Axial Neural Networks for Dimension-Free Foundation Models

Hyunsu Kim[†], **Jonggeon Park**[†], **Joan Bruna**[‡], **Hongseok Yang**[†], **Juho Lee**[†]
[†]KAIST, [‡]New York University,
{kim.hyunsu,parkjonggeon,hongseok.yang,juholee}@kaist.ac.kr
bruna@cims.nyu.edu

Abstract

The advent of foundation models in AI has significantly advanced general-purpose learning, enabling remarkable capabilities in zero-shot inference and in-context learning. However, training such models on physics data, including solutions to partial differential equations (PDEs), poses a unique challenge due to varying dimensionalities across different systems. Traditional approaches either fix a maximum dimension or employ separate encoders for different dimensionalities, resulting in inefficiencies. To address this, we propose a dimension-agnostic neural network architecture, the Axial Neural Network (XNN), inspired by parameter-sharing structures such as Deep Sets and Graph Neural Networks. XNN generalizes across varying tensor dimensions while maintaining computational efficiency. We convert existing PDE foundation models into axial neural networks and evaluate their performance across three training scenarios: training from scratch, pretraining on multiple PDEs, and fine-tuning on a single PDE. Our experiments show that XNNs perform competitively with original models and exhibit superior generalization to unseen dimensions, highlighting the importance of multidimensional pretraining for foundation models.

1 Introduction

The growing scale of deep learning models has led to the emergence of general-purpose AI systems, often called *foundation models*. Trained over a large amount of unlabeled data with self supervision, these models have shown impressive generalization performance, enabling effective zero-shot inference and in-context learning on a wide range of tasks. In practice, these models are further fine-tuned or post-trained for particular target tasks, achieving performance superior to models trained from scratch. A key requirement for developing highly-performing foundation models is the use of vast and diverse training data. In fact, the relationship between a model's performance and the scale of data (and models) is known to follow a version of power law, called scaling law [22, 18, 4].

This paper is concerned with developing key techniques for building successful foundation models for physics data, such as climate time-series data and solutions of partial differential equations (PDEs). When training such a model, we often have to combine datasets from multiple systems or differential equations that operate on different dimensionalities. For instance, the Burgers equation describing dissipative fluid flow is usually studied in one spatial dimension, whereas the Navier–Stokes equations are studied in two or three dimensions. Since the solutions of these equations are typically represented as tensors whose elements are points in spatial and temporal grids, different dimensionalities mean that those tensors storing solutions have different numbers of axes.

A straightforward way to address such mixed-dimensional scenarios is to fix a maximum dimension and either pad lower-dimensional inputs with zeros or build separate encoders for different dimensions that share the same output space. However, both approaches are inefficient for low-

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

dimensional data and incapable of handling inputs whose dimension exceeds the fixed maximum. Consequently, most of the prior works on PDEs have developed models tailored to a specific dimension (typically 2D) [17, 28, 35, 21]. Extending these models to other dimensions is nontrivial. For instance, the commonly-used patchify operation, which applies 2D convolution to extract 16×16 patches for Transformer models [12], is inherently limited to 2D inputs. Although a patchify operation can be designed for any specific dimension, a 2D patchifier is only applicable to 2D inputs; processing 1D or 3D data requires an entirely different set of parameters, as these models lack an intrinsic parameter-sharing mechanism across dimensions. While recent work has proposed dimension-agnostic methods based on neural processes [24] and multilayer perceptrons [25], these methods still incur large computational costs: they either flatten high-dimensional data and process long sequences, leading to expensive attention layers, or pre-compute dimension-equivariant weights via singular-value decomposition, which becomes infeasible in high dimensions.

We propose an efficient, dimension-agnostic, and dimension-generalizable neural network by adopting the core principle of parameter-sharing from permutation-equivariant architectures such as Deep Sets [38] and Graph Neural Networks [32, 15, 30], closely related to De Finetti's theorem [11, 38, 3]. Our architecture achieves permutation equivariance over tensor axes. Concretely, we treat the axes of a tensor as elements of a *set* and introduce a permutation-equivariant architecture which we refer to as the Axial Neural Network (XNN). Although such set-based XNN is simple and computationally efficient, we find it inherently less expressive, and thus further propose an advanced version termed graph-based XNN, which captures relationships among axes by treating the axes of a tensor as vertices of a graph. Finally, we introduce a dimension-agnostic PDE foundational model trained and evaluated on PDEs of varying dimensionality within a single model. Crucially, and in contrast to traditional patchify operations, an XNN-based patchify operation leverages this parameter sharing to make it applicable to inputs of any dimension without modification.

To demonstrate the expressivity and benefits of multidimensional pretraining, we convert existing PDE foundation models [28, 35] to the variants based on our XNNs. We evaluate the resulting models in three different settings: training a single PDE from scratch, pretraining with multiple PDEs, and fine-tuning on a single PDE. We show that our variants perform competitively with their original counterparts. We also conduct experiments to demonstrate the unseen-dimension generalization ability of XNNs, which plays an important role in such a dimension-agnostic strategy. Our XNN architecture shows better performance in unseen dimension fine-tuning, which underscores the necessity of multidimensional pretraining for foundation models. The implemented architectures are summarized in https://github.com/kim-hyunsu/XNN.

2 Backgrounds

2.1 Graph Neural Networks and Deep Sets

A **Graph Neural Network (GNN)** is a deep neural network designed to process and make predictions on data represented as a graph [30, 41]. GNNs are characterized by a message-passing mechanism, in which information is exchanged between nodes through their connections, called edges. Formally, given the feature vector $\boldsymbol{x}_a$ of node a, its hidden representation $\boldsymbol{h}_a$ is computed as

$$\boldsymbol{h}_a = \phi\left(\boldsymbol{x}_a, \bigoplus_{b \in \mathrm{ngbr}(a)} \psi(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{e}_{ab})\right), \tag{1}$$

where ϕ and ψ are neural networks with parameters shared across all nodes, $\bigoplus$ denotes a permutation-invariant aggregation operation, $\mathrm{ngbr}(a)$ is the set of neighbors of node a, and $\boldsymbol{e}_{ab}$ is the edge feature between nodes a and b. The critical architectural feature of a GNN is that the parameters of ϕ and ψ are shared across all nodes in the graph. This means the exact same functions are used to update each node's representation based on its local neighborhood.

This parameter-sharing structure is the fundamental reason GNNs are permutation-equivariant: permuting the input nodes simply changes the order of identical operations, leading to a corresponding permutation in the output. This symmetry is therefore structurally embedded in the model design, allowing the GNN to generalize across graphs of different sizes and structures.

A **Deep Set** is a neural network for set-structured data [38] and can be viewed as a special case of GNNs in which every node is equally connected to every other node. They can therefore be

expressed by the following simplification of Eq. 1:

$$\boldsymbol{h} = \phi\Big(\sum_{i=1}^{K} \psi(\boldsymbol{x}_i)\Big), \tag{2}$$

where K is the number of elements in the set. Similar to GNNs, the design relies on parameter sharing: the same function ψ is applied to every element $\boldsymbol{x}_i$ before a permutation-invariant aggregation (summation or maximum) is performed. This shared application of ψ ensures the model is permutation-invariant by construction.

2.2 Transpose and Axis-Permutation Equivariance

The transpose of a matrix flips the matrix over its diagonal, and the transpose of a tensor swaps two axis indices. Exchanging axes $i, j \in [1, K]$ of a rank-K tensor $\boldsymbol{x} = (x_{d_1 \cdots d_K})_{d_1,\ldots,d_k}$ yields

$$\boldsymbol{x}^{\top_{ij}}_{d_1 \cdots d_i \cdots d_j \cdots d_K} := x_{d_1 \cdots d_j \cdots d_i \cdots d_K}, \tag{3}$$

where $\top_{ij}$ denotes the transpose between axes i and j. An axis permutation is obtained by cumulative transposes, corresponding to reordering the axes. For a rank-4 tensor $\boldsymbol{x}$, for example,

$$\Pi(\boldsymbol{x})_{d_1 d_2 d_3 d_4} = x_{\,d_{\pi(1)}\, d_{\pi(2)}\, d_{\pi(3)}\, d_{\pi(4)}} = x_{d_3 d_2 d_4 d_1}, \tag{4}$$

where π is the permutation associated with Π.

Equivariance is the property that a function commutes with the action of a symmetry group G. If $\mathcal{X}$ and $\mathcal{Y}$ are acted on by $\rho_{\mathcal{X}}(g)$ and $\rho_{\mathcal{Y}}(g)$ for each group element $g \in G$, respectively, then $\phi : \mathcal{X} \to \mathcal{Y}$ is G-equivariant when

$$\phi\big(\rho_{\mathcal{X}}(g)\, \boldsymbol{x}\big) = \rho_{\mathcal{Y}}(g)\, \phi(\boldsymbol{x}), \quad \forall g \in G. \tag{5}$$

Let $\boldsymbol{\Pi}$ be the group of all axis permutations of a rank-K tensor. A mapping ϕ is *axis-permutation equivariant* if

$$\phi\big(\Pi(\boldsymbol{x})\big) = \Pi\big(\phi(\boldsymbol{x})\big), \quad \forall \Pi \in \boldsymbol{\Pi}, \tag{6}$$

i.e., permuting the input axes and then applying ϕ produces the same result as applying ϕ first and then permuting the output.

2.3 Cycle Notation for Axes Permutation

To express the reordering of axes throughout this paper, we require a precise notation for permutations. A permutation is formally defined as a bijection (a one-to-one mapping) from a set onto itself. In our context, the set consists of axis indices, such as $\{H, W, D\}$ for $\mathbf{x} \in \mathbb{R}^{H \times W \times D}$.

First, we review the standard *cycle notation* common in algebra. This notation uses parentheses $()$ to group elements into disjoint cycles that show the path each element follows under the permutation. An element within a cycle is mapped to the element immediately following it. The last element in a cycle is mapped back to the first, completing the loop.

For example, consider permutations on the set of three axis indices $\{1, 2, 3\}$:

- A permutation that maps $1 \to 3$, $3 \to 2$, and $2 \to 1$ is written as the single cycle $(1\ 3\ 2)$.
- A permutation that maps $1 \to 3$, $3 \to 1$, and leaves 2 unchanged, $2 \to 2$, is written as $(1\ 3)$. The element 2 is a fixed point.

Adopting this cycle notation, we define the transformation $\mathrm{T}_{i_1 i_2 \ldots i_n}$ as the permutation that maps the original ordered set of axes $d_1 \times d_2 \times \cdots \times d_n$ to the new ordered arrangement $d_{i_1} \times d_{i_2} \times \cdots \times d_{i_n}$. For example,

$$\begin{aligned} \boldsymbol{y} &= \mathrm{T}_{132}(\boldsymbol{x}), \quad \boldsymbol{y} \in \mathbb{R}^{W \times D \times H}, \quad \boldsymbol{x} \in \mathbb{R}^{H \times W \times D}, \\ \boldsymbol{y}' &= \mathrm{T}_{13}(\boldsymbol{x}), \quad \boldsymbol{y}' \in \mathbb{R}^{D \times W \times H}, \quad \boldsymbol{x} \in \mathbb{R}^{H \times W \times D}. \end{aligned} \tag{7}$$

3 Axial Neural Networks

We draw inspiration from permutation-equivariant architectures such as Deep Sets [38] and GNNs, which process set or graph data with a variable number of elements. A key advantage of these models is their ability to handle inputs of varying sizes by sharing the same parameters across all elements. We apply this core idea to the axes of a tensor, proposing a neural network that can process input tensors of varying dimensions using a single set of parameters.

To this end, we introduce a new type of neural network, the Axial Neural Network (XNN), which is equivariant to permutations of a tensor's axes. It achieves this by applying an identical transformation with a shared set of parameters to each axis, thereby treating them as interchangeable elements, similar to the elements of a set or the vertices of a graph. We propose two variants: the set-based XNN and the graph-based XNN.

3.1 Set-Based Axial Neural Networks

Set-based Axial Neural Networks (SXNNs) are inspired by Deep Sets [38]. They treat the input as the set of all possible axis permutations of a given tensor. Specifically,

$$\begin{aligned} &\text{Rank } 1: \ \{\boldsymbol{x}\}, && \text{Rank } 3: \ \{\Pi_0(\boldsymbol{x}), \Pi_1(\boldsymbol{x}), \Pi_2(\boldsymbol{x}), \Pi_3(\boldsymbol{x}), \Pi_4(\boldsymbol{x}), \Pi_5(\boldsymbol{x})\}, \\ &\text{Rank } 2: \ \{\Pi_0(\boldsymbol{x}), \Pi_1(\boldsymbol{x})\}, && \text{Rank } K: \ \{\Pi_0(\boldsymbol{x}), \Pi_1(\boldsymbol{x}), \ldots, \Pi_{K!-1}(\boldsymbol{x})\}, \end{aligned} \tag{8}$$

where $\Pi_0(\boldsymbol{x}) = \boldsymbol{x}$. A Deep-Set style aggregation as in Eq. 2 is then applied:

$$\boldsymbol{y} = \phi\left(\bigoplus_{\Pi \in \boldsymbol{\Pi}} \Pi^{-1}\big(\psi(\Pi(\boldsymbol{x}))\big) \right), \tag{9}$$

with neural networks ϕ, ψ, permutation-invariant aggregation $\bigoplus$ (e.g. sum, mean, or max), and inverse permutation Π^{-1}.

