\xi^2) \\ P_4'(\xi^2) - P_3'(\xi^2) & (1-\xi^1)y_3''(\xi^2) + \xi^1 y_4''(\xi^2) \end{pmatrix}. \tag{42}$$

C. Architectures and training details

In this section, we provide extended comments on the architectures used and collect in Table 5 the description of the optimization process.

C.1. rFNO

rFNO is a variant of FNO (Li et al., 2020) with two differences.

First, FFT is replaced with projection on the set of trigonometric functions

$$\mathcal{T}_N \equiv \{1, \cos{(2\pi x)}, \sin{(2\pi x)}, \cos{(2\pi 2x)}, \sin{(2\pi 2x)}, \dots, \cos{(2\pi Nx)}, \sin{(2\pi Nx)}\}. \tag{43}$$

That is, for the input function $f(x)$, the function after transform reads

$$c_k = \int_0^1 dx \, \phi_k(x) f(x), \; \phi_k \in \mathcal{T}_N, \tag{44}$$

where the integral is approximated using the trapezoidal rule.

The inverse transform is computed simply as a sum

$$g(x) = \sum_k c_k \phi_k(x), \; \phi_k \in \mathcal{T}_N. \tag{45}$$

Second, between the transformations, FNO (Li et al., 2020) uses a diagonal tensor, so the resulting matrix performs convolution (see (Rippel et al., 2015)). In our case we apply a series of convolutions N_{conv} without activations.

In all $1D$ experiments, we keep $N = 8$ modes. Use $N_{\text{conv}} = 4$ convolutions with kernel size 3 in the Fourier space (43). The encoder lifts the input to the space with $N_{\text{features}} = 64$ features. The number of Fourier layers is 4. We use ReLU activation functions.

C.2. MLP

Since we process functions sampled on the grid, we use MLP that applies a linear operator to each dimension and feature space separately, i.e., if we have a tensor t_{ijk} as an input, the linear (affine transform) layer transforms it as follows

$$\widetilde{t}_{abc} = \sigma \left(\sum_{ijk} A_{ai} B_{bj} C_{ck} t_{ijk} + e_{abc} \right), \tag{46}$$

where A, B, C are parameters of linear transform and e is bias and σ is the activation function.

Table 5. Training details: ν — learning rate, ν decay / epoch — weight decay per epoch, N_{epoch} — number of epoch used for training, N_{batch} — batch size, N_{params} — number of network parameters.
[†] in DeepONet, inverse time function means $\nu = \nu/(1 + 0.5 \cdot \text{steps}/g)$, where $g = \lfloor \text{epoch}/5 \rfloor$
[‡] 1000 epochs were used for $D = 2$ datasets.

Network	ν	ν decay / epoch	weight decay	N_{epoch}	N_{batch}	N_{params}	
						$D = 1$	$D = 2$
FNO	10^{-3}	—	10^{-4}	500	200	549×10^3	236×10^4
DeepONet	10^{-3}	—	inverse time[†]	2×10^4	full train set	150×10^3	816×10^4
rFNO	10^{-3}	$0.5/100$	10^{-2}	500	30	215×10^3	—
MLP	10^{-3}	$0.5/100$	10^{-2}	500[‡]	30	63×10^3	113×10^3
DilResNet	10^{-3}	$0.5/100$	10^{-2}	500	30	87×10^3	261×10^3
U-Net	10^{-3}	$0.5/100$	10^{-2}	500	30	—	263×10^3
SNO	10^{-3}	$0.5/100$	10^{-2}	500	30	—	115×10^3

We use in total 4 such layers both in $D = 1$ and $D = 2$, and with the same number of spatial points as input has, and the number of features increased to 64. Again, ReLU activation functions were used.

C.3. DilResNet

The dilated residual network follows the publication (Stachenfeld et al., 2021). We use 4 blocks and 32 features, each block consists of convolutions with strides $[1, 2, 4, 8, 4, 2, 1]$ each with kernel size 3. After each block, we put a skip connection. As before, ReLU activation functions were used.

C.4. U-Net

The usual form of U-Net was used (Ronneberger et al., 2015).

U-Net induces the series of grids (levels) each having roughly $\times 2$ fewer points, and the number of features doubles. We used 3 convolutions on each level before max pooling, transposed convolution for upsampling, and 3 convolutions for each level after the upsampling. In total we have 4 layers and start with 10 features. Again, ReLU activation functions were used.

C.5. DeepONet

DeepONet (Lu et al., 2019) consists of two sub-networks, one for encoding the input function v at a fixed number of sensors $x_i, i = 1, \ldots, m$ (branch net), and the other for encoding the locations ξ for the output functions (trunk net). The output of the network can be expressed as

$$\mathcal{G}(v)(\xi) = \sum_{k=1}^{p} b_k(v) t_k(\xi) + b_0, \tag{47}$$

where b_0 is the bias, $\{b_k\}_{k=1}^{p}$ are the outputs of the branch net, and $\{t_k\}_{k=1}^{p}$ are the outputs of the trunk net.

