# MESH-INDEPENDENT OPERATOR LEARNING FOR PDES USING SET REPRESENTATIONS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Operator learning, learning the mapping between infinite-dimensional function spaces, has been attracted as an alternative approach to traditional numerical methods to solve partial differential equations (PDEs). In practice, the functions of the physical systems are often observed by sparse or even irregularly distributed measurements, thus the functions are discretized and usually represented by finite structured arrays, which are given as data of input-output pairs. Through training with the arrays, the solution of the trained models should be independent of the discretization formats of input and can be queried at any point continuously. Therefore, the architectures for operator learning should be flexibly compatible with arbitrary sizes and locations of the measurements, otherwise, it can restrict the scalability when the observations have discrepancies between measurement formats. In this paper, we propose to treat the discretized functions as set-valued data and construct an attention-based permutation-symmetric model, called mesh-independent neural operator (MINO), to provide proper treatments of input functions and query coordinates for the solution functions. Our models pre-trained with benchmark datasets 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). The governing PDEs, which are globally shared on the entire domain, are interpreted as how to interact between infinitesimal segments with respect to their geometrical structures 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.

**Motivation.** 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 spaces in a purely data-driven way without any problem-specific knowledge of the system (Nelsen & Stuart, 2021; Li et al., 2020a;b; 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 $\Omega$ 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 $\mathcal{G}_\theta$. 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., 2020a;b; 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 operating $\mathcal{G}_\theta : \mathcal{A} \times Y \to \mathcal{U}$ with two input placeholders, $a \in \mathcal{A}$ and $y \in Y$. Instead of simply mapping between the input-output pairs, the following two considerations could be considered for the model with respect to input function $a$, and query coordinate $y$; (1) output of the model should not depend on any discretization format of the $a$, and (2) 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 (Belbute-Peres et al., 2020; Lienen & Günnemann, 2022). Also, in popular numerical methods for solving the PDEs, such as finite element methods, unstructured meshes are often utilized for the discretization of the domain, and adaptive re-meshing schemes are commonly deployed where the region of domain needs different resolutions depending on the accuracy of the prediction (Brenner et al., 2008; Huang & Russell, 2010). Therefore, the model should aggregate global information over the measurements in themselves to process the $[\mathcal{G}_\theta(a)]$ which then be reusable to any cardinality and configuration of the discretization of $a$. After that, the model should output the solution $[\mathcal{G}_\theta(a)](y)$ at any query coordinates $y$.

**Limitations of existing architectures.** There are two representative existing architectures for operator learning, such as operator networks, and neural operators. Based on a framework of operator networks (Chen & Chen, 1995), (Lu et al., 2019) extended the architectures with modern deep networks, called deep operator networks (DeepONets), which is one of the successful architectures for operator learning. In order to process the input functions and query coordinates, DeepONets consist of two sub-networks, called branch network and trunk network, respectively. DeepONets are able to output a solution at any query coordinate $y$ from the trunk network, but they use the fixed discretization of system inputs $a$ from the branch network (Lu et al., 2019; 2021). Another promising framework is neural operators, which consists of several integral operators with parameterized kernels to map between infinite-dimensional function spaces. Neural operators have been originally implemented by graph neural networks (Li et al., 2020a;b), which can be adapted to different resolutions of the system inputs in a mesh-invariant way. However, the graph-based neural operator was limited due to its unstable issues and the costs for computing the integral operator (You et al., 2022; Li et al., 2021b). Fourier neural operators (FNO), which utilize fast Fourier transform (FFT) to efficiently compute the integral operators in Fourier space, have been widely used as successful models for operator learning in a range of science and engineering for their efficacy and accuracy. (Li et al., 2021b; Pathak et al., 2022; Yang et al., 2021). However, FNO is limited to uniform grid-like discretization due to the usage of FFT. Also, implemented architectures of those neural operators typically assume to have the same sampling points for input and output functions (Li et al., 2020a;b; 2021b; Kovachki et al., 2021; Lu et al., 2022), i.e. have not been formulated as a method decorrelating the discretization of system input and solution, leading to the solution of the neural operators not being flexibly queried at any coordinate $y$, such as implicit neural representations (Sitzmann et al., 2020). There has been limited discussion on the generalization abilities for discretizations with extended variations such as different sampling points for inputs and outputs functions with arbitrary numbers and irregular distributions.