Theorem 3.1. *Let ϕ be an axis-permutation equivariant function (e.g., another SXNN or point-wise operation). The SXNN in Eq. 9 is axis-permutation equivariant for any rank-K tensor $\boldsymbol{x} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_K}$. (§ B.1 for proof)*

Linear. For instance, a simple SXNN may use ϕ as the identity and ψ as a linear layer applied along the last axis, followed by max-pooling over the remaining axes for matching the output size. For a rank-3 tensor $\boldsymbol{x} \in \mathbb{R}^{H \times W \times D}$, it is formalized by

$$\boldsymbol{y} = \sum_{i=0}^{2} \Pi_i^{-1} \left(\mathrm{Pool}_{1,2} \left(\mathrm{Linear}_3 \left(\Pi_i(\boldsymbol{x}) \right) \right) \right), \tag{10}$$

where Linear_3 applies a linear layer along the third axis (D), $\mathrm{Pool}_{1,2}$ pools over the other two axes (H and W), and they treat the remaining axis as batch dimensions. Note that only three out of six permutations are required, since pooling is invariant to permutations across the pooled dimensions, i.e. $\mathrm{Pool}(\mathbb{R}^{H \times W \times D}) = \mathrm{Pool}(\mathbb{R}^{W \times H \times D})$, which omits the redundant permutations. Note also that pooling is equivalent to the approach commonly used in 3D inflation of 2D convolutional layers [6, 28]. The transformation flow of the feature sizes in Eq. 10 is summarized as

$$H \times W \times D \left\{ \begin{array}{l} \xrightarrow{\Pi_0} H \times W \times \boldsymbol{D} \xrightarrow{\mathrm{Linear}_3} \boldsymbol{H} \times \boldsymbol{W} \times d \xrightarrow{\mathrm{Pool}_{1,2}} h \times w \times d \xrightarrow{\Pi_0^{-1}} \\ \xrightarrow{\Pi_1} D \times H \times \boldsymbol{W} \xrightarrow{\mathrm{Linear}_3} \boldsymbol{D} \times \boldsymbol{H} \times d \xrightarrow{\mathrm{Pool}_{1,2}} d \times h \times w \xrightarrow{\Pi_1^{-1}} \\ \xrightarrow{\Pi_2} W \times D \times \boldsymbol{H} \xrightarrow{\mathrm{Linear}_3} \boldsymbol{W} \times \boldsymbol{D} \times h \xrightarrow{\mathrm{Pool}_{1,2}} w \times d \times h \xrightarrow{\Pi_2^{-1}} \end{array} \right\} \xrightarrow{\bigoplus} h \times w \times d,$$

where the bold letters indicate the axes that the operation is applied. This construction generalizes to any rank-K tensor with $(K-1)$-dimensional pooling as

$$(\textbf{Linear}) \quad \boldsymbol{y} = \sum_{i=0}^{K-1} \Pi_i^{-1} \left(\mathrm{Pool}_{1,\ldots,K-1} \left(\mathrm{Linear}_K(\Pi_i(\boldsymbol{x})) \right) \right). \tag{11}$$

Instead of downsampling like pooling, we may use upsampling operations (e.g., resize) to expand the tensor size if Linear_K raises the output size.

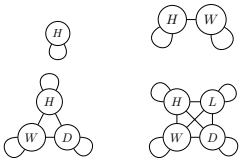
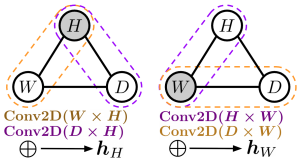
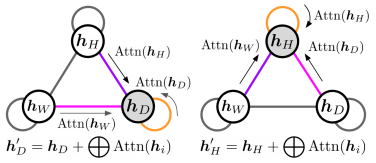


Figure 1: Axis Graphs.



(a) Lifting Layer



(b) Subsequent Layers

Figure 2: Illustration of GXNN in 3D.

Convolution and Attention. For the convolutional layers, we assume the input tensor has an extra channel dimension C, i.e., $\boldsymbol{x} \in \mathbb{R}^{H\times W\times D\times C}$. The axial convolution is still applied along the spatial axes H, W, D. As in the linear case, we can construct the axial convolution using a 1D convolutional layer and the axial self-attention layer applied over one axis. In attention layers, the output sequence length matches the input, so it is unnecessary to apply a pooling or resizing to align output sizes.

$$(\textbf{Conv}) \sum_{i=0}^{K-1} \Pi_i^{-1} \left(\mathrm{Pool}_{1,\dots,K-1} \left(\mathrm{Conv1D}_K(\Pi_i(\boldsymbol{x}))\right)\right), \quad (\textbf{Attn}) \sum_{i=0}^{K-1} \Pi_i^{-1} \left(\mathrm{SelfAttn}_K(\Pi_i(\boldsymbol{x}))\right). \quad (12)$$

Interestingly, the set-based axial attention is already used in a recent PDE foundation model [28] for reducing the computational complexity of the Transformer. Splitting the operation across axes reduces the attention overhead. For instance, self-attention over $\mathbb{R}^{H\times W\times D}$ requires $O((HWD)^2)$ complexity, whereas axial self-attention reduces this to $O(H^2 + W^2 + D^2)$.

Non-linearity. ϕ and ψ in Eq. 9 can be arbitrary neural networks, including those with a single non-linearity such as ReLU or Sigmoid. Therefore, using any type of pointwise operation (including non-linearities) does not violate the axis-permutation equivariance of XNN.

Expressivity. Although SXNNs provide strong expressibility for the dimension-agnostic architecture, their expressivity is inefficient due to their symmetric structure; i.e., they can universally approximate any dimension-agnostic function but require a relatively large width to achieve this. For instance, consider a patch embedding for the Vision Transformer (ViT) [12] and we assume the patch embedding uses a convolution layer with kernel size 2, stride size 2, and both input and output channels are scalar-valued. In a conventional convolution, the operation on a 2×2 pixel patch (2D) can be described as

$$\mathrm{Conv2D}(\boldsymbol{x}): \quad \begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} = ax_1 + bx_2 + cx_3 + dx_4, \quad (13)$$

where $*$ denotes convolution, the first matrix is the kernel, and the second is a 2×2 patch from the input image. On the other hand, the axial convolution in Eq. 12 with average pooling results in:

$$\left\{ \begin{array}{l} \mathrm{Conv1D}(\Pi_0(\boldsymbol{x})): \begin{bmatrix} a & b \\ a & b \end{bmatrix} * \begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_2 \\ ax_3 + bx_4 \end{bmatrix} \xrightarrow{\text{AvgPool}} \frac{ax_1+bx_2+ax_3+bx_4}{2} \\ \mathrm{Conv1D}(\Pi_1(\boldsymbol{x})): \begin{bmatrix} a & b \\ a & b \end{bmatrix} * \begin{bmatrix} x_1 & x_3 \\ x_2 & x_4 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_3 \\ ax_2 + bx_4 \end{bmatrix} \xrightarrow{\text{AvgPool}} \frac{ax_1+ax_2+bx_3+bx_4}{2} \end{array} \right\}$$

$$\xrightarrow{\Sigma} ax_1 + \tfrac{a+b}{2}x_2 + \tfrac{a+b}{2}x_3 + bx_4 = \begin{bmatrix} \boldsymbol{a} & \frac{\boldsymbol{a+b}}{\boldsymbol{2}} \\ \frac{\boldsymbol{a+b}}{\boldsymbol{2}} & \boldsymbol{b} \end{bmatrix} * \begin{bmatrix} \boldsymbol{x_1} & \boldsymbol{x_2} \\ \boldsymbol{x_3} & \boldsymbol{x_4} \end{bmatrix}, \quad (14)$$

which shows that the axial convolution behaves like a convolution with a symmetric kernel. This symmetric kernel structure limits expressivity efficiency, so increasing the number of output channels is often necessary to mitigate this limitation. To address this issue, we also introduce a different type of XNN called the *graph-based XNN*, which avoids the constraint entirely.

3.2 Graph-Based Axial Neural Networks

SXNN produces outputs that are inefficient in terms of expressivity compared to the standard neural networks, due to the simple aggregation $\bigoplus$. To overcome this limitation, we can lift the input into an axes-permutation equivariant space, a strategy widely used in the equivariant neural network literature [8, 9, 36, 13], and aggregate them in the intermediate layers as in GNN.

Lifting Layer. We imagine an undirected graph over the axes such as Fig. 1. We now apply the message-passing logic of GNNs as described in Eq. 1. Since the neighbors of d_1 are $d_2, \dots, d_K$ and

likewise for the other axes, we can informally express the update rule as

$$\boldsymbol{h}_{d_i} = \phi\left(d_i, \bigoplus_{j=1}^{K} \psi(d_i, d_j)\right) \tag{15}$$

where $\oplus$ denotes an arithmetic form of the permutation-invariant operator $\bigoplus$ and $d_1, d_2, \ldots, d_K$ are used as a conceptual feature representing each axis. The functions ϕ and ψ can be chosen based on the architecture or task. Although Eq. 15 follows parameter-sharing principles and permutation equivariance, in practice, we have to modify it to match the axes order after the aggregation $\oplus$.

For example, in the case of convolutional layers, let $\boldsymbol{x} \in \mathbb{R}^{H\times W\times D\times C}$, where C is the number of channels. We can define ϕ as the identity function that returns the second argument (i.e. $\phi(A, B) = B$) and $\psi(H, W)$ as a Conv2D applied over axes H and W with the pooling layer at the end for matching the tensor sizes. We omit $\psi(H, H)$ as it is nontrivial, and the absence of it does not violate the axis permutation equivariance. When the indices of $\{H, W, D\}$ are $\{1, 2, 3\}$, Eq. 15 becomes

$$\begin{aligned}
\boldsymbol{h}_H &= \mathrm{T}_{13}\mathrm{T}_{13}^{-1}\,\mathrm{Pool}_1\,\mathrm{Conv2D}_{2,3}\,\mathrm{T}_{13}(\boldsymbol{x}) + \mathrm{T}_{13}\mathrm{T}_{132}^{-1}\,\mathrm{Pool}_1\,\mathrm{Conv2D}_{2,3}\,\mathrm{T}_{132}(\boldsymbol{x}),\\
\boldsymbol{h}_W &= \mathrm{T}_{23}\mathrm{T}_{23}^{-1}\,\mathrm{Pool}_1\,\mathrm{Conv2D}_{2,3}\,\mathrm{T}_{23}(\boldsymbol{x}) + \mathrm{T}_{23}\mathrm{T}_{123}^{-1}\,\mathrm{Pool}_1\,\mathrm{Conv2D}_{2,3}\,\mathrm{T}_{123}(\boldsymbol{x}),\\
\boldsymbol{h}_D &= \mathrm{T}_{33}\,\mathrm{Pool}_1\,\mathrm{Conv2D}_{2,3}(\boldsymbol{x}) + \mathrm{T}_{33}\mathrm{T}_{12}^{-1}\,\mathrm{Pool}_1\,\mathrm{Conv2D}_{2,3}\,\mathrm{T}_{12}(\boldsymbol{x}),
\end{aligned} \tag{16}$$

where T_{ijk} denotes reordering axes $1, 2, 3$ to i, j, k as explained in § 2.3. Of course, ϕ and ψ need not be linear, and it can be a multilayer perceptron. The transformation flow of the feature sizes of $\boldsymbol{h}_H$ in Eq. 16 would be:

$$H\times W\times D \left\{ \begin{array}{l} \xrightarrow{\mathrm{T}_{13}} D\times \boldsymbol{W}\times \boldsymbol{H} \xrightarrow{\mathrm{Conv2D}_{2,3}} \boldsymbol{D}\times w\times h \xrightarrow{\mathrm{Pool}_1} d\times w\times h \xrightarrow{\mathrm{T}_{13}\mathrm{T}_{13}^{-1}} \\ \xrightarrow{\mathrm{T}_{132}} W\times \boldsymbol{D}\times \boldsymbol{H} \xrightarrow{\mathrm{Conv2D}_{2,3}} \boldsymbol{W}\times d\times h \xrightarrow{\mathrm{Pool}_1} w\times d\times h \xrightarrow{\mathrm{T}_{13}\mathrm{T}_{132}^{-1}} \end{array} \right\} \xrightarrow{+} d\times w\times h = \boldsymbol{h}_H.$$

Here the channel axis C is omitted for simplicity. A CNN example is also illustrated in Fig. 2a.