For the $D = 1$ problem, both branch net and trunk net are fully connected neural networks(FNNs). For branch net, we use 4 layers with $N_{\text{features}} = 128$ features; for trunk net, we use 3 layers with $N_{\text{features}} = 128$ features. Additionally, we utilized $\tanh$ as the activation function, and Glorot normal initializer to initialize the weights of DeepONet.

For the $D = 2$ problem, the trunk net is fully connected neural networks(FNNs) with 3 layers, $N_{\text{features}} = 128$ features each. For the branch net, we utilized 2 convolutional layers with $N_{\text{features}} = 64$ features and $N_{\text{features}} = 128$ features respectively, then we had a flatten layer and two fully connected layers with $N_{\text{features}} = 128$ features. Additionally, we utilized ReLU as the activation function, and Glorot normal initializer to initialize the weights of DeepONet.

C.6. FNO

The original form of FNO was proposed in (Li et al., 2020). This network consists of encoder, several Fourier layers and decoder. The $(l + 1)$-th Fourier layer can be expressed as

$$z_{l+1} = \sigma \left(\mathcal{F}^{-1} \left(R_l \cdot \mathcal{F}(z_l) \right) + \text{conv}(z_l)_{W_l} + b_l \right), \tag{48}$$

where R_l, W_l are the weight matrices, σ is the activation function, b_l is the bias, and $\mathcal{F}$ is the Fast Fourier transform, $\mathcal{F}^{-1}$ is the inverse, and conv stands for convolution with kernel size 1.

For the $D = 1$ problem, the number of Fourier layers is 4. Each layer had $N_{\text{features}} = 64$ features and $N = 16$ modes. Additionally, we utilized GELU as the activation function. In addition, we had a linear encoder and decoder implemented as fully connected layers. The encoder was a single layer that lifts the input function to the space with 64 features. The decoder had two fully connected layers that change the number of features to 64, then to 128, and finally to 1, which is the target number of features for all datasets used.

For the $D = 2$ problem, the same number of FNO layers was used. Each layer had $N_{\text{features}} = 32$ features and $N = 12$ modes. The encoder lifted the input function to the space with 32 features. The decoder consisted of two fully connected layers, that firstly increased number of features to 128 and then reduced them to the target number of features. Also, we used GELU activation functions.

C.7. SNO

SNO follows the design of FNO with two notable differences. First, Discrete Cosine Transform replaces FFT. As explained in (Fanaskov & Oseledets, 2022), this corresponds to the approximate recovery of coefficient in the Chebyshev series, should the input function is sampled on the Chebyshev grid. Second, after DCT, we use three convolutions with kernel size 3 as a linear kernel. The number of features used in the processor is 32, number of modes left after truncation is 16. In total, network contains 4 layers with the integral operator, ReLU nonlinearities in-between them, linear encoder, and decoder.

D. Datasets

In this section, we provide more details on the dataset generation process. Datasets can be downloaded from `https://disk.yandex.ru/d/ArC6jT3TZcKncw`.

D.1. $D = 1$

To generate input data for $D = 1$ PDEs we use two families of trigonometric functions

$$f_N(x) = \sum_{k=0}^{N} c_k \cos(2\pi k x + p_k) \tag{49}$$

and

$$g_N(x) = \sum_{k=0}^{N} c_k \sin(\pi(k+1)x). \tag{50}$$

Functions $g_N(x)$ with c_k from standard normal distribution were used to generate initial conditions for wave and convection-diffusion equations.

Functions $f_N(x)$ were used for two purposes. First, we generated positive diffusion coefficients taking $p_k, c_k, k > 0$ from standard normal distribution and fixing $p_0 = 0, c_0 = \sum_{k>1} |c_k| + \epsilon$ with $\epsilon = 10^{-2}$. Second, we used them with $p_k, c_k, k \geq 0$ taken from standard normal distribution to generate convection coefficient $v(x)$ for the convection-diffusion problem and right-hand side $f(x)$ for the elliptic problem.

The list of generated datasets is as follows:

1. Elliptic (11)

   The right-hand side was generated using (50) with $N = 3$ coefficients, for diffusion coefficient we use (49) with $N = 5$. We perform discretization with standard FEM method with hat functions on the uniform grid with 100 points.

2. Convection-diffusion (12)

   Convection coefficient was generated using (49) with $N = 5$ coefficients, for diffusion coefficient we use (49) with $N = 5$, initial conditions were generated using (50) with $N = 10$. After generation diffusion and convection coefficients are multiplied by $s = 0.01$. Spatial discretization is the same as for the Elliptic dataset, for the time-marching Crank-Nicolson scheme was used, final $t = 1.0$, 200 points were used along t.

