Mesh-Independent Operator Learning for Partial Differential Equations

Seungjun Lee [1]

Abstract

Operator learning, learning the mapping between function spaces, has been attracted as an alternative approach to traditional numerical methods to solve partial differential equations. In this paper, we propose to represent the discretized system as a set-valued data without a prior structure and construct the permutation-symmetric model, called mesh-independent neural operator (MINO), to provide proper treatments of input functions and query coordinates of the solution function. Our models pre-trained with a benchmark dataset of operator learning are evaluated by downstream tasks to demonstrate the generalization abilities to varying discretization formats of the system, which are natural characteristics of the continuous solution of the PDEs.

1. Introduction

Partial Differential equations (PDEs) are one of the most successful mathematical tools for representing the physical systems with governing equations over infinitesimal segments of the domain of interest given some problem-specific boundary conditions or forcing functions (Mizohata, 1973). PDEs, globally shared on the entire domain, are interpreted as how to interact between infinitesimal segments with respect to their geometric structure and values. Because of the universality on the entire domain, the system can be analyzed in a continuous way with respect to system inputs and outputs. In general, identifying appropriate governing equations for unknown systems is very challenging without domain expertise, however, there still remain numerous unknown processes for many complex systems. Even if knowing the governing equation of the system, it requires unnecessarily lots of time and memory costs to solve with conventional numerical methods, and sometimes it is intractable to compute in a complex and large-scale system.

In recent years, operator learning, an alternative to the conventional numerical methods, has been getting attention, pursuing to learn a mapping between infinite-dimensional input/output function space in a purely data-driven way without any human efforts or problem-specific knowledge of the system (Nelsen & Stuart, 2021; Li et al., 2020; 2021b; Lu et al., 2019; 2021; Gupta et al., 2021; Cao, 2021; Kovachki et al., 2021). Intuitively, for underlying PDE, $\mathcal{L}_a u = f$ defined on the continuous bounded domain, Ω, with system parameters $a \in \mathcal{A}$, forcing function $f \in \mathcal{F}$, and solution of the system $u \in \mathcal{U}$, the goal of the operator learning is to approximate the inverse operator $\mathcal{G} = \mathcal{L}_a^{-1} f : \mathcal{A} \to \mathcal{U}$ or $\mathcal{G} : \mathcal{F} \to \mathcal{U}$ with parametric model $\mathcal{G}_\theta$. Then, without loss of generality, in the case of input being a, the output function can be computed by $u = \mathcal{G}_\theta(a)$. Since the operator $\mathcal{G}_\theta$ should be able to capture pairwise and high-order interactions between elements of system inputs a to discover the governing PDEs, $\mathcal{G}_\theta$ has been approximated by a series of integral operators with parameterized kernels which iteratively update the system input to output for representing the global interactions between the elements (Nelsen & Stuart, 2021; Li et al., 2020; 2021b; Gupta et al., 2021; Cao, 2021; Kovachki et al., 2021). In practice, continuous measurements of the input/output functions are infeasible, the observed data are provided as a set of input-output pairs which are point-wise finite discretization of the functions. Meanwhile, the output values at arbitrary query coordinate y can be expressed as $u(y) = [\mathcal{G}_\theta(a)](y)$ which can be viewed as the output of the processing $\mathcal{G}_\theta : \mathcal{A} \times Y \to \mathcal{U}$ from two-part of input placeholders, $a \in \mathcal{A}$ and $y \in Y$. Instead of simply mapping between the input-output pairs, the following two additional considerations should be considered for the model with respect to function a, and query coordinate y, which we call mesh-independent operator learning; (1) the output of the model should not depend on any discretization format of the a, and (2) the model should be able to output a solution at the arbitrary query coordinate y. The measurements of the system are often sparsely and irregularly distributed due to the geometries of the domain, environmental conditions, or occasionally inoperative equipment. Also, in popular numerical methods, such as finite element methods for solving the PDEs, unstructured meshes are often utilized for the discretization of the domain. Therefore, unlike fixed sensor locations or uniform grid-like discretization, without prior assumptions for data structure, the model should aggregate global information over the measurements

[1]Seoul National University, Seoul, South Korea. Correspondence to: Seungjun Lee <tl7qns7ch@snu.ac.kr>.

2nd AI4Science Workshop at the 39^{th} International Conference on Machine Learning (ICML), 2022.

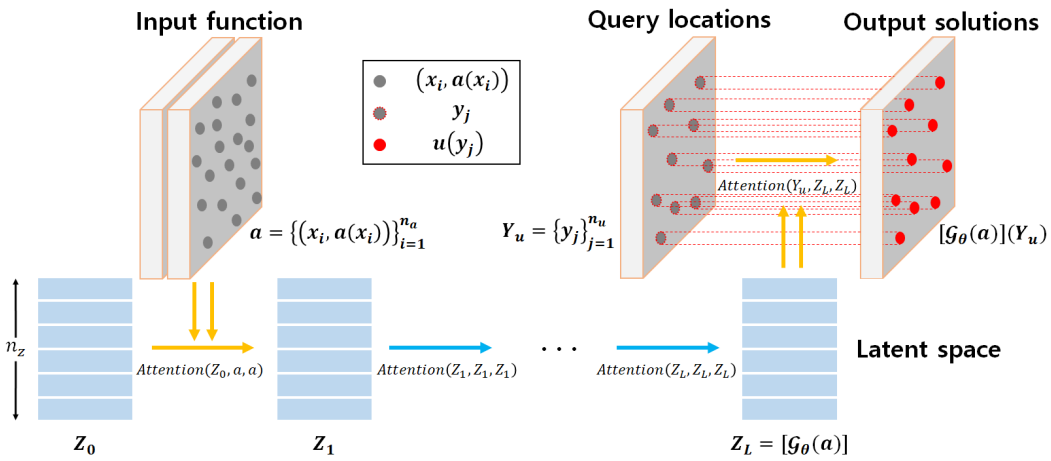

Figure 1. Mesh-independent neural operator.

in themselves to process the $[\mathcal{G}_\theta(a)]$ which then be reusable to any cardinality and permutation of the discretization of a. After that, the model should output the solution $[\mathcal{G}_\theta(a)](y)$ at any query coordinates y with encoded $[\mathcal{G}_\theta(a)]$.