**Contributions.** For these considerations, we treat the observational data as an unordered set without intrinsic data structure but intrinsically satisfying permutation symmetries when represented as feature vectors. The permutation symmetric properties of a set can be the remedy for the compatibility issues on existing neural operators in the perspective of cardinality and ordering of discretization formats (Zaheer et al., 2017; Klus et al., 2021; Zafar et al., 2021). 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 $y$. Finally, we experiment on benchmarks to verify whether our model is robustly applicable when discretizations are irregular distributed and arbitrary permuted, which are natural consequences of continuous solutions for physical systems but not compatible with existing ones.

## 2 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 (Li et al., 2020a;b; 2021b; Kovachki et al., 2021; Lu et al., 2022), i.e., $X_a = Y_u$. The architectures of neural operator, 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 for mapping input features to target dimensional features, and 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 $\mathcal{G}$ with parameterized kernels for representing the global interactions between the latent elements,

$$[\mathcal{G}(v)](y) = \int_{\Omega_x} \mathcal{K}(x, y)v(x)dx, \quad (x, y) \in \Omega_x \times \Omega_y, \tag{2}$$

where the parameterized kernel $\mathcal{K}$ defined on $\Omega_x \times \Omega_y$. The transform $\mathcal{G}$ can be interpreted as mapping a function $v(x)$ defined in domain $x \in \Omega_x$ to the function $[\mathcal{G}(v)](y)$ 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

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

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 $\sigma$ 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 $\Omega_x$ and $\Omega_y$, respectively. Also, the input vectors are projected to query embedding space by the attention mechanism, $Attention(Y, X, X)$. Note that when $X = Y$, the mechanism denote the self-attention, $Attention(X, X, X)$. The details of the attention module we used are described in Appendix A.2.

## 3 APPROACH

### 3.1 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 as small as possible when training with objective 1,

$$E_{a\sim\mu} E_{X_a, Y_u}[\mathcal{L}\left([\mathcal{G}_\theta(a_i)](Y_u), u_i\right)], \tag{4}$$

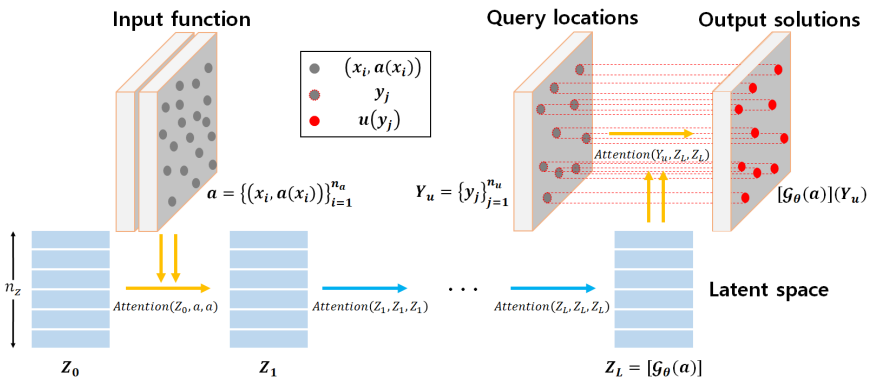


Figure 1: Mesh-independent neural operator.

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 represented by flattened array, 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.,

$$[\mathcal{G}_\theta(\pi a)](Y_u) = [\mathcal{G}_\theta(a)](Y_u). \tag{5}$$

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.,

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

**Positional embeddings.** Instead of using raw position coordinates for both inputs and outputs $x_i, y_j \in R^d$, 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 $K$ frequencies, $f_1, ..., f_K$, spanning from minimum to maximum frequencies sufficiently covering the Nyquist rates for corresponding dimension, resulting $[\sin(f_k \pi x), \cos(f_k \pi x)]$. This simple technique provides the model with the capability of representing fine-grained functions or wide-spectral components.

### 3.2 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

$$Z_1 = \mathcal{G}_{enc}(a) = Attention(Z_0, a, a) \in R^{n_z \times d_h}, \tag{7}$$

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

$$Z_{l+1} = \mathcal{G}_l(Z_l) = Attention(Z_l, Z_l, Z_l) \in R^{n_z \times d_h}, \tag{8}$$

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

$$[\mathcal{G}_\theta(a)](Y_u) = \mathcal{G}_{dec}(Z_L) = Attention(Y_u, Z_L, Z_L) \in R^{n_u \times d_u}, \tag{9}$$

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 $a$.