3. Wave(5) (13)

   All functions were sampled with $N = 5$ modes. For initial conditions, we used (50), and for source terms, diffusion, and convection coefficients we used (49) multiplied by $s = 0.1$. To make the diffusion coefficient positive we squared (49). For all functions, we used an additional factor for $1/k^2$ for coefficient k to obtain smooth functions. Along the spacial dimension, we use 100 points and the standard second-order finite-difference discretization, along temporal 1000 points. As a marching scheme, we used leapfrog and take $t = 1.0$ as a final time.

4. Wave(10) (13)

   The same as previous but with $N = 10$ for all sampled functions.

D.2. $D = 2$

In this case, we use a single family of random functions

$$f(x) = \mathcal{R}\left(\sum_{m=-M}^{M}\sum_{n=-M}^{M} e^{2\pi i(mx+ny)} c_{mn}\right), \tag{51}$$

where $\mathcal{R}$ is a real part and $c_{mn} = x_{mn} + iy_{mn}$ where both x_{mn} and y_{mn} are samples from standard normal distribution.

The elliptic part of differential operators requires a uniformly positive definite matrix. To enforce this condition we generate Cholesky decomposition of diffusion coefficient A as follows

$$A = I + LL^T, \tag{52}$$

where each non-negative element of upper-triangular matrix L is an independently generated function (51).

The list of generated datasets is as follows:

1. Elliptic alpha (11) Diffusion coefficients were generated in the form of Cholesky decomposition (52). The right-hand side was generated using (51). For the simple dataset coefficients in (51) were multiplied by a factor $s = 0.1$ for the complex dataset by the factor $s = 0.5$. For both simple and complex datasets we used $M = 5$ in (51). Bilinear FEM discretization on the rectangular grid with 100×100 points was used for discretization.

2. Elliptic beta (11) For this dataset the matrix A is diagonal. We generated everything the same way as for the previous datasets, and afterward, drop non-diagonal elements of A and replace A_{22} with A_{11}.

3. Convection-diffusion (12) The diffusion coefficients were generated the same way as for the Elliptic alpha dataset. Initial conditions and convection coefficients are taken from (51). For each function, $M = 5$ is used. For the complex dataset, we rescale coefficients by a factor $s = 0.5$ and take $t = 1e - 2$. For the simple dataset, these parameters are $s = 0.1$, $t = 1e - 2$. As in the $D = 1$ case Crank-Nicolson time-marching scheme is used. Spatial discretization is the same as for the Elliptic alpha dataset, 100 points along a temporal dimension are used.

4. Wave (13) All random functions were generated the same way as for the convection-diffusion equation. The source term is not used for $D = 2$. For the complex dataset, we have $s = 0.2$, $t = 1$, and for the simple, we put $s = 0.2$, $t = 1e - 1$.

E. Details on the solution method for lid-driven cavity flow

To integrate the equation we use the Chorin projection method that works in three steps:

1. Advance speed neglecting pressure term

$$\frac{u^i - v^i_{(n)}}{\Delta t} = \frac{\partial}{\partial x^k}\left(-v^k_{(n)} v^i_{(n)} + \nu \frac{\partial v^i_{(n)}}{\partial x^k}\right) \tag{53}$$

2. Solve the Poisson equation to obtain pressure correction term

$$\frac{\partial}{\partial x^k}\frac{\partial p}{\partial x^k} = \frac{\partial u^k}{\partial x^k}, \; p(x,1) = 0, \; \frac{\partial p(x,0)}{\partial y} = \frac{\partial p(0,y)}{\partial x} = \frac{\partial p(1,y)}{\partial x} = 0. \tag{54}$$

3. Correct speed to restore incompressibility

$$v^i_{(n+1)} = u^i - \frac{\partial p}{\partial x^i} \tag{55}$$

Since the problem is defined on a deformed mesh, we need to rewrite the Chorin projection method in the curvilinear coordinates.

Under the coordinate transformations, the equation for pressure changes to

$$\frac{\partial}{\partial \xi^i}\left(J\delta^{\alpha\beta}\frac{\partial \xi^i}{\partial x^\alpha}\frac{\partial \xi^j}{\partial x^\beta}\frac{\partial p}{\partial \xi^j}\right) = \frac{\partial}{\partial \xi^j}\left(\overline{u}^j J\right), \ \overline{u}^j = \frac{\partial \xi^j}{\partial x^\alpha}u^\alpha,$$

$$p\left(\xi^1, 1\right) = 1, \ \left(\frac{\partial p}{\partial \xi^1}\frac{\partial \xi^1}{\partial x^2} + \frac{\partial p}{\partial \xi^2}\frac{\partial \xi^2}{\partial x^2}\right)_{\xi^2=0} = 0, \tag{56}$$

$$\left(\frac{\partial p}{\partial \xi^1}\frac{\partial \xi^1}{\partial x^1} + \frac{\partial p}{\partial \xi^2}\frac{\partial \xi^2}{\partial x^1}\right)_{\xi^1=0} = 0, \ \left(\frac{\partial p}{\partial \xi^1}\frac{\partial \xi^1}{\partial x^1} + \frac{\partial p}{\partial \xi^2}\frac{\partial \xi^2}{\partial x^1}\right)_{\xi^1=1} = 0.$$