Although there are two representative architectures for original operator learning problems, deep operator networks (DeepONets), and neural operators, they can not be applicable to mesh-independent operator learning. DeepONets are able to output a solution at any query coordinate y, but they use the fixed discretization of system inputs a (Lu et al., 2019; 2021). Neural operators can be adapted to the different discretization of the system inputs a, but the experiments are limited to uniform grid-like discretization and the solution of the model is not a function of query coordinate y where the same discretization of the domain is used for input and output space (Nelsen & Stuart, 2021; Li et al., 2020; 2021b; Gupta et al., 2021; Kovachki et al., 2021). There has been limited discussion on the generalization abilities for discretizations with extended variations such as irregular distributions and arbitrary permutations, which are natural characteristics of the continuous solution of PDEs.

Therefore, we treat the observational data as an unordered set without intrinsic data structure but intrinsically satisfying permutation symmetries when represented as feature vectors. Then, we build the mesh-independent neural operator (MINO) as presented in Figure 1 with a fully attentional architecture consisting of encoder-processor-decoder, inspired by the variants of Transformer to process data in modality-agnostic ways (Vaswani et al., 2017; Lee et al., 2019; Jaegle et al., 2021; 2022; Tang & Ha, 2021). The encoder encodes the system inputs a to latent space, the processor processes the pairwise and higher-order interactions between elements of the latent space, and the decoder decodes the latent space to output solutions at query coordinates y.

2. Approach

2.1. Preliminaries

Operator learning. Let the observed data be provided as N instance of input-output pairs, $\{(a_i, u_i)\}_{i=1}^N$, where $a_i = a_i|_{X_a}$ and $u_i = u_i|_{Y_u}$ are finite discretization of the domain for the input function $a_i \in \mathcal{A}$ at $X_a = \{x_1, ..., x_{n_a}\}$ and output function $u_i \in \mathcal{U}$ at $Y_u = \{y_1, ..., y_{n_u}\}$, with the number of discretized points n_a and n_u, respectively. The goal of the original operator learning is to optimize the following objective for learning a model $\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}$,

$$\min_\theta E_{a\sim\mu}[\mathcal{L}\left(\mathcal{G}_\theta(a_i), u_i\right)]. \tag{1}$$

where $a \sim \mu$ is iid on $\mathcal{A}$, and the discretizations of the input and output space are usually assumed to be the same (Nelsen & Stuart, 2021; Li et al., 2020; 2021b; Gupta et al., 2021; Cao, 2021; Kovachki et al., 2021), i.e., $X_a = Y_u$. The architectures for the operator learning, called neural operators, usually consist of lifting-iterative updates-projection, which are corresponding to encoder-processor-decoder, respectively. The lifting $v_1(x) = \mathcal{P}(a(x))$ and projection $u(x) = \mathcal{Q}(v_L(x))$ are local transformations usually implemented by element-wise feed-forward neural networks (Li et al., 2020; 2021b; Kovachki et al., 2021) or problem-specific convolutional modules (Cao, 2021) for mapping input features to target dimensional features. The iterative updates $\mathcal{G}_l : v_l \mapsto v_{l+1}, l \in [1, L-1]$ are global transformations implemented by sequence of kernel integral operations to capture the interactions between the elements.

Kernel integral operation and attention. Inspired by kernel integral operation for PDEs, the parametric model is usually constructed by a series of integral operators $\mathcal{G}_l$ with parameterized kernels for representing the global interac-

tions between the latent elements,

$$v_{l+1}(y) = [\mathcal{G}_l(v_l)](y) = \int_{\Omega_x} \mathcal{K}_l(x, y)v_l(x)dx, \quad (2)$$

where the parameterized kernel $\mathcal{K}_l$ defined on $\Omega_x \times \Omega_y$. The transform $\mathcal{G}_l$ can be interpreted as mapping a function $v_l(x)$ defined in domain $x \in \Omega_x$ to the function $v_{l+1}(y) = [\mathcal{G}_l(v_l)](y)$ defined in domain $y \in \Omega_y$. Recently, the kernel integral operation can be successfully approximated by the attention mechanism of Transformers (Cao, 2021; Kovachki et al., 2021; Guibas et al., 2021; Pathak et al., 2022). For the sake of simplicity, let input vectors $X \in R^{n_x \times d_x}$ and query vectors $Y \in R^{n_y \times d_y}$, then the attention can be expressed as

$$Att(Y, X, X) = \sigma(QK^T)V \approx \int_{\Omega_x} (q(y) \cdot k(x))v(x)dx, \quad (3)$$

where $Q = YW^q \in R^{n_y \times d_q}$, $K = XW^k \in R^{n_x \times d_q}$, $V = XW^v \in R^{n_x \times d_v}$, and σ are the query, key, value matrices, and softmax function, respectively. The attention mechanism, the weighted sum of V with the attention matrix $\sigma(QK^T)$, can be interpreted as the kernel integral operation in which the parameterized kernel is approximated by the attention matrix (Cao, 2021; Tsai et al., 2019; Xiong et al., 2021; Choromanski et al., 2021). Here, the number of rows n_x and n_y can be considered as the cardinality of the discretization of Ω_x and Ω_y, respectively. Also, the input vectors are projected to query embedding space by the attention mechanism, $Att(Y, X, X)$. Note that when $X = Y$, the mechanism denote the self-attention, $Att(X, X, X)$.