For 1D and 2D cases, Eq. 15 reduces to generating the corresponding number of outputs.

$$\text{(1D)}\quad \boldsymbol{h}_H = \phi\left(H, \psi(H, H)\right), \qquad \text{(2D)}\quad \begin{aligned} \boldsymbol{h}_H &= \phi\left(H, \psi(H, H) \oplus \psi(H, W)\right),\\ \boldsymbol{h}_W &= \phi\left(W, \psi(W, W) \oplus \psi(W, H)\right). \end{aligned} \tag{17}$$

The **1D case** does not consider interaction with the other nodes and $\psi(H, H)$ is still nontrivial. Thus, instead of determining $\psi(H, H)$, we omit the self-edge term $\psi(H, H)$ or $\psi(W, W)$ but rather augment a 1D tensor to a 2D tensor to fully utilize Conv2D. Possible augmentations include the outer product, repetition, and the diagonal matrix. We adopt repetition, which repeats the 1D tensor along a new axis to match the kernel size of Conv2D, and then averages it after lifting to recover the original 1D tensor.

Generalization of the **Lifting** layers, Eq. 16, to rank-K tensor is described as

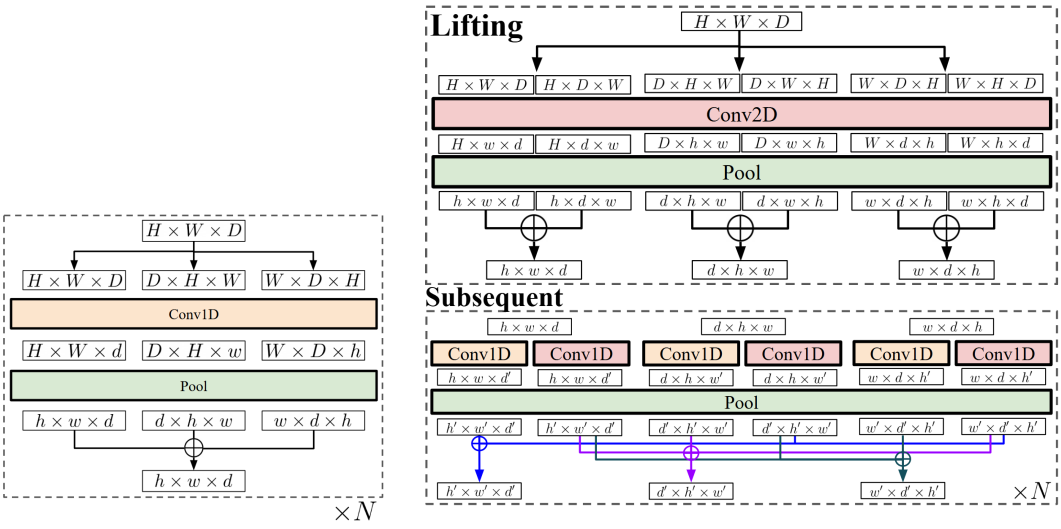
$$\boldsymbol{h}_{d_i} = \phi\left(\mathrm{T}_{(i)(K)}(\boldsymbol{x}), \bigoplus_{j\neq i}^{K} \mathrm{T}_{(i)(K)}\mathrm{T}_{(j)(K-1)(i)(K)}^{-1}\psi\left(\mathrm{T}_{(j)(K-1)(i)(K)}(\boldsymbol{x})\right)\right), \quad K > 1, \tag{18}$$

where $\mathrm{T}_{(a)(b)(c)(d)} = \mathrm{T}_{abcd}$.

Theorem 3.2. *Under some assumptions, the lifting layer of GXNN, Eq. 18, is axis-permutation equivariant for any rank-K (except $K = 1$) tensor $\boldsymbol{x} \in \mathbb{R}^{d_1\times d_2\times\ldots\times d_K}$. (§ B.2 for assumptions and proof)*

Subsequent Layers. Unlike SXNN, which produces a single feature, we obtain three updated features $\boldsymbol{h}_H$, $\boldsymbol{h}_W$, and $\boldsymbol{h}_D$ in GXNN. Due to the lifting construction, these features are equivariant with respect to permutations of the input axes. In other words, the permutation of input axes results in the permutation of output features with rank 3. Therefore, the subsequent layers will be a GNN whose input is a graph with nodes $\boldsymbol{h}_H$, $\boldsymbol{h}_W$, and $\boldsymbol{h}_D$ as the third graph in Fig. 1:

$$\boldsymbol{h}'_{d_i} = \phi\left(\boldsymbol{h}_{d_i}, \bigoplus_{j=1}^{K} \psi(\boldsymbol{h}_{d_i}, \boldsymbol{h}_{d_j})\right). \tag{19}$$



(a) SXCNN (b) GXCNN

Figure 3: An example of SXNN and GXNN for CNN.

Therefore, the subsequent layers should also be dimension-agnostic message-passing architectures, with the lifted features $\boldsymbol{h}_H$, $\boldsymbol{h}_W$, and $\boldsymbol{h}_D$ as graph nodes. Eq. 19 also needs to be modified to match the indices order in the aggregation.

For example, in the case of self-attention layers, we can define ϕ as a residual path, i.e. $\phi(A, B(A)) = A + B(A)$, and ψ as a self-attention applied over the last axis of the second tensor, i.e. $\psi(A, B) = \mathrm{SelfAttn}_K(B)$. Then, as also described in Fig. 2b, the GNN in Eq. 19 becomes

$$\begin{aligned}
\boldsymbol{h}'_H &= \boldsymbol{h}_H + \mathrm{T}_{13}\mathrm{T}_{13}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_H) + \mathrm{T}_{13}\mathrm{T}_{23}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_W) + \mathrm{T}_{13}\mathrm{T}_{33}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_D),\\
\boldsymbol{h}'_W &= \boldsymbol{h}_W + \mathrm{T}_{23}\mathrm{T}_{13}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_H) + \mathrm{T}_{23}\mathrm{T}_{23}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_W) + \mathrm{T}_{23}\mathrm{T}_{33}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_D),\\
\boldsymbol{h}'_D &= \boldsymbol{h}_D + \mathrm{T}_{33}\mathrm{T}_{13}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_H) + \mathrm{T}_{33}\mathrm{T}_{23}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_W) + \mathrm{T}_{33}\mathrm{T}_{33}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_D),
\end{aligned} \tag{20}$$

equivalent to

$$\begin{aligned}
\boldsymbol{m} &= \mathrm{T}_{13}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_H) + \mathrm{T}_{23}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_W) + \mathrm{T}_{33}^{-1}\,\mathrm{SelfAttn}_3(\boldsymbol{h}_D),\\
\boldsymbol{h}'_H &= \boldsymbol{h}_H + \mathrm{T}_{13}(\boldsymbol{m}), \quad \boldsymbol{h}'_W = \boldsymbol{h}_W + \mathrm{T}_{23}(\boldsymbol{m}), \quad \boldsymbol{h}'_D = \boldsymbol{h}_D + \mathrm{T}_{33}(\boldsymbol{m}),
\end{aligned} \tag{21}$$

where Ts are used for aligning the axis order to match the axes in the aggregation. Similarly, the **Subsequent** layers in Eq. 20 for a rank-K tensor can be written as

$$\boldsymbol{h}'_{d_i} = \phi\left(\boldsymbol{h}_{d_i}, \mathrm{T}_{(i)(K)} \bigoplus_{j=1}^{K} \mathrm{T}_{(j)(K)}^{-1} \psi(\boldsymbol{h}_{d_j})\right). \tag{22}$$

Theorem 3.3. *Under some assumptions, the subsequent layers of GXNN in Eq. 22 are axis-permutation equivariant for any rank-K tensor $\boldsymbol{h}_{d_i} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_K}$. (§ B.3 for assumptions and proof)*

Pooling Layer. In typical CNNs such as ResNet [19], before computing the output logit values using the linear head, global average pooling or global max pooling is applied over the height and width of the features to aggregate spatial information. Likewise, in GXNN, we obtain K feature tensors through the lifting layer and subsequent layers, and we need to aggregate these features to merge information across axis permutations. Here is an example and its generalized form:

$$(\textbf{rank } 3) \quad \boldsymbol{h}' = \mathrm{T}_{13}^{-1}(\boldsymbol{h}'_H) \oplus \mathrm{T}_{23}^{-1}(\boldsymbol{h}'_W) \oplus \mathrm{T}_{33}^{-1}(\boldsymbol{h}'_D), \quad (\textbf{rank } K) \quad \boldsymbol{h}' = \bigoplus_{i=1}^{K} \mathrm{T}_{(i)(K)}^{-1} \boldsymbol{h}'_{d_i}. \tag{23}$$

The difference between SXNN and GXNN in a simple CNN architecture can be seen in Fig. 3. SXCNN naturally satisfies the permutation equivariant structure by repeatedly stacking the same layer. On the other hand, GXCNN requires a lifting layer at the beginning of the network and a pooling layer at the end. In the middle, subsequent layers can be stacked repeatedly. Those XCNNs with added nonlinearity and normalization layers are used in § 5.1.

3.3 Example: Dimension-Agnostic PDE Solver

One of the important applications is to solve PDEs. Solving PDEs with AI for reducing the cost of numerical PDE solvers, which often requires supercomputers, is a rapidly rising field in modern machine learning [1, 40, 31, 27]. Each PDE has different spatial dimensionality, and its solutions are represented as a tensor whose elements are points in spatial and temporal grids. Although the neural operator [23] has a crucial benefit in solving PDEs, ViT is still commonly used in PDE foundation models due to its strong generalization. The model for Multiple Physics Pretraining (MPP) [28] is one such model that serves powerful performance in multiple 2D PDE training.

We provide an example of a dimension-agnostic PDE solver by merging GXNN and MPP. MPP consists of patch embedding, multiple attention layers, and patch de-embedding. The patch embedding and patch de-embedding are CNNs. Thus, in the axial implementation, we use the patch embedding as the lifting layer and the rest as the subsequent layers. The attention layers are the same as the SelfAttn example described in Eq. 20, and the patch de-embedding is a convolution variant of it. The details and illustrations of the example can be referred to in § A.

4 Related Work

Several studies have explored dimension-agnostic architectures. Levin and Díaz [25] proposed any-dimensional equivariant neural networks that leverage representation stability from algebraic topology, enabling models trained on fixed input dimensions to generalize to arbitrary sizes. Similarly, Lee et al. [24] introduced dimension-agnostic neural processes, which incorporate a dimension aggregator block to unify inputs of varying dimensions into a shared latent space. These approaches offer strong potential for constructing flexible operators that scale beyond conventional grid-dependent solvers. From a practical standpoint, multi-modal training, particularly joint training on images and videos, has also been widely explored [10, 37, 14, 26, 7]. However, these methods typically treat images as individual video frames and either introduce temporal attention layers or use separate embedders for video, without adopting a truly dimension-agnostic approach.

5 Experiments

5.1 Toy Dataset: Gaussian Process Kernel Prediction

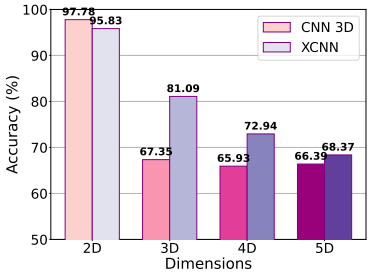


Figure 4: Accuracy on GP kernel prediction.

To demonstrate the dimension-free training and inference capabilities of XNN, we construct a toy dataset of synthetic Gaussian process (GP) data. The task is binary classification between two kernels: radial basis function (RBF) and periodic, given multi-dimensional data (2D, 3D, 4D, and 5D) randomly sampled from the synthetic GP. We build two model to train it: a CNN with 3D convolutions and an XCNN with graph-based axial convolutions composed of 2D convolutions. Although our architecture naturally supports multi-dimensional data, the 3D CNN is restricted to only 3D spatial inputs. Therefore, for 2D data, we pad with zeros to construct 3D-like inputs, and for 4D and 5D data, we flatten the last axes to obtain 3D shapes (e.g., 5D: $16\times16\times16\times16\times16\times C \rightarrow 16\times16\times4096\times C$, where C is the channel dimension). Further experimental details are provided in § D. As shown in Fig. 4, XCNN built with 2D convolutions exhibits superior test accuracy across all dimensions. Notably, despite using 3D convolutions, CNN-3D performs poorly on higher-dimensional data, including 3D, highlighting the robustness of XNN's dimension-agnostic design.

Comparison Between SXNN and GXNN. SXNN exhibits a trade-off between computational efficiency and expressivity inefficiency as described in Eq. 13. To examine this and verify the necessity of GXNN, we build a set-based axial CNN (SXCNN) composed of Conv1Ds and a graph-based axial CNN (GXCNN) from the previous experiment. We then compare their number of parameters (#Params), wall clock time (W. Clock) of forward computation, and inference performance (Acc.) using the GP toy dataset. Table 1 shows the comparisons, where SXCNN-L is an enlarged version of SXCNN for fair comparison with GXCNN. Note that SXCNN with the same depth and width as

Table 1: SXCNN vs. GXCNN.

	SXCNN	SXCNN-L	GXCNN
Depth	4	5	4
Width	128	256	128
#Params	150K	791K	899K
W. Clock	80ms	98ms	101ms
2D Acc.	95.94	92.60	95.45
3D Acc.	63.68	85.72	79.85
4D Acc.	42.54	55.18	70.86
5D Acc.	62.09	62.64	70.48

Table 2: Test NRMSE of PDE solvers on 2D PDEs.