The equation for speed update is a vector conservation law

$$\frac{\partial A^{ij}}{\partial x^j} = F^i, \tag{57}$$

with the following transformation rule (see (Liseikin, 2017), Section 2.4.2)

$$\frac{\partial}{\partial \xi^j}\left(J\overline{A}^{ij}\right) + \frac{\partial^2 x^l}{\partial \xi^k \partial \xi^j}\frac{\partial \xi^i}{\partial x^l}\overline{A}^{kj} = \overline{F}^i, \ \overline{A}^{kj} = \frac{\partial \xi^k}{\partial x^\alpha}\frac{\partial \xi^j}{\partial x^\beta}A^{\alpha\beta}, \ \overline{F}^j = \frac{\partial \xi^j}{\partial x^\alpha}F^\alpha. \tag{58}$$

It is straightforward to apply general equation Equation (58) to a particular case

$$\frac{u^i - v^i_{(n)}}{\Delta t} = \frac{\partial}{\partial x^k}\left(-v^k_{(n)}v^i_{(n)} + \nu\frac{\partial v^i_{(n)}}{\partial x^k}\right). \tag{59}$$

The only problematic term is

$$\frac{\partial v^i_{(n)}}{\partial x^k} \longrightarrow \frac{\partial \xi^i}{\partial x^\alpha}\frac{\partial \xi^k}{\partial x^\beta}\frac{\partial v^\alpha_{(n)}}{\partial x^\beta} = \frac{\partial \xi^i}{\partial x^\alpha}\frac{\partial \xi^k}{\partial x^\beta}\frac{\partial \xi^\rho}{\partial x^\beta}\frac{\partial v^\alpha_{(n)}}{\partial \xi^\rho}. \tag{60}$$

To evaluate this term, we need to switch from $\overline{v}^\alpha$ to v^α which is a simple task since the Jacobi matrix is available.

F. Supplementray results

Here we present additional data on the experiments described in Section 4.

Table 7, Table 8, Table 9, Table 10 contain results on $D = 1$ experiments.

In Table 6, one can find data with sensitivity to coordinate transforms.

Results for DeepONet training on Navier-Stokes dataset are in Table 11.

Table 6. Sensitivity to grid distortion for DilResNet and FNO with $\sqrt{}$ and without $\times$ augmentation. The distortion here refers to the maximal difference between the unperturbed $\boldsymbol{x}$ and perturbed $\boldsymbol{x}(\boldsymbol{\xi})$ grids averaged over 1000 grids used to augment dataset.

	Elliptic alpha				Convection-diffusion				Wave			
	DilResNet		FNO		DilResNet		FNO		DilResNet		FNO	
Δ	$\times$	$\sqrt{}$	$\times$	$\sqrt{}$	$\times$	$\sqrt{}$	$\times$	$\sqrt{}$	$\times$	$\sqrt{}$	$\times$	$\sqrt{}$
0.006	10%	2%	4%	1%	2%	1%	2%	1%	2%	2%	5%	2%
0.040	14%	2%	7%	2%	15%	2%	15%	4%	17%	4%	28%	9%
0.099	42%	3%	15%	5%	—	25%	—	26%	—	18%	—	34%
0.119	70%	4%	18%	6%	—	58%	—	54%	—	44%	—	69%

Table 7. Average test errors $\pm$ standard deviation for elliptic equation in one dimension. Factor m in columns corresponds to the number of extra samples $m \times N_{\text{train}}$ added to the dataset with augmentation or resampling.