Permutation invariance and equivariance. For a function whose input can be represented by a set, there can be two interesting symmetries with respect to any permutations; permutation-invariance, and permutation-equivariance. Let F is the function with set-valued inputs $X = \{x_1, x_2, ..., x_n\}$, $x_i \in R^d$. Since deep learning models can not directly treat the set-valued input, the input data is provided as ordered vectors representing one of $n!$ permutations of X. In practice, with arbitrary permutation action π to the first dimension of the matrix-represented $X \in R^{n \times d}$, the permutation-invariance and equivariance are defined as,

$$\begin{aligned} F(\pi X) &= F(X), \quad \text{(permutation-invariance)}, \\ F(\pi X) &= \pi F(X), \quad \text{(permutation-equivariance)}, \end{aligned} \quad (4)$$

where the output of the permutation-invariant function does not depend on ordering and the cardinality of the input set, while the ordering and the cardinality of the output of the permutation-equivariant function remain consistent with those of the input set. A full mathematical definition and details of the properties can be found in (Zaheer et al., 2017; Murphy et al., 2019; Wagstaff et al., 2021; Maron et al., 2020). It can easily be proved that the output of attention mechanism $Att(Y, X, X)$ is permutation-invariant to X and permutation-equivariant to Y. The permutations acting on X are erased out of each other by the consecutive matrix multiplication, while that on Y remains to outputs. Additionally, it is certain that self-attention $Attention(X, X, X)$ is permutation-equivariant to X. Detailed explanations can be found in Appendix A.1.

2.2. Mesh-independent operator learning

Problem statement. Most conditions remain the same as the original operator learning, but we reconsider the representations of the discretized input-output pairs, $a_i = a_i|_{X_a}$ and $u_i = u_i|_{Y_u}$. Then, we consider the extended problem where the discretized points X_a and Y_u are allowed to be arbitrary on the domain along with varying the number of n_a and n_u for each i. In practice, we aim to build a model that is applicable to the randomly permuted and varying number of discretization points during testing, even if the model is trained with another arbitrary order of observations with smaller numbers. The goal of mesh-independent operator learning is to learn the model $\mathcal{G}_\theta : \mathcal{A} \times Y \to \mathcal{U}$ that is expected to make the following test errors small when training with objective 1,

$$\min_\theta E_{a\sim\mu} E_{X_a, Y_u}[\mathcal{L}([\mathcal{G}_\theta(a_i)](Y_u), u_i)], \quad (5)$$

where X_a and Y_u can be arbitrary on the domain. When the discretization of a_i is fixed, the problem can be considered as a problem such as (Lu et al., 2019; 2021), where the model cannot handle other discretization formats of the system inputs.

Set representations for the discretizations. Before starting to construct a model, we reconsider the representations of $a = a|_{X_a} \in R^{n_a \times d_a}$, $u = u|_{Y_u} \in R^{n_u \times d_u}$ and query coordinates $Y_u \in R^{n_u \times d}$. Since the representations should not include their own data structure, we treat a, u, and Y_u as the set representations, instead of treating them as structured arrays. For a, since our goal is to discover spatial relationships between the elements of a input set, the values of a are concatenated with position coordinates, $a = \{(x_1, a(x_1)), ..., (x_{n_a}, a(x_{n_a}))\} \in R^{n_a \times (d+d_a)}$ for indicating the value of $a(x)$ at position x, which compensate the positional information for representing continuous function as set representation. By the concatenated position coordinates for each value, the output is permutation-invariant to the representation of the input function a, when the set is represented as feature vectors, i.e.,

$$[\mathcal{G}_\theta(\pi a)](Y_u) = [\mathcal{G}_\theta(a)](Y_u). \quad (6)$$

Meanwhile, since the u is output of the model at the query coordinates $Y_u = \{y_1, ..., y_{n_u}\} \in R^{n_u \times d}$, only the values are used, $u = \{u(y_1), ..., u(y_{n_u})\} \in R^{n_u \times d_u}$. Since the

output $u(y)$ is a function of query coordinates, the permutation operation and the function operations are commutative. Therefore, the output is permutation-equivariant to the representation of the query coordinates Y_u, when the set is represented as vectors, i.e.,

$$[\mathcal{G}_\theta(a)](\pi Y_u) = \pi[\mathcal{G}_\theta(a)](Y_u). \tag{7}$$

Positional embeddings. Instead of using raw position coordinates $x_i, y_j \in R^d$, we concatenate the Fourier embeddings for the position coordinates, which is a common strategy to enrich the positional information (Vaswani et al., 2017; Mildenhall et al., 2020; Tancik et al., 2020; Sitzmann et al., 2020). The positional embeddings exploit sine and cosine functions with frequencies spanning from minimum to the Nyquist frequencies sufficiently covering the sampling rates for corresponding dimensions. This simple technique provides the model with the capability of representing fine-grained functions or wide-spectral components.

2.3. Architecture

Following (Lee et al., 2019; Jaegle et al., 2021; 2022; Tang & Ha, 2021), we build our model with a fully attention-based architecture consisting of encoder-processor-decoder with modality-agnostic encoder and decoder, called mesh-independent neural operator (MINO) as presented in Figure 1. As the critical difference of mesh-independent operator learning from the original one is treating a and y as the sets without prior assumption on the data structure, the significant modifications from the existing neural operators are mostly in the lifting (encoder) and projection (decoder), while we can use existing building blocks used as iterative updates in the neural operators to construct the processor. The encoder encodes the input function a to latent feature vectors satisfying permutation-invariance, the processor processes the pairwise and higher-order interactions between elements of the latent features vectors, and the decoder decodes the latent features to output solutions at a set of query coordinates Y_u satisfying permutation-equivariance. The model can also be viewed as continuous function regressor at query set of y conditioned on the encoded a which can be considered as support set (Finn et al., 2017) or context set (Kim et al., 2019).