Model	Pretrain	FineTune	DR	NS	SWE	CFD M0.1	CFD M1.0
CViT	×	×	0.0389	0.1078†	0.1876†	-	-
X-CViT	×	×	0.0382	0.1148†	0.1948†	-	-
MPP	×	×	0.0157	-	0.0015	0.0132	0.0181
X-MPP	×	×	0.0118	-	0.0012	0.0118	0.0163
MPP	2D	×	0.0447		0.0087	0.0404*	0.0499*
MPP	1D,2D	×	0.2183	-	0.0265	0.1881*	0.2199*
X-MPP	1D,2D,3D	×	0.0430	-	0.0086	0.0428*	0.0517*
MPP	2D	✓	0.0516	-	0.0022	0.0319*	0.0422*
X-MPP	1D,2D,3D	✓	0.0058	-	0.0011	0.0209*	0.0294*

GXCNN has $6\times$ fewer parameters and 80% reduced wall clock time, but its performance degrades in high dimensions. After increasing the depth and width, it performs fairly well in high dimensions, but GXCNN still exhibits superior performance, indicating the necessity of GXNN.

5.2 PDE Solver Foundation Models

In this experiment, we evaluate the effectiveness of XNN as an architecture for multi-dimensional training. A compelling use case would be multi-PDE solution training, which involves different PDEs with varying dimensionalities. We train PDE solver foundation models on the solutions of time-homogeneous PDEs drawn from a variety of physical systems. The data are sourced from widely used PDE solution benchmark datasets: PDEBench [33] and PDEArena [16]. The list of PDEs and their details are provided in § E.

We implemented our architecture in two state-of-the-art PDE foundation model baselines: CViT [35] and MPP [28]. Both are based on ViT [12], incorporating patch embedding and multiple attention layers with some revisions optimized for PDE learning. Note that these baselines are designed only for 2D data, as the 2D convolutional layer for patch embedding is dimension-dependent, though the attention layers are not. We build their XNN variants, termed X-CViT and X-MPP, which include axial linears, axial convolutions, and axial attentions. The specifications are detailed in § C.

Throughout the experiments, we followed the basic training procedures of baselines such as CViT and MPP, which take a few timesteps as input and predict the next timestep of a PDE solution. We set CViT, X-CViT to take $s = 2$ timesteps as input, and MPP, X-MPP to take $s = 4$, in contrast to the original MPP paper, which used $s = 16$. We also replace InstanceNorm [34] of MPP by LayerNorm [2] due to training instability observed on 1D and 3D data. The evaluation metric is the Normalized Root Mean Squared Error (NRMSE), defined in § D.

The handling of domain-specific features such as boundary conditions, geometry, and time is determined by the baseline methods we modified for XNN; i.e., X-MPP follows the handling method of MPP. Additionally, the baseline methods target PDE solutions defined only on regular grids. According to MPP, the boundary conditions, geometry, and time are not separately input to the model. Instead, MPP is pretrained on multiple PDE solutions with varying boundary conditions and equations, allowing it to learn general patterns of PDE solutions. Since we eventually finetune the model on a specific PDE with known boundary conditions and geometry, it is not necessary to encode them separately as inputs. The only thing MPP handles separately is the periodicity of the boundary condition. Depending on the periodicity of the boundary condition, MPP determines whether to use sequential position bias or periodic position bias in the attention layer.

To isolate the effect of our architectural modifications, we use the smallest versions of CViT and MPP as backbones: CViT-S and MPP-Ti, with 12M and 7M parameters, respectively. Unlike McCabe et al. [28], we exclude incompressible fluid dynamics from training due to its large data size relative to model capacity. The model size is smaller than the pretraining dataset ($\sim$40M), so pretraining offers limited benefit over training from scratch, but it is sufficient to highlight the advantage of multidimensional training.

Expressivity of XNN. We evaluated the architectural expressivity of XNN by measuring how well it performs compared to dimension-specific models in three different settings: single PDE training, multiple PDE training, and single PDE finetuning. Since the baselines are designed for 2D PDEs, we evaluate performance only on 2D PDEs: Diffusion-Reaction (DR), Incompressible Navier-Stokes (NS), Shallow Water Equation (SWE), and Compressible Fluid Dynamics (CFD) with Mach numbers $M = 0.1$ and 1.0. We measure NRMSE for each PDE. Notably, for 1D-2D joint training in MPP, we convert 1D PDE solutions to 2D by padding with zero values.

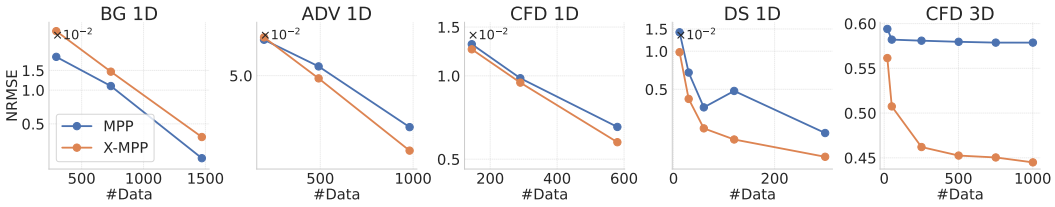


Figure 5: Test NRMSE of finetuning on unseen dimensions (1D and 3D).

The results are shown in Table 2. The notation (†) in the table denotes evaluation on PDEArena, meaning the rest are on PDEBench. (*) indicates results trained with both CFD M0.1 and CFD M1.0, meaning more challenging setup compared to training respectively. The underscores denote the best result compared to the competitors. Since we report based on the baseline codebase, the empty slots denote PDEs that the codebase does not support. In every scenario, the axial variants exhibit competitive results compared to the non-axial baselines. In particular, due to the benefit of multidimensional pretraining, the finetuned X-MPP (trained with 1D, 2D, and 3D data) outperforms MPP trained only with 2D. Note that pretraining MPP with both 1D and 2D leads to significant degradation, as it fails to learn a unified representation space across dimensions.

Unseen Dimension Generalization. In this experiment, we demonstrate the dimension-generalization (few-shot learning) ability of XNN on data from unseen dimensions. To do so, we pretrain both MPP and X-MPP on 2D PDE data and compare their finetuning performance on unseen 1D and 3D PDEs, demonstrating the benefit of multi-dimensional pretraining in XNN-based foundation models. We use three 1D PDEs and one 3D PDE: Diffusion Sorption (DS), Burgers' equation (BG), 1D Compressible Fluid Dynamics (CFD 1D), and 3D Compressible Fluid Dynamics (CFD 3D). We evaluate using the same metric, NRMSE, and compare how finetuning performance improves as the size of the given dataset increases.

For 3D finetuning of MPP, we use the inflation technique described in McCabe et al. [28]. It repeats a $P \times P$ kernel of the 2D convolution layers P times and divides by P. The weights of the linear projection for the additional variable in the 3D PDE are initialized with the average of the trained weights corresponding to the existing variables. For 1D finetuning of MPP, we augment the input tensor from 1D to 2D only in the patch embedding at the beginning and the patch de-embedding at the end, so that the attention layers in the middle operate on purely 1D PDEs. In contrast, our model naturally extends to 3D and reduces to 1D. For X-MPP, we use a shared linear projection across different PDEs for unified representation learning. This contrasts with MPP, where a different linear projection learns each PDE during fine-tuning.

As shown in Fig. 5, except for BG 1D, X-MPP (orange) accelerates the finetuning performance with only a small amount of data. In particular for 3D, the redundancy of the inflated layers in MPP (blue) limits 3D generalization, whereas the dimension-agnostic architecture is free from this issue, thereby efficiently utilizing the representation trained from 2D PDEs.

6 Conclusion

In this work, we introduce XNNs, a family of efficient and expressive architectures designed to be agnostic to input dimensionality. Motivated by the limitations of prior PDE models constrained to fixed dimensions, XNNs leverage axis permutation equivariance over tensor axes to naturally generalize across 1D, 2D, and 3D domains, which can be applied to develop a dimension-agnostic PDE solver foundation model.

In the PDE solving problem, our empirical results highlight the benefits of multidimensional pretraining and the superior finetuning ability of XNNs compared to the dimension-specific models such as MPP. These findings underscore the importance of designing foundation models capable of operating across varying spatial dimensions, which is a critical step toward scalable and adaptable scientific machine learning systems.

Limitations and Future Work. An axial version of the cross attention between two tensors of different dimensions is not yet fully explored. Addressing this issue as a natural extension of this work is a valuable future direction that would further widen the usability of XNNs. Additionally, improving the computational efficiency of GXNN, especially for high-dimensional data, is also a promising direction towards the practical deployment of XNNs in large-scale scientific simulations.

Acknowledgement

We thank *Heejun Lee* for his thoughtful technical support, *Jules Berman* for his insightful concerns, and *Kyunghyun Cho* for facilitating our wonderful collaborative research. This work was partly supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2019-II190075, Artificial Intelligence Graduate School Program(KAIST); No.RS-2024-00509279, Global AI Frontier Lab; No.RS-2022-II220713, Meta-learning Applicable to Real-world Problems) and Artificial intelligence industrial convergence cluster development project funded by the Ministry of Science and ICT(MSIT, Korea) & Gwangju Metropolitan City. H. Yang was supported by the National Research Foundation of Korea(NRF) grant funded by the Korean Government(MSIT) (No. RS-2023-00279680).

References

[1] Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[2] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

[3] Benjamin Bloem-Reddy and Yee Whye Teh. Probabilistic symmetries and invariant neural networks. *J. Mach. Learn. Res.*, 21:90:1–90:61, 2020. URL https://jmlr.org/papers/v21/19-322.html.

[4] Blake Bordelon, Alexander B. Atanasov, and Cengiz Pehlevan. A dynamical model of neural scaling laws. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

[6] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4724–4733. IEEE Computer Society, 2017.

[7] Shoufa Chen, Mengmeng Xu, Jiawei Ren, Yuren Cong, Sen He, Yanping Xie, Animesh Sinha, Ping Luo, Tao Xiang, and Juan-Manuel Pérez-Rúa. Gentron: Diffusion transformers for image and video generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 6441–6451. IEEE, 2024. doi: 10.1109/CVPR52733.2024.00616. URL https://doi.org/10.1109/CVPR52733.2024.00616.

[8] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *Proceedings of The 33rd International Conference on Machine Learning (ICML 2016)*, 2016.

[9] Taco S. Cohen and Max Welling. Steerable cnns. In *International Conference on Learning Representations (ICLR)*, 2017.

[10] Yatin Dandi, Aniket Das, Soumye Singhal, Vinay P. Namboodiri, and Piyush Rai. Jointly trained image and video generation using residual vectors. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2020, Snowmass Village, CO, USA, March 1-5, 2020*, pages 3017–3031. IEEE, 2020. doi: 10.1109/WACV45572.2020.9093308. URL https://doi.org/10.1109/WACV45572.2020.9093308.

[11] de Finetti and B. Funzione caratteristica di un fenomeno aleatorio. *Attidella R. Academia Nazionale dei Lincei, Serie*, 6.(4):251299., 1931.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[13] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *Proceedings of The 37th International Conference on Machine Learning (ICML 2020)*, 2020.

[14] Rohit Girdhar, Mannat Singh, Nikhila Ravi, Laurens van der Maaten, Armand Joulin, and Ishan Misra. Omnivore: A single model for many visual modalities. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 16081–16091. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01563. URL https://doi.org/10.1109/CVPR52688.2022.01563.

[15] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks, IJCNN 2005, Montreal, QC, Canada, July 31 - August 4, 2005*, pages 729–734. IEEE, 2005.

[16] Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

[17] Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. DPOT: auto-regressive denoising operator transformer for large-scale PDE pre-training. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[18] Alexander Havrilla and Wenjing Liao. Understanding scaling laws with statistical and approximation theory for transformer neural networks on intrinsically low-dimensional data. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[20] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2024. URL http://github.com/google/flax.

[21] Maximilian Herde, Bogdan Raonic, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[22] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

[23] Nikola B. Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *J. Mach. Learn. Res.*, 24:89:1–89:97, 2023. URL https://jmlr.org/papers/v24/21-1524.html.

[24] Hyungi Lee, Chaeyun Jang, Dongbok Lee, and Juho Lee. Dimension agnostic neural processes, 2025. URL https://arxiv.org/abs/2502.20661.

[25] Eitan Levin and Mateo Díaz. Any-dimensional equivariant neural networks. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li, editors, *International Conference on Artificial Intelligence and Statistics, 2-4 May 2024, Palau de Congressos, Valencia, Spain*, volume

238 of *Proceedings of Machine Learning Research*, pages 2773–2781. PMLR, 2024. URL https://proceedings.mlr.press/v238/levin24a.html.

[26] Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Limin Wang, and Yu Qiao. Uniformerv2: Unlocking the potential of image vits for video understanding. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 1632–1643. IEEE, 2023. doi: 10.1109/ICCV51070.2023.00157. URL https://doi.org/10.1109/ICCV51070.2023.00157.