Model		$N_{\text{train}} \backslash m$	1	2	3	4
DeepONet	augmentation	500	0.573 ± 0.145	0.419 ± 0.032	0.415 ± 0.038	0.427 ± 0.041
		1000	0.489 ± 0.071	0.415 ± 0.033	0.383 ± 0.022	0.373 ± 0.013
		1500	0.461 ± 0.041	0.398 ± 0.019	0.399 ± 0.075	0.465 ± 0.101
		2000	0.417 ± 0.023	0.385 ± 0.025	0.383 ± 0.023	0.375 ± 0.024
	resampling	500	0.633 ± 0.066	0.492 ± 0.042	0.445 ± 0.026	0.388 ± 0.017
		1000	0.597 ± 0.07	0.536 ± 0.072	0.441 ± 0.023	0.705 ± 0.606
		1500	0.646 ± 0.092	0.505 ± 0.031	0.46 ± 0.039	0.415 ± 0.011
		2000	0.624 ± 0.072	0.504 ± 0.04	0.428 ± 0.026	0.452 ± 0.078
FNO	augmentation	500	0.125 ± 0.014	0.063 ± 0.003	0.047 ± 0.002	0.039 ± 0.002
		1000	0.09 ± 0.006	0.052 ± 0.002	0.04 ± 0.002	0.032 ± 0.001
		1500	0.074 ± 0.004	0.043 ± 0.003	0.035 ± 0.002	0.028 ± 0.001
		2000	0.064 ± 0.004	0.04 ± 0.002	0.032 ± 0.002	0.026 ± 0.001
	resampling	500	0.12 ± 0.001	0.068 ± 0.004	0.053 ± 0.002	0.043 ± 0.002
		1000	0.106 ± 0.004	0.063 ± 0.001	0.049 ± 0.002	0.04 ± 0.001
		1500	0.102 ± 0.005	0.061 ± 0.003	0.049 ± 0.002	0.041 ± 0.003
		2000	0.098 ± 0.004	0.063 ± 0.004	0.048 ± 0.003	0.04 ± 0.001
rFNO	augmentation	500	0.146 ± 0.004	0.121 ± 0.004	0.106 ± 0.004	0.099 ± 0.003
		1000	0.103 ± 0.002	0.087 ± 0.004	0.08 ± 0.004	0.076 ± 0.002
		1500	0.082 ± 0.002	0.074 ± 0.002	0.07 ± 0.003	0.064 ± 0.001
		2000	0.073 ± 0.001	0.065 ± 0.002	0.061 ± 0.001	0.056 ± 0.001
	resampling	500	0.17 ± 0.005	0.154 ± 0.003	0.148 ± 0.006	0.148 ± 0.004
		1000	0.111 ± 0.002	0.111 ± 0.002	0.107 ± 0.002	0.105 ± 0.004
		1500	0.089 ± 0.002	0.087 ± 0.003	0.086 ± 0.002	0.084 ± 0.002
		2000	0.078 ± 0.002	0.076 ± 0.003	0.074 ± 0.002	0.075 ± 0.002
DilResNet	augmentation	500	0.374 ± 0.037	0.304 ± 0.033	0.243 ± 0.031	0.214 ± 0.026
		1000	0.208 ± 0.005	0.172 ± 0.013	0.145 ± 0.004	0.141 ± 0.015
		1500	0.179 ± 0.013	0.143 ± 0.006	0.119 ± 0.009	0.114 ± 0.01
		2000	0.13 ± 0.009	0.118 ± 0.009	0.104 ± 0.008	0.095 ± 0.007
	resampling	500	0.485 ± 0.044	0.48 ± 0.087	0.41 ± 0.063	0.425 ± 0.052
		1000	0.255 ± 0.009	0.255 ± 0.027	0.218 ± 0.011	0.226 ± 0.021
		1500	0.185 ± 0.024	0.17 ± 0.014	0.171 ± 0.008	0.17 ± 0.015
		2000	0.15 ± 0.014	0.141 ± 0.011	0.143 ± 0.008	0.14 ± 0.005
MLP	augmentation	500	0.346 ± 0.031	0.337 ± 0.031	0.261 ± 0.041	0.262 ± 0.043
		1000	0.268 ± 0.042	0.181 ± 0.03	0.162 ± 0.026	0.126 ± 0.016
		1500	0.23 ± 0.049	0.145 ± 0.017	0.13 ± 0.007	0.105 ± 0.009
		2000	0.155 ± 0.028	0.109 ± 0.007	0.09 ± 0.005	0.089 ± 0.012
	resampling	500	0.342 ± 0.023	0.377 ± 0.011	0.332 ± 0.066	0.286 ± 0.053
		1000	0.309 ± 0.048	0.301 ± 0.105	0.239 ± 0.096	0.184 ± 0.043
		1500	0.211 ± 0.03	0.133 ± 0.026	0.111 ± 0.007	0.111 ± 0.018
		2000	0.151 ± 0.038	0.099 ± 0.013	0.096 ± 0.012	0.089 ± 0.006

Table 8. Average test errors $\pm$ standard deviation for convection-diffusion equation in one dimension. Factor m in columns corresponds to the number of extra samples $m \times N_{\text{train}}$ added to the dataset with augmentation or resampling.