Encoder. We use cross-attention module as the encoder to encode inputs $a \in R^{n_a \times (d+d_a)}$ to a smaller number n_z of learnable queries $Z_0 \in R^{n_z \times d_z}$ (typically $n_z < n_a$), then the result of the module is

$$Z_1 = \mathcal{G}_{enc}(a) = Att(Z_0, a, a) \in R^{n_z \times d_h}, \tag{8}$$

which is permutation-invariant to the elements of a and independent of the size of the input n_x. These properties, which make the model mesh-invariant to a, are a significant difference from the lifting component in the existing neural operators, which is not permutation-invariant and outputs the same size as the inputs n_x with element-wise transformation. The encoder is also interpreted as a projection inputs domain discretized by n_x elements to a latent domain consisting of the smaller number of n_z elements, which is called "inducing points" in (Lee et al., 2019) and reduces the computational complexity of the following attentional modules (Jaegle et al., 2021; 2022).

Processor. We use a series of self-attention modules as the processor each of which takes $z_l \in R^{n_z \times d_h}$ as the input of the query, key, and value components. Then the output of each self-attention module with $l \in [1, L-1]$ is

$$Z_{l+1} = \mathcal{G}_l(Z_l) = Att(Z_l, Z_l, Z_l) \in R^{n_z \times d_h}, \tag{9}$$

which is permutation-equivariant to the elements of Z_l, therefore the permutation-invariant property to the elements of a is preserved through successive modules. Also, the results Z_{l+1} have fixed discretization format with fixed ordering and number of elements n_z which is decoupled from discretization format of the input function n_a and output functions n_u. Due to the decoupling property, the whole architecture can not only capture the global interactions by the processor but is also applicable to mesh-independent operator learning independent of discretization formats of the input and output functions.

Decoder. We use cross-attention module as the decoder to decode the latent vectors from the processor $Z_L \in R^{n_z \times d_h}$ at query coordinates $Y_u \in R^{n_u \times d}$. Then the final output of the entire architecture is

$$[\mathcal{G}_\theta(a)](Y_u) = \mathcal{G}_{dec}(Z_L) = Att(Y_u, Z_L, Z_L) \in R^{n_u \times d_u}, \tag{10}$$

which is permutation-equivariant to the elements of Y_u, therefore every solution u_j corresponds to query coordinates y_j. This property makes the model applicable to any arbitrary discretization format of Y_u. Since the result from the processor is independent of the discretization format of input function a, the model is also applicable to any arbitrary discretization format of a.

3. Related Works

3.1. Neural networks for PDEs

There has been increasing interest in utilizing neural networks for PDEs that can be divided into several lines. In (Raissi et al., 2019; Sirignano & Spiliopoulos, 2018; Karniadakis et al., 2021), neural networks were used as approximations of the solution when given the boundary conditions and collocations points constrained to known governing equations of the system, called physics-informed neural networks. It can be an alternative to traditional PDE solvers, yielding mesh-independent and relatively high-fidelity solutions. However, these methods require full knowledge of

the PDEs, and the trained model is usually not reusable for new systems, otherwise requires expensive re-training for every new boundary condition. Meanwhile, several studies have been conducted to learn reusable governing operators for generalization to new systems. DeepONet pursues to learn the coefficients and basis of the operators by two sub-networks, respectively (Lu et al., 2019; 2021). The random feature models (Nelsen & Stuart, 2021) and message passings on graphs (Li et al., 2020) are used to approximate the governing integral operators. Fourier neural operator (FNO) utilizes fast Fourier transform to efficiently compute the integral operator in Fourier space (Li et al., 2021b). Furthermore, (Gupta et al., 2021) uses multi-wavelet transform to approximate the kernel of the operators. Close to our works, (Cao, 2021) use Transformer-style architectures to approximate the integral operation, but they use problem-specific feature extractors and decoding regressors which make the model biased toward specific equations. Most of these lines of work only focus on approximating the governing operator for PDEs mapping between two function spaces. However, there has been limited discussion on the representations of the discretizations of the system which are naturally mesh-independent and infinite-dimensional but represented by structured arrays for compatibility with modern deep learning models.

3.2. Set representation learning

In recent years, it has been witnessed that neural networks on sets gained attention starting from learning simple operations on unordered sets (e.g., sum or max) and processing the point clouds (Zaheer et al., 2017; Ravanbakhsh et al., 2016; Qi et al., 2017). The elements of the set are transformed into new representations through element-wise feed-forward neural networks which are permutation-equivariance. Then, the representation of the set is obtained by aggregation across the entire elements of the set by permutation-invariant functions, such as sum-pooling (Zaheer et al., 2017) or max-pooling (Qi et al., 2017). However, a theoretical study has been investigated to claim the limitations of the representation power of the methods (Wagstaff et al., 2019; Jurewicz & Derczynski, 2021). Additionally, the methods make it difficult for the model to learn pairwise and higher-order interactions between the elements of the sets, which are lost during those pooling operations. More recently, several groups have suggested more sophisticated models for treating set-valued inputs through modifying Transformers (Vaswani et al., 2017). Set Transformer uses a cross-attention module considered as learnable parameterized pooling operations to project set-valued input into smaller latent vectors, which ensure handling arbitrary cardinality and permutation of the inputs (Lee et al., 2019). Perceiver uses a similar strategy by emphasizing the efficiency of computational cost and the property of modality-agnostic which can be applied to a wide range of large-scale input modalities, such as images, point clouds, audio, and video (Jaegle et al., 2021). Furthermore, beyond simple outputs like classification, Perceiver IO uses another cross-attention module to decode complex outputs modalities which can be applied to the domains of language, optical flow, audio-visual sequence, and game environments (Jaegle et al., 2022). A similar strategy is also applied to reinforcement learning (Tang & Ha, 2021).