[27] Yuxuan Liu, Jingmin Sun, Xinjie He, Griffin Pinney, Zecheng Zhang, and Hayden Schaeffer. Prose-fd: A multimodal pde foundation model for learning multiple operators for forecasting fluid dynamics, 2024. URL https://arxiv.org/abs/2409.09811.

[28] Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles D. Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Géraud Krawezik, François Lanusse, Mariel Pettee, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. Multiple physics pretraining for spatiotemporal surrogate models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

[30] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605. URL https://doi.org/10.1109/TNN.2008.2005605.

[31] Zezheng Song, Jiaxin Yuan, and Haizhao Yang. Fmint: Bridging human designed and data pretrained models for differential equation foundation model, 2024. URL https://arxiv.org/abs/2404.14688.

[32] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Networks*, 8(3):714–735, 1997. doi: 10.1109/72.572108. URL https://doi.org/10.1109/72.572108.

[33] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[34] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. URL http://arxiv.org/abs/1607.08022.

[35] Sifan Wang, Jacob H Seidman, Shyam Sankaran, Hanwen Wang, and George J Pappas Paris. Cvit: Continuous vision transformer for op-erator learning. *arXiv preprint arXiv:2405.13998*, 3, 2024.

[36] Maurice Weiler and Gabriele Cesa. General e(2)-equivariant steerable cnns. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019.

[37] Haiyang Xu, Qinghao Ye, Ming Yan, Yaya Shi, Jiabo Ye, Yuanhong Xu, Chenliang Li, Bin Bi, Qi Qian, Wei Wang, Guohai Xu, Ji Zhang, Songfang Huang, Fei Huang, and Jingren Zhou. mplug-2: A modularized multi-modal foundation model across text, image and video. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38728–38748. PMLR, 2023.

[38] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017.

[39] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, 2010. doi: 10.1109/CVPR.2010.5539957.

[40] Hang Zhou, Yuezhou Ma, Haixu Wu, Haowen Wang, and Mingsheng Long. Unisolver: Pde-conditional transformers are universal pde solvers, 2024. URL https://arxiv.org/abs/2405.17527.

[41] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. doi: 10.1016/J.AIOPEN.2021.01.001. URL https://doi.org/10.1016/j.aiopen.2021.01.001.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction reflects our main contribution: novel architecture for dimension-free training and inference.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discuss the limitations in the last paragraph of the conclusion.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The paper provides the assumptions and proofs in the appendix.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides experimental details in the experiment section and the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We share the source code with configuration files in the submission and the data is publicly available.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The paper provides the experiemental details including hyperparameters in the appendix section.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Error bars are not reported because the pretraining is too computationally expensive.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper provide such information in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The paper only targets mathematical data, which does not involve any harmful aspect.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: The paper only targets mathematical data, which does not involve any societal impacts.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper trains with only PDEs.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper clarifies the source of baseline codes and data.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper will release the code with the documentation after accept.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: No crowdsourcing and human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No crowdsourcing and human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

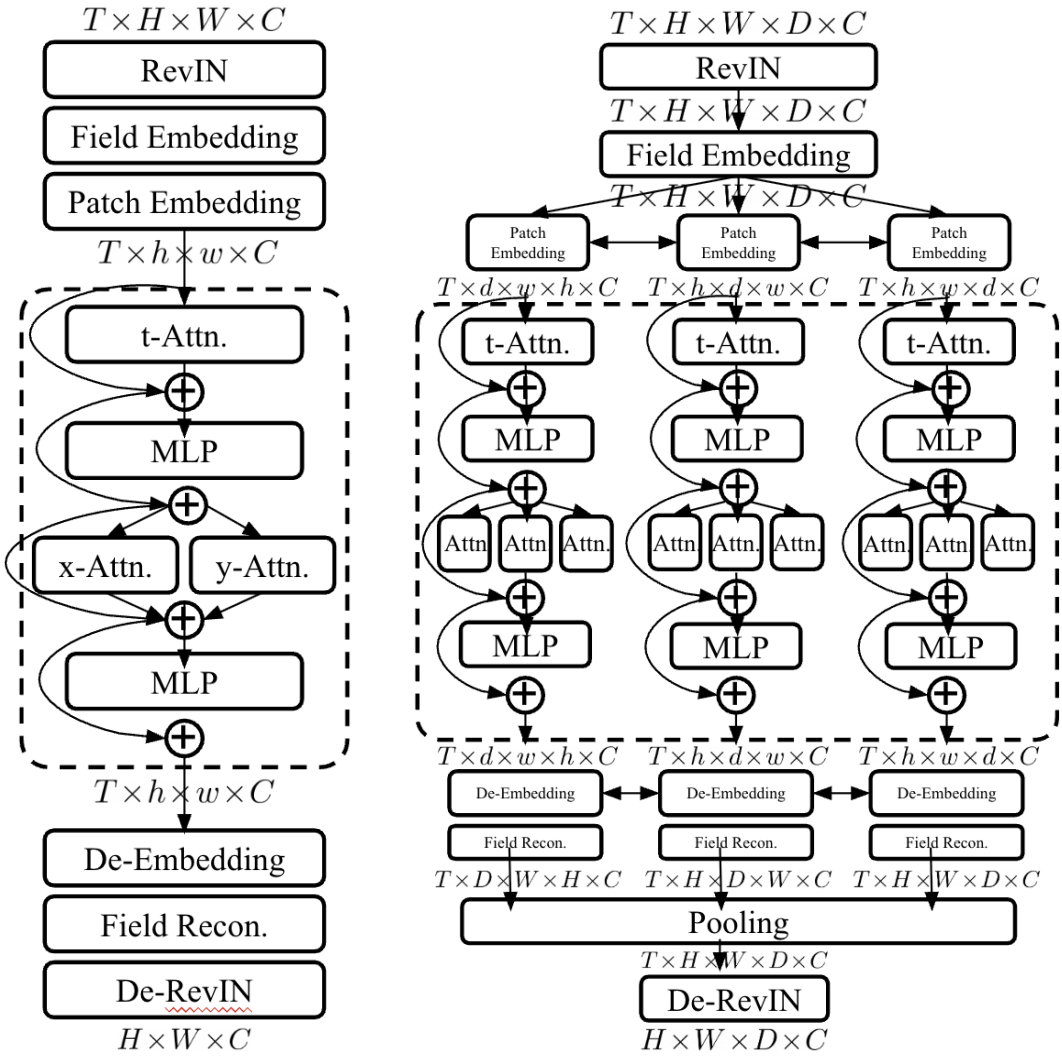
    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: LLM is used only for editing.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.



(a) MPP (b) Dimension-Agnostic MPP

Figure 6: Comparison between MPP and its dimension-agnostic version.

A Dimension-Agnostic PDE Solver

In this section, we present an example of the dimension-agnostic PDE solver using the ViT-based model MPP as a backbone. MPP processes a PDE solution from timestep 1 to T (e.g., $\boldsymbol{x}_{1:T} \in \mathbb{R}^{T \times H \times W \times C}$) and outputs the next timestep $T+1$ (e.g., $\boldsymbol{x}_{T+1} \in \mathbb{R}^{H \times W \times C}$). MPP consists of patch embedding, multiple self-attention layers, and patch de-embedding. In the self-attention layers, MPP separates the attention along the time axis (temporal self-attention) in a depthwise manner. We also follow this separate temporal self-attention, but make the spatial self-attention dimension-agnostic. The comparison between MPP and its dimension-agnostic version is illustrated in Fig. 6.

Patch Embedding. The patch embedding is a three-layer CNN, used as the lifting layer. The original patch embedding in MPP is as follows:

$$\begin{aligned} &\text{EmbBlock}(\boldsymbol{x}) = \text{GeLU}(\text{Norm}(\text{Conv2D}(\boldsymbol{x}))), \\ &\text{Embedder}(\boldsymbol{x}) = \text{Conv2D}(\text{EmbBlock}(\text{EmbBlock}(\boldsymbol{x}))), \\ &\boldsymbol{h} = \text{Norm}(\text{Embedder}(\boldsymbol{x})), \end{aligned} \tag{24}$$

where Norm indicates InstanceNorm [34]. On the other hand, in the axial version, we use $\mathrm{Embedder}$ for ψ in Eq. 18 with $\max$ aggregation as follows:

$$\begin{aligned}\tilde{\boldsymbol{h}}_{d_i} &= \max_{j\neq i,\, j\in[1,K]} \mathrm{T}_{(K-1)(j)} \left(\mathrm{AvgPool}_{1,\ldots,K-2} \left(\mathrm{Embedder}_{K-1,K} \left(\mathrm{T}_{(j)(K-1)(i)(K)}(\boldsymbol{x})\right)\right)\right), \\ \boldsymbol{h}_{d_i} &= \mathrm{Norm}(\tilde{\boldsymbol{h}}_{d_i}),\end{aligned} \tag{25}$$

where $\mathrm{T}_{(m)(n)}(\boldsymbol{x})$ denotes the transpose between the m-th and n-th axes. Through $\mathrm{T}_{(j)(K-1)}\mathrm{T}_{(i)(K)}$, the j-th and i-th axes are located at the $(K-1)$-th and K-th positions, respectively. Then, $\mathrm{Embedder}_{K-1,K}$ applies a 2D CNN along the $K-1$ and K axes, and the remaining axes $1,\ldots,K-2$ are reduced via average pooling to match the output tensor size. Since the final Norm is a pointwise operation, it preserves axis-permutation equivariance. We replace InstanceNorm with LayerNorm [2] to avoid training instability in multidimensional training.

Self-Attentions. MPP already employs set-based axial self-attention, which outputs a single feature as follows:

$$\tilde{\boldsymbol{h}}' = \frac{1}{K}\sum_{i=1}^{K} \mathrm{T}_{(K)(i)}(\mathrm{SelfAttn}_K(\mathrm{T}_{(i)(K)}(\boldsymbol{h}))), \quad \boldsymbol{h}' = \boldsymbol{h} + \tilde{\boldsymbol{h}}'. \tag{26}$$

In contrast, in the graph-based axial case, we obtain K features due to the lifting layer in the patch embedding. Therefore, we also use a set-based structure but apply the same attention to each feature independently, which differs from the graph-based attention example in Eq. 21. The self-attention used is:

$$\tilde{\boldsymbol{h}}'_{d_i} = \frac{1}{K}\sum_{i=1}^{K} \mathrm{T}_{(K)(i)}(\mathrm{SelfAttn}_K(\mathrm{T}_{(i)(K)}(\boldsymbol{h}_{d_i}))), \quad \boldsymbol{h}'_{d_i} = \boldsymbol{h}_{d_i} + \tilde{\boldsymbol{h}}'_{d_i}, \quad \forall i \in [1,K]. \tag{27}$$

Additional operations such as layer normalization and drop residual paths are included, but omitted here for simplicity. These operations follow MPP exactly.

Patch De-Embedding. In this block, we recover the input tensor's width and height using a CNN consisting of $\mathrm{ConvTranspose}$ layers (also known as deconvolutions) [39]. In the original implementation, this CNN is defined as:

$$\begin{aligned}\mathrm{DeembBlock}(\boldsymbol{h}') &= \mathrm{GeLU}(\mathrm{Norm}(\mathrm{ConvT2D}(\boldsymbol{h}'))), \\ \mathrm{Deembedder}(\boldsymbol{h}') &= \mathrm{ConvT2D}(\mathrm{DeembBlock}(\mathrm{DeembBlock}(\boldsymbol{h}'))), \\ \boldsymbol{h}'' &= \mathrm{Deembedder}(\boldsymbol{h}').\end{aligned} \tag{28}$$

In the axial version, we define two $\mathrm{Deembedder}$ blocks: one for updating the node feature and the other for updating the neighborhood feature (nhbr). We aggregate them using a $\max$ operation. Then, Eq. 22 becomes:

$$\begin{aligned}\boldsymbol{h}''^{(\mathrm{node})}_{d_i} &= \mathrm{Deembedder}^{(\mathrm{node})}_{K-1,K}(\boldsymbol{h}'_{d_i}), \\ \boldsymbol{h}''^{(\mathrm{nhbr})}_{d_i} &= \max_{j\neq i,\, j\in[1,K]} \mathrm{T}_{(i)(K)}\mathrm{T}_{(K)(j)}\, \mathrm{Deembedder}^{(\mathrm{nhbr})}_{K-1,K}(\boldsymbol{h}'_{d_j}), \\ \boldsymbol{h}''_{d_i} &= \max\left\{\boldsymbol{h}''^{(\mathrm{node})}_{d_i}, \boldsymbol{h}''^{(\mathrm{nhbr})}_{d_i}\right\}, \quad \forall i \in [1,K],\end{aligned} \tag{29}$$

where $\mathrm{T}_{(i)(K)}\mathrm{T}_{(K)(j)}$ aligns the axis order of $\boldsymbol{h}'_{d_j}$ to match that of $\boldsymbol{h}'_{d_i}$.

Unfortunately, this dimension-agnostic PDE solver is not axis-permutation equivariant because the *positional encoding* (not mentioned in this section but present in the actual implementation) and the patch de-embedding layer break axis-permutation equivariance, even though the solver anyway works for every dimensionality.