Model		$N_{\text{train}}\backslash m$	1	2	3	4
DeepONet	augmentation	500	0.8 ± 0.013	0.619 ± 0.012	0.522 ± 0.022	0.449 ± 0.018
		1000	0.716 ± 0.013	0.536 ± 0.004	0.469 ± 0.013	0.401 ± 0.02
		1500	0.646 ± 0.02	0.477 ± 0.009	0.407 ± 0.01	0.327 ± 0.02
		2000	0.607 ± 0.019	0.46 ± 0.014	0.386 ± 0.015	0.312 ± 0.019
	resampling	500	0.92 ± 0.034	0.771 ± 0.02	0.671 ± 0.025	0.619 ± 0.024
		1000	0.915 ± 0.017	0.769 ± 0.016	0.665 ± 0.017	0.612 ± 0.022
		1500	0.926 ± 0.041	0.772 ± 0.017	0.676 ± 0.011	0.619 ± 0.02
		2000	0.92 ± 0.041	0.768 ± 0.031	0.672 ± 0.019	0.623 ± 0.03
FNO	augmentation	500	0.464 ± 0.008	0.366 ± 0.023	0.235 ± 0.039	0.117 ± 0.026
		1000	0.472 ± 0.011	0.324 ± 0.033	0.137 ± 0.036	0.072 ± 0.007
		1500	0.438 ± 0.043	0.229 ± 0.085	0.101 ± 0.017	0.059 ± 0.005
		2000	0.426 ± 0.036	0.212 ± 0.057	0.081 ± 0.016	0.05 ± 0.008
	resampling	500	0.499 ± 0.01	0.426 ± 0.016	0.315 ± 0.027	0.157 ± 0.031
		1000	0.529 ± 0.009	0.442 ± 0.026	0.258 ± 0.04	0.133 ± 0.028
		1500	0.553 ± 0.013	0.434 ± 0.03	0.264 ± 0.051	0.124 ± 0.017
		2000	0.543 ± 0.009	0.444 ± 0.02	0.233 ± 0.033	0.125 ± 0.013
rFNO	augmentation	500	0.524 ± 0.008	0.49 ± 0.007	0.436 ± 0.039	0.402 ± 0.049
		1000	0.176 ± 0.005	0.157 ± 0.037	0.124 ± 0.006	0.115 ± 0.007
		1500	0.108 ± 0.003	0.097 ± 0.004	0.088 ± 0.004	0.082 ± 0.004
		2000	0.083 ± 0.005	0.071 ± 0.002	0.068 ± 0.002	0.067 ± 0.004
	resampling	500	0.536 ± 0.007	0.513 ± 0.006	0.507 ± 0.007	0.485 ± 0.019
		1000	0.265 ± 0.057	0.258 ± 0.04	0.219 ± 0.013	0.194 ± 0.02
		1500	0.137 ± 0.008	0.124 ± 0.009	0.119 ± 0.005	0.115 ± 0.007
		2000	0.102 ± 0.004	0.094 ± 0.003	0.09 ± 0.003	0.09 ± 0.002
DilResNet	augmentation	500	0.133 ± 0.014	0.109 ± 0.006	0.1 ± 0.006	0.091 ± 0.008
		1000	0.07 ± 0.004	0.058 ± 0.002	0.048 ± 0.004	0.044 ± 0.002
		1500	0.047 ± 0.004	0.041 ± 0.002	0.037 ± 0.002	0.033 ± 0.003
		2000	0.038 ± 0.003	0.032 ± 0.001	0.029 ± 0.001	0.026 ± 0.002
	resampling	500	0.155 ± 0.006	0.172 ± 0.026	0.144 ± 0.008	0.144 ± 0.014
		1000	0.075 ± 0.002	0.072 ± 0.005	0.067 ± 0.003	0.068 ± 0.007
		1500	0.052 ± 0.002	0.05 ± 0.003	0.047 ± 0.003	0.047 ± 0.001
		2000	0.041 ± 0.002	0.038 ± 0.002	0.036 ± 0.002	0.037 ± 0.001
MLP	augmentation	500	0.425 ± 0.03	0.439 ± 0.011	0.439 ± 0.031	0.423 ± 0.015
		1000	0.306 ± 0.049	0.301 ± 0.045	0.291 ± 0.086	0.266 ± 0.055
		1500	0.26 ± 0.059	0.175 ± 0.058	0.125 ± 0.013	0.115 ± 0.032
		2000	0.105 ± 0.008	0.101 ± 0.015	0.071 ± 0.005	0.074 ± 0.014
	resampling	500	0.464 ± 0.018	0.507 ± 0.037	0.5 ± 0.014	0.482 ± 0.012
		1000	0.381 ± 0.034	0.361 ± 0.086	0.379 ± 0.047	0.328 ± 0.061
		1500	0.306 ± 0.056	0.299 ± 0.064	0.2 ± 0.074	0.231 ± 0.039
		2000	0.19 ± 0.052	0.183 ± 0.091	0.135 ± 0.023	0.114 ± 0.015

Table 9. Average test errors $\pm$ standard deviation for wave equation (10 modes) in one dimension. Factor m in columns corresponds to the number of extra samples $m \times N_{\text{train}}$ added to the dataset with augmentation or resampling.