## 4 Related Works

### 4.1 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. A universal operator approximation theorem (Chen & Chen, 1995) was originally implemented by single-layer neural networks, and (Lu et al., 2019; 2021) develop the theorem by extending the architectures with current deep networks, called DeepONets, leading them to be more expressive. The random feature models (Nelsen & Stuart, 2021) and message passings on graphs (Li et al., 2020a;b) 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.

### 4.2 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., 2017a;b). 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. (2017a;b). 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).

## 5 EXPERIMENTS

Our experiments are conducted on several PDE benchmarks 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.

### 5.1 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

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

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, $\mathcal{G} : u_0 \mapsto u(\cdot, 1)$.

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

| | Train grids | Test grids | | Models | | |
|---|---|---|---|---|---|---|
| | $a$, $u$ | $a$ | $u$ | DeepONet | FNO | MINO (ours) |
| (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** |

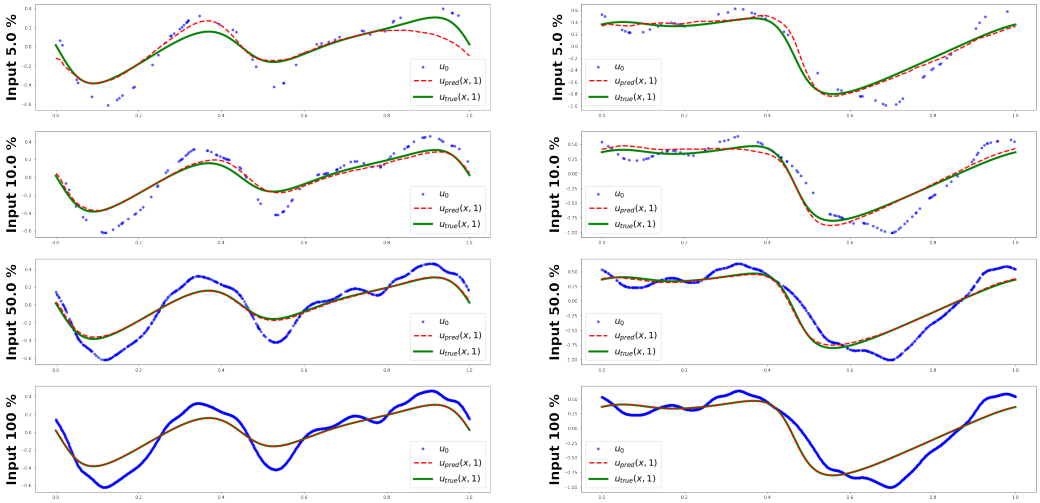


Figure 2: Ground truth solutions (green lines) and predictions of MINO at query locations (red dashed lines) given the varying size of discretized inputs (blue points) for Burgers' equation.

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 positional order 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 a series of integral transforms. As presented, the more shots are given, the closer the predictions are to the ground truth.

### 5.2 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

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

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, $\mathcal{G} : a \mapsto u$.

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

| | Train grids | Test grids | | Models | | |
|---|---|---|---|---|---|---|
| | $a$, $u$ | $a$ | $u$ | DeepONet | FNO | MINO (ours) |
| (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** |

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.

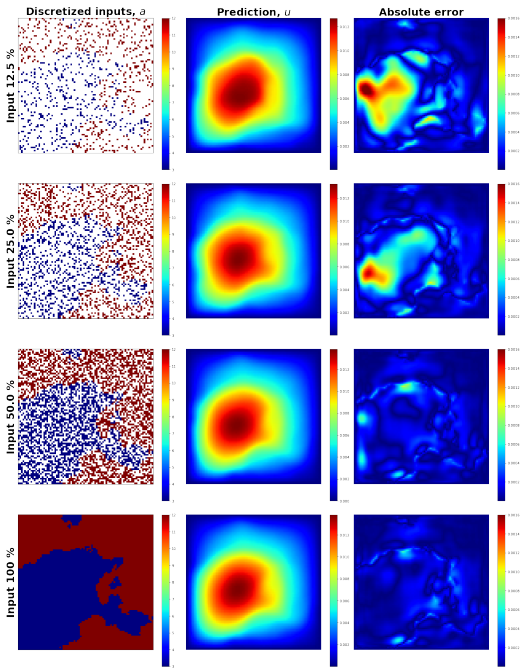


Figure 3: Discretized inputs with varying sizes (left), their corresponding predictions of MINO at query locations (middle), and absolute error with the ground truth (right) for Darcy flow.