B Proofs of Theorems

B.1 Theorem 3.1

Let ϕ be an axis-permutation equivariant function (e.g., another SXNN or pointwise operation). The SXNN in Eq. 9 is axis-permutation equivariant for any rank-K tensor $\boldsymbol{x} \in \mathbb{R}^{d_1\times d_2\times\ldots\times d_K}$.

Proof. We prove the axis-permutation equivariance property defined in Eq. 6. For any axis permutation $\Pi' \in \mathbf{\Pi}$, the permutation of the input in Eq. 9 before applying ϕ becomes

$$\bigoplus_{i=1}^{K!-1} \Pi_i^{-1}\big(\psi(\Pi_i(\Pi'(\boldsymbol{x})))\big). \tag{30}$$

Now, by substituting $\Pi_j = \Pi_i\Pi'$ (equivalently, $\Pi_i = \Pi_j{\Pi'}^{-1}$), we get

$$\begin{aligned}\bigoplus_{i=1}^{K!-1} \Pi_i^{-1}\big(\psi(\Pi_i(\Pi'(\boldsymbol{x})))\big) &= \bigoplus_{j=1}^{K!-1} \Pi'\Pi_j^{-1}\big(\psi(\Pi_j(\boldsymbol{x}))\big) \\ &= \Pi' \bigoplus_{j=1}^{K!-1} \Pi_j^{-1}\big(\psi(\Pi_j(\boldsymbol{x}))\big) \\ &= \Pi' \bigoplus_{i=1}^{K!-1} \Pi_i^{-1}\big(\psi(\Pi_i(\boldsymbol{x}))\big).\end{aligned} \tag{31}$$

The last equality holds due to the permutation-invariant nature of the operation $\bigoplus$.

By the assumption that ϕ is axis-permutation equivariant, it follows that

$$\begin{aligned}\phi\Bigg(\bigoplus_{i=1}^{K!-1} \Pi_i^{-1}\big(\psi(\Pi_i(\Pi'(\boldsymbol{x})))\big)\Bigg) &= \phi\Bigg(\Pi' \bigoplus_{i=1}^{K!-1} \Pi_i^{-1}\big(\psi(\Pi_i(\boldsymbol{x}))\big)\Bigg) \\ &= \Pi'\phi\Bigg(\bigoplus_{i=1}^{K!-1} \Pi_i^{-1}\big(\psi(\Pi_i(\boldsymbol{x}))\big)\Bigg),\end{aligned} \tag{32}$$

which proves that Eq. 9 is axis-permutation equivariant for all $\Pi' \in \mathbf{\Pi}$. □

B.2 Theorem 3.2

Assumption 1. All layers follow the same axis order. For example, if the lifting generates $\boldsymbol{h}_H$ and $\boldsymbol{h}_W$ with shapes $H \times W \times C$ and $W \times H \times C$, respectively, then every layer should produce $\boldsymbol{h}'_H$ and $\boldsymbol{h}'_W$ with the same shapes.

Assumption 2. In the lifting layer, ϕ must be axis-permutation equivariant along axes $1, 2, \ldots, K-1$, and ψ along axes $1, 2, \ldots, K-2$. For instance, if $\boldsymbol{x} \in \mathbb{R}^{H\times W\times D\times C}$ for $K = 3$, then $\mathrm{T}_{12}(\phi(\boldsymbol{x})) = \phi(\mathrm{T}_{12}(\boldsymbol{x}))$ must hold, but $\mathrm{T}_{13}(\phi(\boldsymbol{x})) = \phi(\mathrm{T}_{13}(\boldsymbol{x}))$ is not required.

Under assumptions Assumption 1 and Assumption 2, the lifting layer of GXNN, Eq. 18, is axis-permutation equivariant for any rank-K (except $K = 1$) tensor $\boldsymbol{x} \in \mathbb{R}^{d_1\times d_2\times\ldots\times d_K}$.

Proof. The lifting layer Lifting described in Eq. 18 can be equivalently written as

$$\boldsymbol{h}_{d_i} = \phi\Bigg(\mathrm{T}_{(i)(K)}(\boldsymbol{x}), \bigoplus_{j\neq i}^{K} \mathrm{T}_{(i)(K)}\mathrm{T}^{-1}_{(i)(K)(j)(K-1)}\psi\left(\mathrm{T}_{(i)(K)(j)(K-1)}(\boldsymbol{x})\right)\Bigg) \tag{33}$$

where $\mathrm{T}_{(a)(b)(c)(d)} = \mathrm{T}_{abcd}$ denotes the axes permutation from indices $\{1, 2, 3, 4\}$ to $\{a, b, c, d\}$ as explained in § 2.3. Applying an axis permutation $\Pi(\boldsymbol{x})$ to the input gives

$$\phi\Bigg(\mathrm{T}_{(i)(K)}(\Pi(\boldsymbol{x})), \bigoplus_{j\neq i}^{K} \mathrm{T}_{(i)(K)}\mathrm{T}^{-1}_{(i)(K)(j)(K-1)}\psi\left(\mathrm{T}_{(i)(K)(j)(K-1)}(\Pi(\boldsymbol{x}))\right)\Bigg) \tag{34}$$

Substituting $\Pi'\mathrm{T}_{(k)(K)} = \mathrm{T}_{(i)(K)}\Pi$ for $i = \pi(k)$ and $\Pi''\mathrm{T}_{(i)(K)(j)(K-1)} = \mathrm{T}_{(k)(K)(l)(K-1)}\Pi$ for $i = \pi(k), j = \pi'(l)$, where Π' permutes only axes 1 through $K-1$ and Π'' permutes only axes 1 through $K-2$, we get

$$= \phi\Bigg(\Pi'\mathrm{T}_{(k)(K)}(\boldsymbol{x}), \bigoplus_{\pi'(l)\neq\pi(k)}^{K} \Pi'\mathrm{T}_{(k)(K)}\mathrm{T}^{-1}_{(k)(K)(l)(K-1)}{\Pi''}^{-1}\psi\left(\Pi''\mathrm{T}_{(k)(K)(l)(K-1)}(\boldsymbol{x})\right)\Bigg). \tag{35}$$

Let π, π' be the element-wise permutation across axes induced by Π. Since the aggregation is permutation-invariant and $\pi'(l) \neq \pi(k)$ is equivalent to $l \neq k$ (as π' does not act on the K-th axis and k-th axis is transposed to K by $\mathrm{T}_{(k)(K)}$), and using Assumption 2 (permutation equivariance of ψ under Π''), we have

$$= \phi\left(\Pi'\mathrm{T}_{(k)(K)}(\boldsymbol{x}), \bigoplus_{l\neq k}^{K} \Pi'\mathrm{T}_{(k)(K)}\mathrm{T}^{-1}_{(k)(K)(l)(K-1)}\Pi''^{-1}\Pi''\psi\left(\mathrm{T}_{(k)(K)(l)(K-1)}(\boldsymbol{x})\right)\right). \tag{36}$$

Applying Assumption 2 again for ϕ under Π' yields

$$= \Pi'\phi\left(\mathrm{T}_{(k)(K)}(\boldsymbol{x}), \bigoplus_{l\neq k}^{K} \mathrm{T}_{(k)(K)}\mathrm{T}^{-1}_{(k)(K)(l)(K-1)}\psi\left(\mathrm{T}_{(k)(K)(l)(K-1)}(\boldsymbol{x})\right)\right), \tag{37}$$

which follows the definition of axes-permutation equivariance as in Eq. 6. □

B.3 Theorem 3.3

Assumption 3. In the subsequent layers, both ϕ and ψ must be axis-permutation equivariant along axes $1, 2, \ldots, K-1$.

Under assumptions Assumption 1 and Assumption 3, the subsequent layers of GXNN in Eq. 22 are axis-permutation equivariant for any rank-K tensor $\boldsymbol{h}_{d_i} \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_K}$.

Proof. The subsequent layers Subsequent described in Eq. 22 can be equivalently written as

$$\boldsymbol{h}'_{d_i} = \phi\left(\boldsymbol{h}_{d_i}, \mathrm{T}_{(i)(K)} \sum_{j=1}^{K} \mathrm{T}^{-1}_{(j)(K)}\psi(\boldsymbol{h}_{d_j})\right). \tag{38}$$

Using Eq. 37, permutation of input leads to permutation of output of the lifting layer, which is the permutation of input of subsequent layers,

$$\phi\left(\Pi'\boldsymbol{h}_{d_k}, \mathrm{T}_{(\pi'(k))(K)} \sum_{\pi'(l)=1}^{K} \mathrm{T}^{-1}_{(\pi'(l))(K)}\psi(\Pi'\boldsymbol{h}_{d_l})\right). \tag{39}$$

Because the index exchanges in T must adjust to the input permutation Π', and both ϕ and ψ are Π'-equivariant by Assumption 3, and the sum is permutation-invariant:

$$= \Pi'\phi\left(\boldsymbol{h}_{d_k}, \mathrm{T}_{(k)(K)} \sum_{l=1}^{K} \mathrm{T}^{-1}_{(l)(K)}\psi(\boldsymbol{h}_{d_l})\right) = \Pi'\boldsymbol{h}'_{d_k}. \tag{40}$$

B.4 Axes-Permutation Equivariance of Pooling Layer

Finally, the pooling layer aggregates the axis-wise features into a single feature, which remains equivariant to axis permutations Π. As described in the example in Eq. 23, it is equivalently written as

$$\boldsymbol{h}'' = \sum_{j=1}^{K} \mathrm{T}^{-1}_{(j)(K)}\boldsymbol{h}'_{d_j}. \tag{41}$$

By definition,

$$\Pi'\mathrm{T}_{(k)(K)} = \mathrm{T}_{(\pi(k))(K)}\Pi \quad \Leftrightarrow \quad \Pi'\mathrm{T}_{(K)(k)} = \mathrm{T}_{(K)(\pi(k))}\Pi. \tag{42}$$

Using this identity and Eq. 40, permutation of the input yields

$$\begin{aligned}\sum_{\pi'(l)=1}^{K} \mathrm{T}^{-1}_{(\pi'(l))(K)}\Pi'\boldsymbol{h}'_{d_l} &= \sum_{\pi'(l)=1}^{K} \Pi'\mathrm{T}^{-1}_{(l)(K)}\boldsymbol{h}'_{d_l} = \sum_{\pi'(l)=1}^{K} \mathrm{T}^{-1}_{(\pi(l))(K)}\Pi\boldsymbol{h}'_{d_l} \\ &= \sum_{\pi'(l)=1}^{K} \Pi\mathrm{T}^{-1}_{(l)(K)}\boldsymbol{h}'_{d_l} = \Pi\sum_{l=1}^{K} \mathrm{T}^{-1}_{(l)(K)}\boldsymbol{h}'_{d_l} = \Pi\boldsymbol{h}'',\end{aligned} \tag{43}$$

which concludes that the axis-permutation of the input yields the axis-permutation of the output. □

C X-CViT and X-MPP

For X-MPP, we exactly followed the example of the dimension-agnostic PDE solver introduced in § A.

For X-CViT, we mostly followed the structure of CViT but modified the dimension-dependent components, including patch embedding, spatial self-attention, coordinate query embedding, and decoder cross-attention. For the remaining components, we refer the reader to [35].

Patch Embedding. X-CViT also uses patch embedding as a lifting layer. In the original CViT, the patch embedding is defined as

$$\boldsymbol{h} = \mathrm{Conv2D}(\boldsymbol{x}). \tag{44}$$

In contrast, in X-CViT, the patch embedding is defined as

$$\boldsymbol{h}_{d_i} = \frac{1}{K}\sum_{j\neq i}^{K} \left(\mathrm{T}_{(K-1)(j)(K)(i)} \left(\mathrm{MaxPool}_{1,\dots,K-2} \left(\mathrm{Conv2D}_{K-1,K} \left(\mathrm{T}_{(i)(K)(j)(K-1)}(\boldsymbol{x})\right)\right)\right)\right), \tag{45}$$

where $\mathrm{T}_{(i)(K)(j)(K-1)}$ is used to apply $\mathrm{Conv2D}_{K-1,K}$ along the i-th and j-th axes, and $\mathrm{T}_{(K-1)(j)(K)(i)}$ reorders the axes back to enable aligned feature aggregation through the summation.

Spatial Self-Attentions. CViT separates temporal and spatial attention and X-CViT also follows the separated structure. In X-CViT, we use set-based axial self-attention for each feature. The original self-attention in CViT is defined as

$$\boldsymbol{h}' = \mathrm{SelfAttn}(\mathrm{Flatten}(\boldsymbol{x})), \tag{46}$$

which flattens the spatial axes of the input into a single sequence before applying self-attention (e.g., $H \times W \to HW$). In contrast, X-CViT does not perform flattening but instead aggregates self-attention outputs along each axis:

$$\boldsymbol{h}'_{d_i} = \boldsymbol{h}_{d_i} + \sum_{j=1}^{K} \mathrm{T}_{(K)(j)} \left(\mathrm{SelfAttn}(\mathrm{T}_{(j)(K)}(\boldsymbol{h}_{d_i}))\right), \tag{47}$$

where $\mathrm{T}_{(j)(K)}$ is used to apply $\mathrm{SelfAttn}$ along the j-th axis, and $\mathrm{T}_{(K)(j)}$ reorders the axes back for aggregation.