4. Experiments

Our experiments are conducted on several PDE benchmark to investigate the flexibility and generalization abilities of MINO through various downstream tasks. We select two representative models for original operator learning, FNO (Li et al., 2021b) and DeepONet (Lu et al., 2019) as the main baselines. While we follow their training procedures of (Li et al., 2021b), we extend the test sets with varying discretization formats of the test sets (originally given by equally spaced grids) with respect to the discretization points of a and u (or Y); (1) same grids, (2) super-resolutions on solutions u, (3) permuted grids of inputs a, (4) permuted and partially masked grids of inputs a.

First, (1) is the original settings of the existing operator learnings which evaluate the performance under supervised learning settings. Second, (2) is also discussed in previous works of both FNO and DeepONet (Li et al., 2021b; Lu et al., 2019) which evaluate the performance under zero-shot learning settings where the solutions are provided in unseen locations during training. However, the same discretization formats of inputs a are required during train and test time for DeepONet. Also, the same discretization formats with equally spaced grids of a and u are required for FNO. In order to remove the structural bias of the discretization of a and u, in (3) and (4), we randomly permute the elements of a, resulting in every a having different order. Additionally, some portions of inputs a in (4) are randomly masked, so that all observations of a are considered to be arbitrary sampled formats of a. (4) is also considered as continuous function regression $u(y)$ at query y conditioned on masked a which can be considered as few-shot samples of a (Finn et al., 2017; Kim et al., 2019). From the viewpoint of continuous function regression, the solutions $u(y)$ are evaluated on all given grids.

While results for experiments already discussed on these baselines are obtained from the related literature, results for the extended tasks that have not been discussed before are reproduced from their original codes. Note that the n/a result will occur when the baselines cannot be applicable for some downstream tasks. The implementation details and more experimental results are described in the Appendix A.3, A.4.

Table 1. Relative L^2 errors on Burgers' equation under different settings.

	Train grids	Test grids		Models		
	a, u	a	u	DeepONet	FNO	MINO
(1)	1024	1024	1024	0.1582	0.0160	**0.0104**
(2)		1024	8192	0.1584	n/a	**0.0106**
(2)		8192	8192	n/a	0.0139	**0.0090**
(3)		1024, perm	1024	0.9914	0.9852	**0.0104**
(4)		512 (50% mask), perm	1024	n/a	n/a	**0.0479**

Table 2. Relative L^2 errors on Darcy flow under different settings.

	Train grids	Test grids		Models		
	a, u	a	u	DeepONet	FNO	MINO
(1)	85×85	85×85	85×85	0.0765	**0.0108**	0.0172
(2)		85×85	421×421	0.0766	n/a	**0.0173**
(2)		421×421	421×421	n/a	**0.0098**	0.0170
(3)		85×85, perm	85×85	0.2532	0.2978	**0.0172**
(4)		3612 (50% mask), perm	85×85	n/a	n/a	**0.0272**

4.1. Burgers' equation

First, we consider a benchmark problem of 1D Burgers' equation which is a non-linear parabolic PDE combining the terms of convection and diffusion. The equation with periodic boundary conditions is

$$\partial_t u(x,t) + \partial_x(u^2(x,t)/2) = \nu \partial_{xx} u(x,t),$$
$$u(x,0) = u_0(x),$$

where $u_0 \sim \mu$ is the initial state generated from $\mu = \mathcal{N}(0, 625(-\Delta + 25I)^{-2})$ and $\nu = 0.1$ is the viscosity coefficient. The goal of operator learning is to learn mapping the initial state to the solution at time one, $\mathcal{G} : u_0 \mapsto u(\cdot, 1)$.

The quantitative results for Burgers' equation are shown in Table 1. It is shown that MINO outperforms the others not only in original tasks for operator learning, (1) same grids, and (2) super-resolution tasks but also in extended tasks, (3) permuted input, and (4) permuted and partially masked input tasks. In particular, in (3), the performances of baselines are drastically degraded by the permutation operations to input elements which destroy the structural bias of the discretization, while MINO is not affected. Even more, in (4), MINO can robustly cope with the discretized inputs of reduced size through random masking operations, while the baselines are not applicable to this task. Additionally, Figure 2 visualizes the ground truth output solutions (green lines) and the predictions of MINO at fine-grained query locations (red dashed lines) given the varying size of discretized inputs (blue points) for Burgers' equation. This visualization is reminiscent of few-shot regression problems $u(y)$ at query locations y where the support set is given by relatively few samples of discretized inputs a which should be mapped to output space u through series of integral trans-

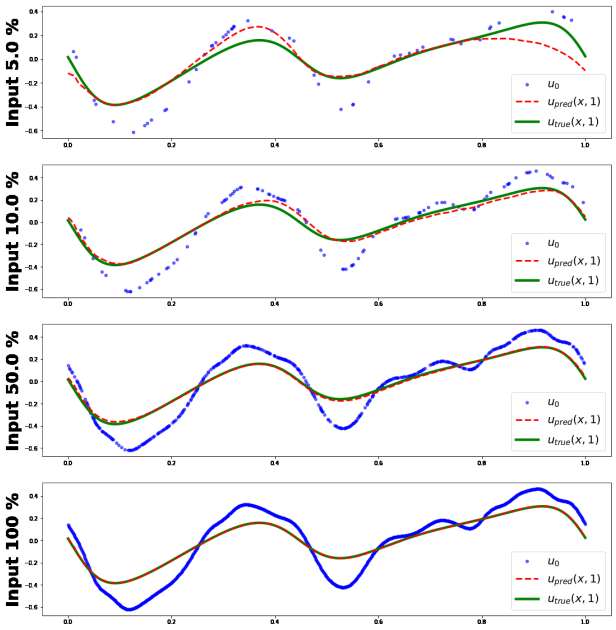

Figure 2. Ground truth solutions (green lines) and predictions of MINO at query locations (red dashed lines) given the varying size of discretized inputs (blue points) for Burgers' equation.

forms. As presented, the more shots are given, the closer the predictions are to the ground truth.