### 5.3 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

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

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, $\nu$ is the viscosity coefficient and forcing function is kept $f(x) = 0.1\left(\sin\left(2\pi\left(x_1 + x_2\right)\right) + \cos\left(2\pi\left(x_1 + x_2\right)\right)\right)$.

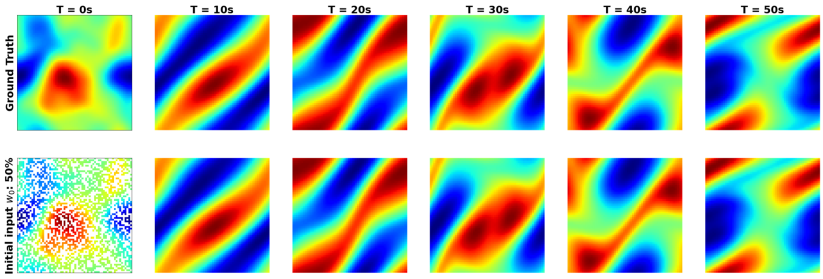


Figure 4: Discretized input with reduced size is encoded, processed iteratively, and decoded to output the predictions of vorticity field for the Navier-Stokes equation at the corresponding time.

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 $w_t = \mathcal{G}_{dec} \cdot [\mathcal{G}_{t-1} \cdot \cdots \cdot \mathcal{G}_1] \cdot \mathcal{G}_{enc}(w_0)$ 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 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 9. Due to the problem-specific structural bias, FNO shows better performance in the original tasks, but MINO 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.

## 6 Conclusions and Limitations

**Conclusions.** In this work, we raise potential issues with existing operator learning models for PDEs in the perspective of discretization 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 MINO 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.

**Limitations.** First, although our main contribution is constructing a neural operator to be flexible on measurement formats, the higher the dimension in which the measurement points can be placed, the performances of our model tested on gird-structured data get slightly worse than the problem-specific model (details in Appendix A.4), and the computational costs to obtain the flexibility grow exponentially (in Table 4). It could be an important direction for future work to address the efficiency and scaling issues for the neural operator. Second, the experimental setting is far from a real-world problems. Applications to realistic scenarios are an interesting direction for future work. For example, we might consider using a model trained on the dataset of Navier-Stokes to analyze the real fluid flow along complex geometries regarded as the same phenomena with different measurement formats.

## REFERENCES


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

Filipe De Avila Belbute-Peres, Thomas Economon, and Zico Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *international conference on machine learning*, pp. 2402–2411. PMLR, 2020.

Susanne C Brenner, L Ridgway Scott, and L Ridgway Scott. *The mathematical theory of finite element methods*, volume 3. Springer, 2008.

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

Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

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

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

Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. *Advances in Neural Information Processing Systems*, 34, 2021.

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

Weizhang Huang and Robert D Russell. *Adaptive moving mesh methods*, volume 174. Springer Science & Business Media, 2010.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pp. 4651–4664. PMLR, 2021.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J Henaff, Matthew Botvinick, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations*, 2022.

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

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

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

Stefan Klus, Patrick Gelß, Feliks Nüske, and Frank Noé. Symmetric and antisymmetric kernels for machine learning problems in quantum physics and chemistry. *Machine Learning: Science and Technology*, 2(4):045016, aug 2021.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pp. 3744–3753. PMLR, 2019.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Markov neural operators for learning chaotic systems. *arXiv preprint arXiv:2106.06898*, 2021a.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021b.

Marten Lienen and Stephan Günnemann. Learning the dynamics of physical systems from sparse observations with finite element networks. In *International Conference on Learning Representations*, 2022.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Haggai Maron, Or Litany, Gal Chechik, and Ethan Fetaya. On learning sets of symmetric elements. In *International Conference on Machine Learning*, pp. 6734–6744. PMLR, 2020.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

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

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

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

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 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, 2017a.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017b.