Model		$N_{\text{train}}\backslash m$	1	2	3	4
DeepONet	augmentation	500	0.275 ± 0.009	0.215 ± 0.004	0.181 ± 0.005	0.171 ± 0.005
		1000	0.259 ± 0.004	0.207 ± 0.005	0.175 ± 0.004	0.163 ± 0.004
		1500	0.255 ± 0.001	0.202 ± 0.003	0.174 ± 0.007	0.162 ± 0.006
		2000	0.244 ± 0.003	0.197 ± 0.005	0.172 ± 0.003	0.164 ± 0.006
	resampling	500	0.322 ± 0.014	0.241 ± 0.004	0.198 ± 0.003	0.18 ± 0.006
		1000	0.319 ± 0.009	0.25 ± 0.021	0.196 ± 0.005	0.179 ± 0.005
		1500	0.314 ± 0.011	0.237 ± 0.005	0.212 ± 0.029	0.18 ± 0.007
		2000	0.321 ± 0.008	0.237 ± 0.005	0.197 ± 0.005	0.18 ± 0.006
FNO	augmentation	500	0.135 ± 0.003	0.088 ± 0.002	0.068 ± 0.002	0.057 ± 0.001
		1000	0.123 ± 0.003	0.082 ± 0.003	0.062 ± 0.001	0.049 ± 0.001
		1500	0.119 ± 0.004	0.076 ± 0.002	0.058 ± 0.001	0.045 ± 0.001
		2000	0.108 ± 0.002	0.072 ± 0.002	0.052 ± 0.001	0.042 ± 0.001
	resampling	500	0.208 ± 0.008	0.14 ± 0.007	0.108 ± 0.004	0.091 ± 0.003
		1000	0.178 ± 0.004	0.124 ± 0.004	0.096 ± 0.004	0.078 ± 0.005
		1500	0.168 ± 0.003	0.118 ± 0.005	0.09 ± 0.004	0.074 ± 0.003
		2000	0.163 ± 0.003	0.114 ± 0.003	0.087 ± 0.003	0.07 ± 0.002
rFNO	augmentation	500	0.19 ± 0.003	0.179 ± 0.002	0.172 ± 0.003	0.166 ± 0.003
		1000	0.147 ± 0.002	0.137 ± 0.002	0.131 ± 0.002	0.127 ± 0.003
		1500	0.127 ± 0.003	0.117 ± 0.004	0.112 ± 0.002	0.108 ± 0.002
		2000	0.111 ± 0.001	0.103 ± 0.001	0.098 ± 0.002	0.094 ± 0.002
	resampling	500	0.213 ± 0.004	0.212 ± 0.003	0.21 ± 0.002	0.207 ± 0.005
		1000	0.168 ± 0.005	0.168 ± 0.002	0.164 ± 0.003	0.163 ± 0.002
		1500	0.144 ± 0.004	0.141 ± 0.001	0.142 ± 0.003	0.143 ± 0.003
		2000	0.129 ± 0.003	0.127 ± 0.003	0.126 ± 0.002	0.126 ± 0.002
DilResNet	augmentation	500	0.157 ± 0.009	0.133 ± 0.007	0.128 ± 0.006	0.121 ± 0.005
		1000	0.109 ± 0.006	0.092 ± 0.003	0.086 ± 0.007	0.084 ± 0.005
		1500	0.084 ± 0.006	0.076 ± 0.003	0.072 ± 0.004	0.066 ± 0.005
		2000	0.076 ± 0.007	0.063 ± 0.004	0.058 ± 0.004	0.056 ± 0.005
	resampling	500	0.173 ± 0.008	0.174 ± 0.007	0.175 ± 0.012	0.182 ± 0.013
		1000	0.121 ± 0.007	0.119 ± 0.011	0.118 ± 0.009	0.126 ± 0.014
		1500	0.098 ± 0.006	0.095 ± 0.006	0.098 ± 0.007	0.093 ± 0.011
		2000	0.082 ± 0.005	0.08 ± 0.002	0.08 ± 0.005	0.079 ± 0.003
MLP	augmentation	500	0.349 ± 0.038	0.331 ± 0.041	0.304 ± 0.041	0.294 ± 0.015
		1000	0.231 ± 0.022	0.198 ± 0.021	0.178 ± 0.015	0.187 ± 0.033
		1500	0.186 ± 0.013	0.145 ± 0.009	0.137 ± 0.007	0.134 ± 0.013
		2000	0.151 ± 0.012	0.131 ± 0.01	0.115 ± 0.012	0.099 ± 0.006
	resampling	500	0.376 ± 0.072	0.349 ± 0.044	0.341 ± 0.033	0.334 ± 0.052
		1000	0.236 ± 0.01	0.225 ± 0.015	0.236 ± 0.036	0.233 ± 0.017
		1500	0.188 ± 0.005	0.192 ± 0.012	0.178 ± 0.009	0.193 ± 0.01
		2000	0.152 ± 0.007	0.159 ± 0.004	0.158 ± 0.001	0.17 ± 0.014

Table 10. Average test errors $\pm$ standard deviation for wave equation (5 modes) in one dimension. Factor m in columns corresponds to the number of extra samples $m \times N_{\text{train}}$ added to the dataset with augmentation or resampling.