Pooling Axial Features. Right after computing the encoder, we still have K axial features lifted by the patch embedding. Thus, before applying the decoder, we aggregate them by averaging to obtain a single feature.

Coordinate Query Embedding. While MPP always predicts the entire spatial grid points at the next timestep, CViT selects where to predict by providing spatial coordinates, assuming the PDE solutions lie on a $[0, 1]^2$ grid. To achieve this, CViT calculates the distance between N query coordinates and all grid points in G^K, resulting in a tensor of shape $N \times G^K$, which is then encoded into a $N \times Q$ tensor, where Q is the feature size of the coordinate embedding. CViT uses two linear layers to embed G^K into $\mathbb{R}^Q$ for the distance tensor $\boldsymbol{q} \in \mathbb{R}^{N\times G^K}$ (with $K = 2$ in 2D), treating N as a batch dimension:

$$\boldsymbol{q}' = \mathrm{Norm}(\mathrm{Linear}(\mathrm{Linear}(\boldsymbol{q}))). \tag{48}$$

On the other hand, X-CViT utilizes a set-based axial linear layer with $\max$ aggregation as follows:

$$\boldsymbol{q}' = \max_{i\in[1:K]} \mathrm{T}_{(K)(i)} \left(\mathrm{Norm}_K \left(\mathrm{Linear}_K \left(\mathrm{GlobalMaxPool}_{1,\dots,K-1} \left(\mathrm{Linear}_K(\mathrm{T}_{(i)(K)}(\mathrm{Unflatten}(\boldsymbol{q})))\right)\right)\right)\right), \tag{49}$$

where $\mathrm{Unflatten}$ reshapes $G^K \to G \times G \times \cdots \times G$, and $\mathrm{T}_{(i)(K)}$ is used to apply Linear_K along the i-th axis. Note that $\mathrm{GlobalMaxPool}_{1,\dots,K-1}$ reduces the size of axes 1 through $K-1$ to one, converting $\mathbb{R}^{Q\times\cdots\times Q}$ to $\mathbb{R}^Q$. This illustrates that XNN can be applied even when the number of input axes differs from the number of output axes.

Decoder Cross-Attention. In the decoder cross-attention, we perform attention by treating the coordinate embedding $\boldsymbol{q}$ as the query and the features produced by the CViT encoder as the key and

value. Since this operation involves a rank-1 vector and a rank-K tensor, we must carefully design the XNN architecture to handle such multi-dimensional cross-attention. In CViT, the cross-attention is formulated as

$$\boldsymbol{h}'' = \boldsymbol{q}' + \mathrm{Attn}(\boldsymbol{q}', \mathrm{Flatten}(\boldsymbol{h}'), \mathrm{Flatten}(\boldsymbol{h}')). \tag{50}$$

In contrast, X-CViT uses SXNN with a Repeat function to align dimensions and tensor sizes:

$$\boldsymbol{h}'' = \boldsymbol{q}' + \sum_{i=1}^{K} \mathrm{GlobalMaxPool}_{1,\ldots,K-1}\left(\mathrm{Attn}(\mathrm{Repeat}_{1,\ldots,K-1}(\boldsymbol{q}'), \mathrm{T}_{(i)(K)}(\boldsymbol{h}'), \mathrm{T}_{(i)(K)}(\boldsymbol{h}'))\right), \tag{51}$$

where $\mathrm{Repeat}_{1,\ldots,K-1}(\boldsymbol{q}')$ expands and repeats the vector $\boldsymbol{q}' \in \mathbb{R}^Q$ along axes $1, \ldots, K-1$, resulting in $\mathrm{Repeat}_{1,\ldots,K-1}(\boldsymbol{q}') \in \mathbb{R}^{Q \times Q \times \cdots \times Q}$.

D Experimental Details

D.1 GP Kernel Prediction

Synthetic Dataset Construction. To construct the synthetic dataset, samples were generated from zero-mean Gaussian processes using either the radial basis function kernel or the periodic kernel. For each kernel, data were generated in dimensions ranging from 2 to 5. In each dimension $d \in \{2, 3, 4, 5\}$, a structured grid that equally divides the hypercube $[-2, 2]^d$ was created over the input space, with grid sizes tailored to 2^{7-d}. This results in output shapes as specified in Table 3, where the last channel of length 1 was appended for convenience.

Table 3: Shape of tensors in the synthetic dataset

Dimensions	Shape	Dimensions	Shape
2D	(32, 32, 1)	4D	(8, 8, 8, 8, 1)
3D	(16, 16, 16, 1)	5D	(4, 4, 4, 4, 4, 1)

Each grid point represents an input to the GP kernel, from which the full covariance matrix was computed for a batch of samples. Kernel parameters were randomly sampled, i.e., length within $[0.1, 0.6]$, scale within $[0.1, 1]$, and period within $[0.1, 0.5]$ for the periodic kernel, to introduce variation. The resulting covariance matrices were corrected via eigenvalue clipping to ensure positive semidefiniteness. Each generated sample from the periodic kernel was labeled 0, while those from the RBF kernel were labeled 1. An equal number of samples from each kernel type was used to ensure balanced classes. All data were normalized to have zero mean and unit variance before being fed into the models. The dataset was then divided into training and validation subsets with an 80/20 split.

3D CNN. As 3D convolution operations are only compatible with 5D inputs, we forced the dimensions of the tensor to 5; for lower-dimensional samples, we zero-padded along missing axes; for higher-dimensional samples, we reshaped the tensor to be 5D by aggregating all the dimensions after the third dimension. We adopted a conventional 3D convolutional neural network operating on tensors of shape (B, H, W, D, C), where B is the batch size. The network consists of three stacked 3D convolutional layers, each with 32 filters and kernel sizes of $3 \times 3 \times 3$, followed by LayerNorm and ReLU activations. After the convolutional layers, global max pooling is applied across all spatial dimensions, and a final dense layer maps the resulting features to a single output for binary classification.

SXCNN. Set-based XCNN generalizes convolution to multidimensional inputs using directional convolutions along each axis. At its core is the SXConv module, which applies 1D convolutions separately along each axis and merges the resulting features using an element-wise maximum. Each SXConv operation is followed by LayerNorm and a ReLU activation. The network consists of five SXConv layers, with each layer operating at 64 hidden features, effectively doubling the base hidden dimensionality. The final feature maps are globally pooled across all spatial dimensions, and classification is performed using a fully connected output layer.

GXCNN. Graph-based XCNN introduces a more complex interaction between spatial axes through the use of cross-axial convolutions. The architecture begins with a lifting layer, which performs

2D convolutions across each pair of axes to produce a set of intermediate representations. This is followed by a series of XConv layers, where each layer simultaneously considers pairs of axes using separate *node* and *neighbor* convolutions. The feature maps from each interaction are merged using max operations after appropriate axis permutation. Each convolution operation is followed by LayerNorm and ReLU activation. The network consists of one XLift layer and four XConv layers, each with 32 hidden features. Axial max pooling was applied across each dimension, followed by global max pooling, and the pooled output was passed to a dense layer for binary classification.

Hyperparameters. All models were trained using the Adam optimizer, with a fixed learning rate of 0.001 and a batch size of 64. Training was conducted for 10 epochs for each model. The loss function used was binary cross-entropy computed from the sigmoid of the output logits. Training and evaluation were implemented in JAX 0.4.30 [5] and Flax 0.8.5 [20], with PyTorch 2.7.0+cu118 [29] used primarily for data preprocessing and batching. To ensure that performance comparisons were valid, the same hyperparameters and preprocessing procedures were applied across all models. The table below summarizes the hyperparameter settings used for training each of the three neural network models.

Table 4: Hyperparameters for each model architecture

Model	Architecture	Hidden Dim	Learning Rate	Batch Size	Epochs
CNN	3 Conv layers	32	1e-3	64	10
SXCNN	5 SXConv layers	64	1e-3	64	10
XCNN	1 XLift + 4 XConv layers	32	1e-3	64	10

D.2 PDE Foundation Model

Hardware and Software. We implemented CViT and X-CViT using JAX 0.4.30 [5] and FLAX 0.8.5 [20], while MPP and X-MPP were implemented using PyTorch 2.1.0+cu121 [29]. All experiments were conducted on NVIDIA GPUs: RTX 3090, RTX A6000, and RTX 5090. For the RTX 5090 machine, we used PyTorch 2.8.0+cu128 due to CUDA driver compatibility.

Training Loss. In CViT training, the objective function is the l_2 loss between the predicted next-timestep solution and the ground truth:

$$\mathcal{L}_{\text{CViT}} = \frac{1}{|B|} \sum_{\boldsymbol{x} \in \mathcal{X}} \left\| \hat{\boldsymbol{x}}_{t+1} - f(\boldsymbol{x}_{(t-s):t}) \right\|_2^2, \tag{52}$$

where B is the mini-batch, $\hat{\boldsymbol{x}}_{t+1}$ is the ground-truth solution at the next timestep, f is the neural PDE solver being trained, t is the timestep, and s is the number of input timesteps.

However, different PDEs exhibit varying magnitudes in their state variables, which can lead to imbalanced training that overemphasizes PDEs with larger magnitudes. To address this, McCabe et al. [28] used the normalized mean squared error (NMSE), which scales each output to unit magnitude. We adopted the same loss function for X-MPP training:

$$\mathcal{L}_{\text{MPP}} = \frac{1}{|B|} \sum_{\boldsymbol{x} \in \mathcal{X}} \frac{\left\| \hat{\boldsymbol{x}}_{t+1} - f(\boldsymbol{x}_{(t-s):t}) \right\|_2^2}{\|\hat{\boldsymbol{x}}_{t+1}\|_2^2 + \epsilon}. \tag{53}$$

Since each PDE may produce tensors of different sizes, composing multiple PDEs in a single mini-batch is not feasible. Instead, we accumulate gradients across multiple mini-batches by summing them before performing a parameter update.

Evaluation Metric. For evaluation, we use the normalized root mean squared error (NRMSE), which is the square root of the normalized mean squared error (NMSE), to compare with baseline methods. The metric is defined as

$$\frac{1}{|B|} \sum_{\boldsymbol{x} \in \mathcal{X}} \frac{\left\| \hat{\boldsymbol{x}}_{t+1} - f(\boldsymbol{x}_{(t-s):t}) \right\|_2}{\|\hat{\boldsymbol{x}}_{t+1}\|_2}. \tag{54}$$

Table 5: Hyperparameters in the CViT training.

	DR		NS		SWE	
Hyperparam.	CViT	X-CViT	CViT	X-CViT	CViT	X-CViT
Patch Size	(8,8)	(8,8)	(8,8)	(8,8)	(8,8)	(8,8)
Grid Size	(128,128)	(128,128)	(128,128)	(128,128)	(96,192)	(96,192)
Latent Dim, Embed Dim, Depth, Attn Heads	512,384,5,6	512,384,6,6	512,384,5,6	512,384,5,6	512,384,5,6	512,384,5,6
Decoder Embed Dim, Decoder Attn Heads, Decoder Depth	512,16,1	512,16,1	512,16,1	512,16,1	512,16,1	512,16,1
Out Dim	2	2	3	3	2	2
Input Timesteps	2	2	10	10	2	2
Train/Val/Test Splits	900/0/50	900/0/50	6500/0/10	6500/0/10	5600/0/10	5600/0/10
Minibatch	16	32	16	16	32	32
Optim	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
LR Schedule	Warm. Exp. Decay	Warm. Exp. Decay	Warm. Exp. Decay	Warm. Exp. Decay	Warm. Exp. Decay	Warm. Exp. Decay
LR init, end, peak	0, 1E-6, 5E-4	0, 1E-6, 1E-3	0, 1E-6, 1E-3	0, 1E-6, 1E-3	0, 1E-6, 1E-3	0, 1E-6, 1E-3
LR decay, transit, warmup	0.9, 5000, 5000	0.9, 5000, 5000	0.9, 5000, 5000	0.9, 5000, 5000	0.9, 5000, 5000	0.9, 5000, 5000
Weight Decay	1E-5	1E-5	1E-5	1E-5	1E-5	1E-5
Grad. Clip	1.0	1.0	1.0	1.0	1.0	1.0
Iterations	3E+5	3E+5	2E+5	2E+5	2E+5	2E+5

Table 6: Hyperparameters in the MPP from-scratch training and pretraining.

	From-Scratch						Pretraining			
	MPP			X-MPP			MPP		X-MPP	
Hyperparam.	DR	SWE	CFD	DR	SWE	CFD	2D	1,2D	2D	1,2,3D
Batch Size	64	128	8	64	64	8	4	4	16	16
Input Timesteps	16	16	16	16	16	16	4	4	4	4
Accumulation Steps	5	5	5	5	5	5	5	5	6	5
Epochs	120	120	120	120	120	120	120	120	120	110

Table 7: Hyperparameters in the MPP finetuning.