4.2. Darcy flow

Second, we consider another benchmark problem of 2D steady-state Darcy flow which is a second-order elliptic PDE describing the flow of fluid through a porous medium. The equation of Darcy flow on the unit box is

$$-\nabla \cdot (a(x)\nabla u(x)) = f(x),$$
$$u(x) = 0,$$

where u is density of the fluid, $a \sim \mu$ is the diffusion field generated from $\mu = \mathcal{N}(0, (-\Delta + 9I)^{-2})$ with fixed forcing function $f = 1$. The goal of operator learning is to learn mapping the diffusion field to the solution of the density, $\mathcal{G} : a \mapsto u$.

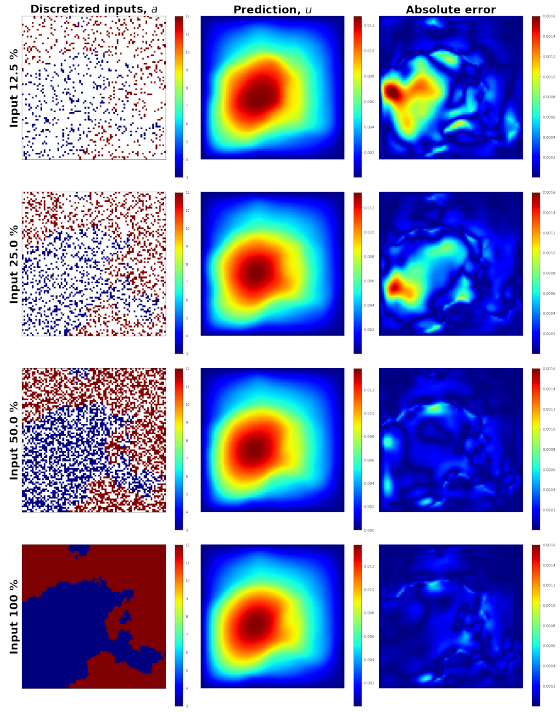

Figure 3. Discretized inputs with varying sizes (left column), their corresponding predictions of MINO at query locations (middle column), and absolute error with the ground truth (right column) for Darcy flow.

The quantitative results for Darcy flow are shown in Table 2. It is shown that MINO shows slightly worse but competitive performances than FNO in the original tasks, (1) and (2). This is because FNO explicitly exploits structural bias of the 2D-structured array with equally spaced grids, while MINO do not use structural bias but implicitly learn it through the Nerf-style positional encodings (Vaswani et al., 2017; Mildenhall et al., 2020; Tancik et al., 2020; Sitzmann et al., 2020). However, in extended tasks, (3) and (4), MINO outperforms the others due to the same reason explained at those of Burgers' equation. Additionally, Figure 3 visualizes discretized inputs with varying sizes (left column), their corresponding predictions of MINO at fine-grained query locations (middle column), and absolute error with the ground truth (right column) for Darcy flow. As also presented, MINO is robust under arbitrary discretization formats with varying sizes, even for few samples.

4.3. Navier-Stokes equation

Third, we consider another benchmark problem of 2D Navier-Stokes equation describing the dynamics of a viscous, incompressible fluid. The equation in vorticity form on the unit torus is

$$\partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) = \nu \Delta w(x,t) + f(x),$$
$$\nabla \cdot u(x,t) = 0,$$
$$w(x,0) = w_0(x),$$

where u is the velocity field, $w = \nabla \times u$ is the vorticity field, $w_0 \sim \mu$ is the initial vorticity field generated from $\mu = \mathcal{N}(0, 7^{3/2}(-\Delta + 49I)^{-2.5})$ with periodic boundary conditions, ν is the viscosity coefficient and forcing function is kept $f(x) = 0.1\left(\sin\left(2\pi\left(x_1 + x_2\right)\right) + \cos\left(2\pi\left(x_1 + x_2\right)\right)\right)$.

We assume that the system is under Markov process that the vorticity field at time $t + dt$ is described as $w_{t+dt} = \mathcal{G}_{dt} \cdot w_t$, where the operator $\mathcal{G}_{dt}$ during unit time step dt is not depend on time (Li et al., 2021a). When we set the unit time step $dt = 1$, the vorticity field at time t is denoted as $w_t = \mathcal{G}_{dec} \cdot [\mathcal{G}_{t-1} \cdots \cdot \mathcal{G}_1] \cdot \mathcal{G}_{enc}(w_0)$ where $\mathcal{G}_{enc}$ and $\mathcal{G}_{dec}$ are the encoder and decoder implemented by cross-attentional modules, respectively, and $\mathcal{G}_{dt} = \mathcal{G}_1 = \cdots = \mathcal{G}_{t-1}$ are identical processors implemented by the same self-attentional modules. The goal of operator learning is to learn mapping the initial vorticity up to time T, $\mathcal{G} : w_0|_{(0,1)^2} \mapsto w|_{(0,1)^2 \times (0,T]}$. The models are trained with minimizing the object $E_{w_0 \sim \mu}\left[\frac{1}{T}\sum_{t=0}^{T-1} \mathcal{L}(\mathcal{G}_{dec} \cdot \mathcal{G}_{dt} \cdot \mathcal{G}_{enc}(w_t), w_{t+1})\right]$. The quantitative results for the Navier-Stokes equation are shown in Table 3. Due to the problem-specific structural bias, FNO shows better performance in the original tasks, but MNIO shows better performance and is applicable in more generalized tasks, (3) and (4). Additionally, Figure 4 visualizes that discretized initial vorticity field with the reduced size is encoded and processed iteratively with the same processors in latent space. The latent features are decoded to output the predictions of vorticity field at the corresponding time.