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

Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*, 2016.

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

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33, 2020.

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

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: A unified understanding of transformer's attention via the lens of kernel. *arXiv preprint arXiv:1908.11775*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pp. 6487–6494. PMLR, 2019.

Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Michael A Osborne, and Ingmar Posner. Universal approximation of functions on sets. *arXiv preprint arXiv:2107.01959*, 2021.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nystöm-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.

Yan Yang, Angela F Gao, Jorge C Castellanos, Zachary E Ross, Kamyar Azizzadenesheli, and Robert W Clayton. Seismic wave propagation and inversion with neural operators. *The Seismic Record*, 1(3):126–134, 2021.

Huaiqian You, Yue Yu, Marta D'Elia, Tian Gao, and Stewart Silling. Nonlocal kernel network (nkn): A stable and resolution-independent deep neural network. *Journal of Computational Physics*, 469: 111536, 2022. ISSN 0021-9991.

Muhammad I Zafar, Jiequn Han, Xu-Hui Zhou, and Heng Xiao. Frame invariance and scalability of neural operators for partial differential equations. *arXiv preprint arXiv:2112.14769*, 2021.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

# A APPENDIX

**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 $\pi$ to the first dimension of the matrix-represented $X \in R^{n \times d}$, the permutation-invariance and equivariance are defined as,

$$\begin{aligned} F(\pi X) &= F(X), \qquad \text{(permutation-invariance)}\ , \\ F(\pi X) &= \pi F(X), \qquad \text{(permutation-equivariance)}\ , \end{aligned} \tag{10}$$

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 $Attention(Y, X, X)$ 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.

## 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

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

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 Lee et al. (2019); 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 feed-forward 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 $Q = X^q W^q \in R^{n_y \times d_q}$, $K = X^k W^k \in R^{n_x \times d_q}$, and $V = X^v W^v \in R^{n_x \times d_v}$ 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 for readability.

### A.3 Implementation details

Table 3: Implementation details for training.

| System | Burgers' equation | Darcy flow | Navier-Stokes equation |
|---|---|---|---|
| Encoder | | | |
| Input function values, $a\|_{X_a} \in R^{n_a \times d_a}$ | 1024×1 | 7225×1 | 4096×1 |
| Positional embedding | 1024×129 | 7225×130 | 4096×50 |
| Frequency bins, $K$ | 64 | 32 | 12 |
| Number of heads | 4 | 1 | 1 |
| Inputs, $a \in R^{(d+d_a)}$ | 1024×130 | 7225×131 | 4096×51 |
| Processor | | | |
| Learnable queries, $Z_0 \in R^{n_z \times d_z}$ | 256×64 | 256×64 | 128×64 |
| Latent channels, $d_h$ | 64 | 64 | 64 |
| Number of heads | 8 | 8 | 8 |
| Number of blocks, $L$ | 1 | 4 | 2 |
| Decoder | | | |
| Query coordinates, $Y_u \in R^{n_u \times d}$ | 1024×1 | 7225×2 | 4096×2 |
| Positional embedding | 1024×129 | 7225×130 | 4096×50 |
| Outputs, $u \in R^{n_u \times d_u}$ | 1024×1 | 7225×1 | 4096×1 |

**Burgers' equation.** The positional embeddings at position coordinate $x \in [0, 1]$ from both input/output domain consists of $[x, \sin(f_k \pi x), \cos(f_k \pi x)]$, where $f_k$ are from equally spaced 64 frequencies from min 1 to max resolution 64 are used, resulting to have $129 = 1 + 2 \times 64$ dimensions. For the encoder, a 4-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 4-headed cross-attention module with the channel dimension of 64 is used.

**Darcy flow.** The positional embeddings at position coordinate $(x_1, x_2) \in [0, 1]^2$ of diffusion field $a(x_1, x_2)$ consists of $[x_1, x_2, \sin(f_k \pi x_1), \sin(f_k \pi x_2), \cos(f_k \pi x_1), \cos(f_k \pi x_2)]$, where $f_k$ are from equally spaced 32 frequencies from min 1 to max resolution 32 are used, resulting to have $130 = 2 \times (1 + 2 \times 32)$ dimensions. 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.** The positional embeddings at position coordinate $(x_1, x_2) \in [0, 1]^2$ of the vorticity field $w(x_1, x_2)$ consists of $[x_1, x_2, \sin(f_k \pi x_1), \sin(f_k \pi x_2), \cos(f_k \pi x_1), \cos(f_k \pi x_2)]$, where $f_k$ are from equally spaced 32 frequencies from min 1 to max resolution 32 are used, resulting to have $50 = 2 \times (1 + 2 \times 12)$ dimensions. 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.