Model		$N_{\text{train}} \backslash m$	1	2	3	4
DeepONet	augmentation	500	0.262 ± 0.005	0.178 ± 0.004	0.152 ± 0.003	0.147 ± 0.011
		1000	0.247 ± 0.011	0.169 ± 0.006	0.158 ± 0.013	0.141 ± 0.008
		1500	0.237 ± 0.006	0.161 ± 0.008	0.143 ± 0.006	0.134 ± 0.004
		2000	0.228 ± 0.005	0.162 ± 0.004	0.147 ± 0.01	0.138 ± 0.005
	resampling	500	0.306 ± 0.008	0.196 ± 0.004	0.157 ± 0.005	0.145 ± 0.008
		1000	0.31 ± 0.004	0.195 ± 0.004	0.159 ± 0.006	0.148 ± 0.008
		1500	0.308 ± 0.011	0.197 ± 0.004	0.172 ± 0.015	0.144 ± 0.006
		2000	0.303 ± 0.006	0.197 ± 0.007	0.16 ± 0.007	0.142 ± 0.005
FNO	augmentation	500	0.088 ± 0.002	0.057 ± 0.002	0.042 ± 0.002	0.034 ± 0.001
		1000	0.083 ± 0.003	0.051 ± 0.002	0.038 ± 0.001	0.031 ± 0.001
		1500	0.078 ± 0.002	0.048 ± 0.001	0.036 ± 0.001	0.029 ± 0.001
		2000	0.074 ± 0.0	0.045 ± 0.001	0.033 ± 0.001	0.027 ± 0.001
	resampling	500	0.18 ± 0.012	0.104 ± 0.009	0.073 ± 0.005	0.058 ± 0.002
		1000	0.138 ± 0.007	0.081 ± 0.004	0.057 ± 0.003	0.047 ± 0.002
		1500	0.12 ± 0.004	0.071 ± 0.003	0.052 ± 0.002	0.041 ± 0.002
		2000	0.109 ± 0.004	0.065 ± 0.003	0.047 ± 0.001	0.039 ± 0.001
rFNO	augmentation	500	0.16 ± 0.003	0.145 ± 0.003	0.138 ± 0.002	0.136 ± 0.002
		1000	0.11 ± 0.003	0.102 ± 0.002	0.098 ± 0.002	0.094 ± 0.002
		1500	0.092 ± 0.002	0.085 ± 0.002	0.08 ± 0.001	0.077 ± 0.002
		2000	0.078 ± 0.002	0.072 ± 0.001	0.069 ± 0.001	0.065 ± 0.001
	resampling	500	0.176 ± 0.004	0.175 ± 0.004	0.174 ± 0.002	0.173 ± 0.002
		1000	0.125 ± 0.005	0.126 ± 0.001	0.122 ± 0.002	0.123 ± 0.002
		1500	0.104 ± 0.003	0.101 ± 0.002	0.1 ± 0.001	0.1 ± 0.001
		2000	0.089 ± 0.001	0.087 ± 0.001	0.087 ± 0.002	0.087 ± 0.002
DilResNet	augmentation	500	0.132 ± 0.012	0.113 ± 0.01	0.116 ± 0.014	0.105 ± 0.009
		1000	0.093 ± 0.008	0.091 ± 0.009	0.068 ± 0.005	0.068 ± 0.003
		1500	0.077 ± 0.008	0.066 ± 0.009	0.061 ± 0.006	0.055 ± 0.006
		2000	0.061 ± 0.007	0.053 ± 0.004	0.05 ± 0.003	0.055 ± 0.004
	resampling	500	0.169 ± 0.019	0.174 ± 0.013	0.147 ± 0.009	0.155 ± 0.009
		1000	0.098 ± 0.007	0.105 ± 0.01	0.112 ± 0.008	0.11 ± 0.009
		1500	0.09 ± 0.011	0.082 ± 0.005	0.076 ± 0.003	0.079 ± 0.005
		2000	0.065 ± 0.003	0.07 ± 0.006	0.068 ± 0.006	0.066 ± 0.006
MLP	augmentation	500	0.395 ± 0.09	0.277 ± 0.013	0.286 ± 0.016	0.243 ± 0.015
		1000	0.259 ± 0.049	0.169 ± 0.025	0.144 ± 0.022	0.131 ± 0.017
		1500	0.172 ± 0.02	0.138 ± 0.034	0.098 ± 0.009	0.085 ± 0.005
		2000	0.112 ± 0.014	0.084 ± 0.007	0.076 ± 0.005	0.069 ± 0.007
	resampling	500	0.358 ± 0.051	0.387 ± 0.053	0.383 ± 0.057	0.401 ± 0.054
		1000	0.288 ± 0.122	0.224 ± 0.018	0.202 ± 0.029	0.233 ± 0.049
		1500	0.178 ± 0.011	0.155 ± 0.015	0.14 ± 0.031	0.136 ± 0.011
		2000	0.148 ± 0.032	0.119 ± 0.018	0.103 ± 0.01	0.111 ± 0.013

Table 11. Relative errors for DeepONet, Navier-Stokes dataset, $\sqrt{}$ marks training run with augmentation and $\times$ — without augmentation.

model	v^1				v^2			
	E_{train}		E_{test}		E_{train}		E_{test}	
	$\times$	$\sqrt{}$	$\times$	$\sqrt{}$	$\times$	$\sqrt{}$	$\times$	$\sqrt{}$
DeepONet	0.074	0.063	0.162	0.163	0.212	0.232	0.368	0.395
POD-DeepONet	0.622	0.622	0.589	0.589	0.349	0.358	0.414	0.419