5. Conclusion

In this work, we raise potential issues with existing operator learning models for PDEs in the perspective of discretiza-

Table 3. Relative L^2 errors on Navier-Stokes equation under different settings.

	Train grids	Test grids		Models	
	a, u	a	u	FNO	MINO (ours)
(1)	64×64	64×64	64×64	**0.0110**	0.0349
(3)		64×64, perm	64×64	0.7106	**0.0349**
(4)		2048 (50% mask), perm	64×64	n/a	**0.0544**

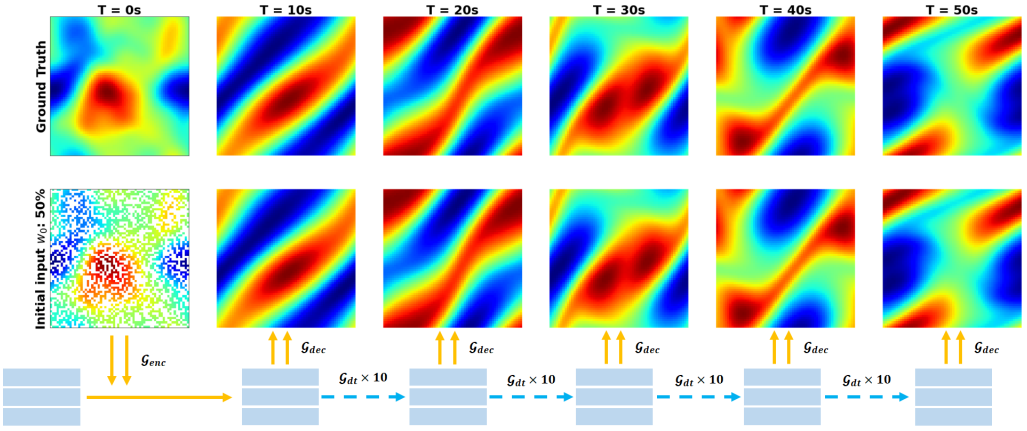

Figure 4. Discretized input with reduced size is encoded, processed iteratively, and decoded to output the predictions of vorticity field for the Navier-Stokes equation at the corresponding time.

tion for the continuous input/output functions of the systems when the observations are irregular and have discrepancies between training and testing measurement formats. Discretized functions are treated as set-valued data without prior data structure but with permutation symmetric modules and concatenated positional encodings to access positional structure with compact and continuous queries. To solve the issues, we propose MNIO constructed by fully-attentional modules and evaluate it to original tasks and extended downstream tasks compared with other existing representative models. The results show that our model is not only competitive in original operator learning tasks but also robustly applicable in extended tasks which are natural consequences of continuous solutions for physical systems but not compatible with existing representative models.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Cao, S. Choose a transformer: Fourier or galerkin. *Advances in Neural Information Processing Systems*, 34, 2021.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Guibas, J., Mardani, M., Li, Z., Tao, A., Anandkumar, A., and Catanzaro, B. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

Gupta, G., Xiao, X., and Bogdan, P. Multiwavelet-based operator learning for differential equations. *Advances in Neural Information Processing Systems*, 34, 2021.

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

Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., and Carreira, J. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pp. 4651–4664. PMLR, 2021.

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., Henaff, O. J., Botvinick, M., Zisserman, A., Vinyals, O., and Carreira, J. Perceiver IO: A general architecture for structured inputs & outputs. In

International Conference on Learning Representations, 2022.

Jurewicz, M. and Derczynski, L. Set-to-sequence methods in machine learning: a review. *Journal of Artificial Intelligence Research*, 71:885–924, 2021.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

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

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.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pp. 3744–3753. PMLR, 2019.

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, 2020.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Markov neural operators for learning chaotic systems. *arXiv preprint arXiv:2106.06898*, 2021a.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021b.

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, 2021.

Maron, H., Litany, O., Chechik, G., and Fetaya, E. On learning sets of symmetric elements. In *International Conference on Machine Learning*, pp. 6734–6744. PMLR, 2020.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

Mizohata, S. *The theory of partial differential equations*. CUP Archive, 1973.

Murphy, R., Srinivasan, B., Rao, V., and Riberio, B. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019.

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

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.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

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

Ravanbakhsh, S., Schneider, J., and Poczos, B. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*, 2016.

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

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron,

J., and Ng, R. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33, 2020.

Tang, Y. and Ha, D. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Tsai, Y.-H. H., Bai, S., Yamada, M., Morency, L.-P., and Salakhutdinov, R. Transformer dissection: A unified understanding of transformer's attention via the lens of kernel. *arXiv preprint arXiv:1908.11775*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wagstaff, E., Fuchs, F., Engelcke, M., Posner, I., and Osborne, M. A. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pp. 6487–6494. PMLR, 2019.

Wagstaff, E., Fuchs, F. B., Engelcke, M., Osborne, M. A., and Posner, I. Universal approximation of functions on sets. *arXiv preprint arXiv:2107.01959*, 2021.

Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., and Singh, V. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, volume 35, pp. 14138. NIH Public Access, 2021.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in neural information processing systems*, 30, 2017.