**Optimization.** For all systems, the experiments are conducted on a GTX TITAN X GPU and we use Adam optimizer Kingma & Ba (2014) to train MINO for 500 epochs with initial learning rate of 0.001 which is halved every 100 epochs. Batch sizes for Burgers' equation and Navier-Stokes equations are set to 20, and for Darcy flow is set to 10.

**Computational complexity comparisons.** In Table 4, we compare MINO with the baselines in perspective of the training time, memory usage, and number of parameters on the benchmarks. The training time is measured during one epoch of each benchmarks training set, the memory usage indicates the GPU memory usage for the active Python process recorded from nvidia-smi command. The time and memory cost of MINO is on par with both baselines in the case of Burgers' equation but increases drastically compared with both baselines in the case of Darcy flow. By flattening the discretized inputs into a 1D sequence (identical point set) before processing the inputs and query coordinates, which removes any intrinsic locality of data, MINO is the modality-agnostic architecture that does not exploit the structural bias of data (such as a 2D structured array). Therefore, for instance, MINO regards the inputs of Darcy flow as 7,225 sizes of 1D sequence instead of $85\times85$ sizes of 2D structured data, which leads to exponential growth computational cost compared to encoding 1D data such as inputs of Burgers' equations. Exponentially more resources are required for MINO to have the capability to handle the more highly structured data in a modality-agnostic way for flexible input formats and query coordinates. Meanwhile, while attention-based architecture consists of QKV modules where the parameters are shared across the input elements, kernel-based architecture needs a large number of parameters when the dimension of data increases due to the dimension of the kernel increasing along the dimension of data. Therefore, although the number of parameters of FNO is smaller than MINO in the case of Burgers' equations, it becomes much larger than those of MINO. In summary, MINO has fewer parameters but becomes more expressive when paying a large amount of computational cost for gaining the flexibility to data formats, while FNO has a huge number of parameters, which already include some prior data structure, requiring less training cost but could be overfitted to the data structure.

Table 4: Computational complexity comparisons with baselines.

| Model | Training time per epoch | Memory usage | Number of parameters |
|---|---|---|---|
| Burgers' equation | | | |
| DeepONet | 0.645 s | 1.34 GB | 107,405 |
| FNO | 0.871 s | 1.45 GB | 57,825 |
| MINO | 0.715 s | 1.63 GB | 125,204 |
| Darcy flow | | | |
| DeepONet | 0.732 s | 1.40 GB | 417,550 |
| FNO | 1.881 s | 1.96 GB | 1,188,353 |
| MINO | 8.843 s | 5.68 GB | 401,514 |

### A.4 Additional Results

**Original benchmarks.** Original benchmarks on Burgers' equation and Darcy flow for different resolutions are presented in Figure 5, Table 5, and Table 6, where all of the results except ours are brought from Li et al. (2021b).

**Performances according to input masking ratios.** Relative $L^2$ errors according to input masking ratios (%) on the extended tasks are presented in Figure 6 and Table 7. The discretized inputs are permuted and randomly masked with corresponding ratios. To the best of our knowledge, MINO is the only model which is robustly applicable to this extended tasks, but not compatible with existing operator learning models.

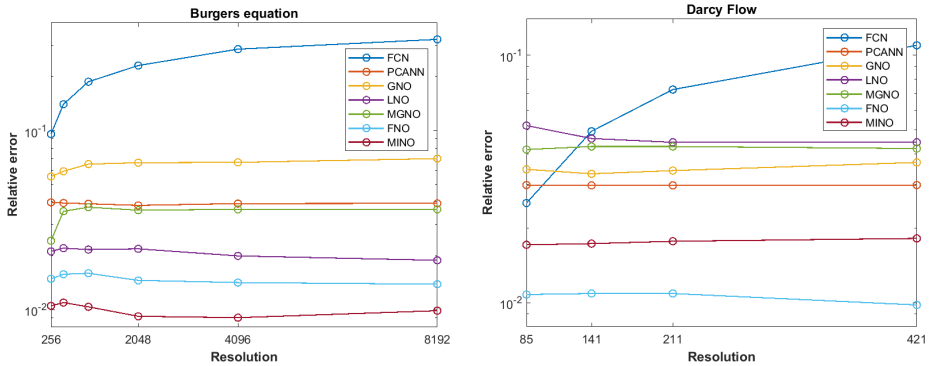


Figure 5: Benchmarks on (a) Burgers' equation, and (b) Darcy Flow for different resolutions.

Table 5: 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 6: 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 |

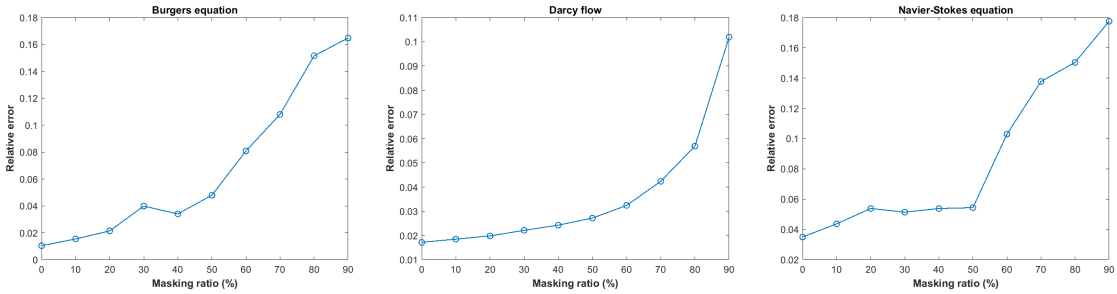


Figure 6: Relative $L^2$ errors according to input masking ratios (%) on (a) Burgers' equation, (b) Darcy Flow, and (c) Navier-Stokes equations.

Table 7: Relative $L^2$ errors according to input masking ratios (%).

| Masking ratio | 0% | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% |
|---|---|---|---|---|---|---|---|---|---|---|
| Burgers | 0.0104 | 0.0154 | 0.0214 | 0.0399 | 0.0340 | 0.0479 | 0.0809 | 0.1081 | 0.1516 | 0.1649 |
| Darcy flow | 0.0172 | 0.0185 | 0.0198 | 0.0221 | 0.0243 | 0.0272 | 0.0324 | 0.0424 | 0.0569 | 0.1019 |
| Navier-Stokes | 0.0349 | 0.0437 | 0.0538 | 0.0514 | 0.0538 | 0.0544 | 0.1030 | 0.1378 | 0.1504 | 0.1776 |

**Comparisons with other set-encoder.** The comparison model consists of encoder-processor-decoder like MINO, where encoder is replaced by Deepsets Zaheer et al. (2017) variant with average-pooling operations, while processor and decoder are the same as MINO, called Deepsets-variant. The performance of Deepsets-variants on Burgers equation is presented in Table 8, which shows not competitive compared to MINO. The outputs of the attention layer can be calculated by the weighted sum of attention and inputs, where the attention values have more expressive representation with extra learnable parameters $Z_0$ and inputs $a$, while the outputs of the existing pooling operation are calculated by fixed coefficient weighted sum (or operation) of inputs $a$ (in the case of avg-pooling, the weights are $\frac{1}{n_a}$). Therefore, from the perspective of the modeling operator, much of the information of interactions between pair-wise input elements is lost during those pooling operations.

Table 8: Comparisons with other set-encoder.

| | Train grids | Test grids | | Models | |
|---|---|---|---|---|---|
| | $a$, $u$ | $a$ | $u$ | Deepsets-variants | MINO (ours) |
| (1) | 1024 | 1024 | 1024 | 0.0179 | **0.0104** |
| (2) | | 1024 | 8192 | 0.0189 | **0.0106** |
| (2) | | 8192 | 8192 | 0.0183 | **0.0090** |
| (3) | | 1024, perm | 1024 | 0.0179 | **0.0104** |
| (4) | | 512 (50% mask), | 1024 | 0.0748 | **0.0479** |
| (4) | | 512 (50% mask), perm | 1024 | 0.0748 | **0.0479** |

Table 9: 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** |