A. Appendix

A.1. Permutation-symmetric property of attention

Here, we attempt to explain that the attention module $Attention(Y, X, X)$ in Equation 3 is permutation-invariant to X and permutation-equivariant to Y. Let the permutation matrix to the element (first) dimension, $P \in R^{n\times n}$. First, when the permutation matrix P is multiplied to the X, the output of the attention is

$$\begin{aligned} Attention(Y, PX, PX) &= \sigma\left(QK^T\right)V \\ &= \sigma\left(YW^q\left(PXW^k\right)^T\right)PXW^v \\ &= \sigma\left(YW^q\left(XW^k\right)^T P^T\right)PXW^v \\ &= \sigma\left(YW^q\left(XW^k\right)^T\right)\left(P^TP\right)XW^v \\ &= \sigma\left(YW^q\left(XW^k\right)^T\right)XW^v \\ &= Attention(Y, X, X), \end{aligned} \tag{11}$$

which states that the $Attention(Y, X, X)$ is permutation-invariant to X. Also, when the permutation matrix P is multiplied to the Y, the output of the attention is

$$\begin{aligned} Attention(PY, X, X) &= \sigma\left(QK^T\right)V \\ &= \sigma\left(PYW^q\left(XW^k\right)^T\right)XW^v \\ &= P\sigma\left(YW^q\left(XW^k\right)^T\right)XW^v \\ &= PAttention(Y, X, X), \end{aligned} \tag{12}$$

which states that the $Attention(Y, X, X)$ is permutation-equivariant to Y. Thus, it is easily proved that self-attention $Attention(X, X, X)$ is permutation-equivariant to X,

$$Attention(PX, PX, PX) = PAttention(X, PX, PX) = PAttention(X, X, X). \tag{13}$$

A.2. Attention modules

Following (Jaegle et al., 2021; 2022), mesh-independent neural operator (MINO) consists of two types of attention modules, cross- and self-attention modules, which implement the respective attention mechanisms. The attention modules have the following shared structure, which takes two input arrays, a query input $Y \in R^{n_y\times d_y}$ and a key-value input $X \in R^{n_x\times d_x}$,

$$\begin{aligned} O = Y + Att(LayerNorm(Y), LayerNorm(X), LayerNorm(X)), \\ Attention(Y, X, X) = O + FF(LayerNorm(O)), \end{aligned} \tag{14}$$

where $LayerNorm$ is layer normalization (Ba et al., 2016), FF consists of two channel-wise feedforward neural networks with a GELU nonlinearity (Hendrycks & Gimpel, 2016), and the exact calculation of attention array Att is

$$Att(X^q, X^k, X^v) = softmax\left(\frac{QK^T}{\sqrt{d_q}}\right)V, \tag{15}$$

where $Q = X^qW^q \in R^{n_y\times d_q}$, $K = X^kW^k \in R^{n_x\times d_q}$, and $V = X^vW^v \in R^{n_x\times d_v}$ for a single headed attention. In the case of multi-headed attention, several outputs from different learnable parameters are concatenated and projected with the linear transformation. For avoiding confusion, the Equation 3 is the simplification of the attention module in Equation 14.

A.3. Implementation details

Burgers' equation. For the positional encodings, equally spaced 64 frequency from min 1 to max 64 are used. For the encoder, a 8-headed cross-attention module is used, and the number of the elements (first) and channel (second) dimension

of latent space are 256 and 64, respectively. For the processor, a 8-headed self-attention module with the same channel dimension of latent's is used. For the decoder, a 8-headed cross-attention module with the channel dimension of 64 is used.

Darcy flow. For the positional encodings, equally spaced 32 frequency bins from min 1 to max 64 for x_1, x_2 of the diffusion field $a(x_1, x_2)$ are used. For the encoder, a single-headed cross-attention module is used, and the number of the elements (first) and channel (second) dimension of latent space are 256 and 64, respectively. For the processor, four 8-headed self-attention modules with the same channel dimension of latent's are used. For the decoder, a single-headed cross-attention module with the channel dimension of 64 is used.

Navier-Stokes equation. For the positional encodings, equally spaced 12 frequency bins from min 1 to max 32 for x_1, x_2 of the vorticity field at each time $w(x_1, x_2)$ are used. For the encoder, a single-headed cross-attention module is used, and the number of the elements (first) and channel (second) dimension of latent space are 128 and 64, respectively. For the processor, two 8-headed self-attention modules with the same channel dimension of latent's are used. For the decoder, a single-headed cross-attention module with the channel dimension of 64 is used.

A.4. Additional Results

Original benchmarks on Burgers' equation and Darcy flow for different resolutions are presented in Figure 5, Table 4, and Table 5, where all of the results except ours are brought from (Li et al., 2021b).

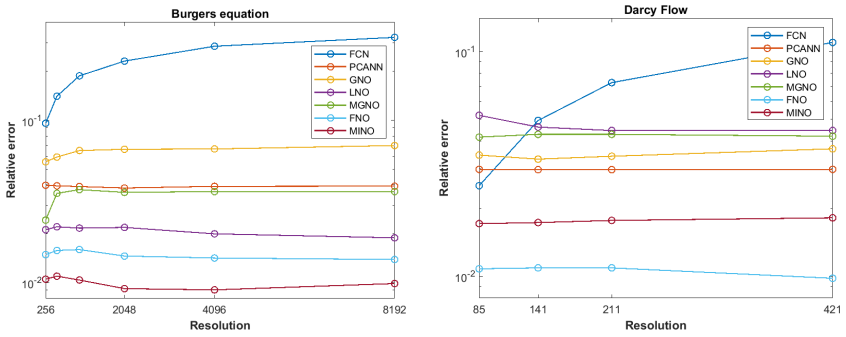

Figure 5. Benchmarks on (a) Burgers' equation, and (b) Darcy Flow for different resolutions.

Table 4. Benchmarks on 1D Burgers' equation

Models	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
FCN	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
GNO	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO	0.0149	0.0158	0.0160	0.0146	0.0142	0.0139
MINO (ours)	0.0105	0.0109	0.0104	0.0092	0.0090	0.0099

Table 5. Benchmarks on 2D Darcy Flow

Models	$s = 85$	$s = 141$	$s = 211$	$s = 421$
FCN	0.0253	0.0493	0.0727	0.1097
PCANN	0.0299	0.0298	0.0298	0.0299
GNO	0.0346	0.0332	0.0342	0.0369
LNO	0.0520	0.0461	0.0445	-
MGNO	0.0416	0.0428	0.0428	0.0420
FNO	0.0108	0.0109	0.0109	0.0098
MINO (ours)	0.0172	0.0173	0.0177	0.0182