	MPP								X-MPP							
Hyperparam.	DR2D	SWE2D	CFD2D	DS1D	CFD1D	BG1D	ADV1D	CFD3D	DR2D	SWE2D	CFD2D	DS1D	CFD1D	BG1D	ADV1D	CFD3D
Batch Size	16	16	16	16	16	16	16	16	64	16	64	16	16	16	16	16
Input Timesteps	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Accumulation Steps	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Epochs	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120	120

Hyperparameters. We report the hyperparameters used for training CViT, X-CViT, MPP, and X-MPP. The hyperparameters include the number of training epochs, train/val/test split, minibatch size, gradient accumulation steps, optimizer, weight decay, drop path probability, learning rate, learning rate scheduling, gradient clipping, and others.

For CViT training, the hyperparameters are listed in Table 5. For MPP training, most hyperparameters are shared across different settings, though some differ. The base values of the hyperparameters are:

- **Epochs:** 120
- **Train/Val/Test Splits:** X%/10%/10% split on each dataset at the trajectory level, where X denotes a subsample from 80% of the total dataset
- **Minibatch Size:** 16
- **Accumulation Steps:** 5
- **Optimizer:** Adan
- **Weight Decay:** 1E-3
- **Drop Path Probability:** 0.1
- **Learning Rate:** DAdaptation
- **Learning Rate Scheduling:** Cosine Decay
- **Gradient Norm Clipping:** 1.0

These choices mostly follow the settings of McCabe et al. [28]. The varying hyperparameters for from-scratch training, pretraining, and finetuning are described in Table 6 and Table 7.

E Partial Differential Equations

The PDE solution benchmarks are employed from PDEBench [33] and PDEArena [16]. Here are the specifications of the equations and their boundary conditions.

E.1 2D Shallow Water Equations.

The shallow water equations, derived from the general Navier–Stokes equations, provide a reduced-order model for free-surface flows such as waves and dam breaks. In the PDEBench benchmark, they are used to simulate a 2D radial dam break scenario. The governing equations are given by:

$$\begin{aligned} \partial_t h + \partial_x(hu) + \partial_y(hv) &= 0, \\ \partial_t(hu) + \partial_x\left(u^2 h + \frac{1}{2} g_r h^2\right) + \partial_y(uvh) &= -g_r h \partial_x b, \\ \partial_t(hv) + \partial_y\left(v^2 h + \frac{1}{2} g_r h^2\right) + \partial_x(uvh) &= -g_r h \partial_y b, \end{aligned} \tag{55}$$

where h denotes the water depth, (u, v) are the horizontal velocity components, g_r is the gravitational acceleration, and $b(x, y)$ represents the bathymetry.

The initial condition corresponds to a circular bump in the center of the domain $\Omega = [-2.5, 2.5]^2$:

$$h(t = 0, x, y) = \begin{cases} 2.0, & \text{if } \sqrt{x^2 + y^2} < r, \\ 1.0, & \text{otherwise}, \end{cases} \tag{56}$$

where the radius r is randomly sampled from the uniform distribution $U(0.3, 0.7)$.

The simulation is performed using the PyClaw finite volume solver. This PDE setup introduces realistic dynamics including shock propagation and wave reflections.

E.2 2D Compressible Fluid Dynamics.

The 2D compressible Navier–Stokes equations describe the dynamics of a compressible fluid, accounting for variations in mass, momentum, and energy over time. These equations are fundamental for modeling gas flows where density changes are significant. The system consists of the conservation laws for mass, momentum, and energy:

$$\begin{aligned} &\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \\ &\rho(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \eta \Delta \mathbf{v} + \left(\zeta + \frac{\eta}{3}\right) \nabla(\nabla \cdot \mathbf{v}), \\ &\partial_t \left(\epsilon + \frac{1}{2}\rho |\mathbf{v}|^2\right) + \nabla \cdot \left[\left(\epsilon + p + \frac{1}{2}\rho |\mathbf{v}|^2\right) \mathbf{v} - \mathbf{v} \cdot \boldsymbol{\sigma}'\right] = 0, \end{aligned} \tag{57}$$

where ρ is the fluid density, $\mathbf{v}$ is the velocity vector, p is the pressure, $\epsilon = p/(\Gamma - 1)$ is the internal energy (with $\Gamma = 5/3$), and $\boldsymbol{\sigma}'$ is the viscous stress tensor. The parameters η and ζ denote the shear and bulk viscosities, respectively.

To generate the data, the simulations employ a second-order accurate HLLC finite volume solver for the inviscid terms, coupled with central differencing for the viscous contributions. This setup enables the benchmark to test model fidelity across a wide range of physically realistic fluid dynamics problems.

E.3 2D Diffusion-Reaction Equation.

This equation models the interaction between two spatially distributed quantities (commonly an *activator* and an *inhibitor*) across a two-dimensional domain. It captures complex spatiotemporal behaviors such as wave propagation and pattern formation, often seen in biological or chemical systems.

The system is described by:

$$\begin{aligned} \partial_t u &= D_u \left(\partial_{xx} u + \partial_{yy} u\right) + R_u(u, v), \\ \partial_t v &= D_v \left(\partial_{xx} v + \partial_{yy} v\right) + R_v(u, v), \end{aligned} \tag{58}$$

where $u(t, x, y)$ and $v(t, x, y)$ denote the concentrations of the activator and inhibitor, respectively. D_u and D_v are their diffusion coefficients. The reaction dynamics are governed by the FitzHugh–Nagumo model:

$$R_u(u, v) = u - u^3 - k - v, \quad R_v(u, v) = u - v, \tag{59}$$

with $k = 5 \times 10^{-3}$.

In the benchmark setup, the simulation domain is $x, y \in (-1, 1)$ and $t \in (0, 5]$. The initial conditions are generated using Gaussian noise, and Neumann boundary conditions (zero flux) are applied to ensure no flow across domain boundaries. Numerical solutions are computed using the finite volume method with fourth-order Runge–Kutta time integration.

E.4 2D Incompressible Navier-Stokes Equations (PDEArena).

The Navier-Stokes equations are a cornerstone of fluid dynamics, describing the motion of fluid substances under the influence of internal and external forces. In PDEArena, the two-dimensional incompressible Navier-Stokes equations are employed to investigate complex multi-scale flow phenomena. These equations govern the evolution of the velocity field $\mathbf{v}(t, \mathbf{x}) \in \mathbb{R}^2$ in a domain $\mathbf{x} \in \mathbb{R}^2$, and are formulated in the velocity-pressure form as:

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla p + \mu \nabla^2 \mathbf{v} + \mathbf{f}, \quad \nabla \cdot \mathbf{v} = 0, \tag{60}$$

where p is the pressure, μ is the kinematic viscosity (diffusion coefficient), and $\mathbf{f}$ represents external force, such as buoyancy.

For simulation in PDEArena, an additional scalar field is introduced, representing a passive scalar (e.g., particle concentration) that is advected by the velocity field and interacts with it through an external buoyancy force $\mathbf{f} = (0, f)^\top$.

The initial conditions include both the velocity and scalar fields, defined over a 128×128 grid with a resolution of $\Delta x = \Delta y = 0.25$. The simulation time-step is $\Delta t = 1.5$ seconds, and the domain is closed with Dirichlet boundary conditions $\mathbf{v} = 0$ and Neumann conditions $\partial s / \partial x = 0$ for the scalar field.

The simulations are numerically solved using the Φ`Flow` framework and serve as a rich testbed for evaluating neural PDE surrogates in capturing advection-diffusion dynamics, vortex interactions, and response to varying force parameters.

E.5 Shallow Water Equations (PDEArena).

The shallow water equations are a set of hyperbolic partial differential equations that describe the flow of a thin layer of incompressible fluid under the influence of gravity. They are derived from the incompressible Navier–Stokes equations by assuming that the horizontal length scales are much larger than the vertical ones, leading to a vertically averaged flow model. In PDEArena, the shallow water equations are used to model both local and global geophysical flow phenomena, such as waves and large-scale atmospheric dynamics.

The equations govern the evolution of the fluid height $h(t, \mathbf{x})$ and the horizontal velocity field $\mathbf{v}(t, \mathbf{x}) = (u, v)$ over a 2D domain $\mathbf{x} \in \mathbb{R}^2$, and take the following form:

$$\begin{aligned} \frac{\partial h}{\partial t} + \nabla \cdot (h\mathbf{v}) &= 0, \\ \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} + g\nabla h &= \mu \nabla^2 \mathbf{v} + \mathbf{f}, \end{aligned} \tag{61}$$

where g is the gravitational acceleration, μ is the viscosity, and $\mathbf{f}$ represents external forces such as wind stress or Coriolis effects.

The simulations are initialized with spatial fields of velocity and pressure, and are performed on a global grid with resolution 192×96, corresponding to a spatial discretization of $\Delta x = 1.875°$, $\Delta y = 3.75°$, and a temporal resolution of $\Delta t = 48$ hours. Periodic boundary conditions are applied in the longitudinal direction, while appropriate boundary conditions (e.g., reflective or free-slip) are used in the latitudinal direction.

These simulations are generated using a modified version of the `SpeedyWeather.jl` framework. The setup allows for evaluating the performance of neural PDE surrogates on both velocity-pressure and vorticity-stream function formulations, capturing a wide range of scales and flow features relevant to climate and weather modeling.

E.6 1D Diffusion-Sorption Equation.

The diffusion-sorption equation models a diffusion process in porous media that is retarded by a non-linear sorption mechanism. This type of process is relevant in real-world applications such as groundwater contaminant transport. In the PDEBench benchmark, it is used to simulate the 1D transport of a solute under the influence of sorption based on the Freundlich isotherm. The governing equation is given by:

$$\partial_t u(t,x) = \frac{D}{R(u)} \partial_{xx} u(t,x), \tag{62}$$

where $u(t,x)$ denotes the concentration of the solute, $D = 5 \times 10^{-4}$ is the effective diffusion coefficient, and $R(u)$ is the retardation factor that accounts for the sorption effect. The retardation factor is defined as:

$$R(u) = 1 + \frac{1-\phi}{\phi} \rho_s k_{\mathrm{nf}} u^{n_f - 1}, \tag{63}$$

with $\phi = 0.29$ the porosity, $\rho_s = 2880$ the bulk density, $k_{\mathrm{nf}} = 3.5 \times 10^{-4}$ the Freundlich coefficient, and $n_f = 0.874$ the Freundlich exponent.

The initial condition is sampled from a uniform distribution:

$$u(t=0,x) \sim \mathcal{U}(0, 0.2), \quad \text{for } x \in (0,1). \tag{64}$$

The boundary conditions are given by:

$$u(t,0) = 1.0, \quad u(t,1) = D \partial_x u(t,1), \tag{65}$$

where the second condition is a Cauchy-type boundary involving a spatial derivative, introducing numerical challenges.

The simulation is performed using a finite volume method for spatial discretization and a fourth-order Runge–Kutta method for time integration. This PDE setup captures realistic nonlinear diffusion behaviors with singularities and complex boundary dynamics.

E.7 1D Burgers' Equation.

The Burgers' equation is a fundamental nonlinear partial differential equation that models the interplay between convection and diffusion in fluid dynamics. It serves as a simplified prototype for the Navier–Stokes equations and is used to study shock formation and dissipative processes. In the PDEBench benchmark, the 1D viscous Burgers' equation is used to simulate such nonlinear dynamics. The governing equation is given by:

$$\partial_t u(t,x) + \partial_x \left(\frac{1}{2} u^2(t,x) \right) = \frac{\nu}{\pi} \partial_{xx} u(t,x), \tag{66}$$

where $u(t,x)$ is the velocity field and ν is the diffusion coefficient, representing the kinematic viscosity.

The initial condition is constructed as a superposition of sinusoidal modes:

$$u(t=0,x) = \sum_{i=1}^{N} A_i \sin(k_i x + \phi_i), \tag{67}$$

where the wave numbers $k_i = 2\pi n_i / L_x$ are randomly selected integers $n_i \in [1, n_{\text{max}}]$, $A_i \sim \mathcal{U}(0, 1)$ are amplitudes, $\phi_i \sim \mathcal{U}(0, 2\pi)$ are phases, and $L_x = 1$ is the domain length. Additional operations such as applying the absolute value or a window function are applied with small probability to introduce further variability.

The domain is defined as $x \in (0, 1)$, and periodic boundary conditions are imposed:

$$u(t, 0) = u(t, 1), \quad \partial_x u(t, 0) = \partial_x u(t, 1). \tag{68}$$

The simulation is performed using a second-order upwind finite difference scheme for the convective term and a central difference scheme for the diffusive term. This PDE setup is particularly suited for studying shock dynamics, nonlinear wave interactions, and the effect of viscosity on solution smoothness.

E.8 1D Advection Equation.

The advection equation is a linear hyperbolic partial differential equation that models the transport of a conserved quantity without diffusion or reaction. It serves as a canonical example for studying wave propagation and translation phenomena in physics and engineering. In the PDEBench benchmark, the 1D advection equation is used to simulate pure transport dynamics. The governing equation is given by:

$$\partial_t u(t, x) + \beta \, \partial_x u(t, x) = 0, \tag{69}$$

where $u(t, x)$ represents the advected scalar quantity and β is the constant advection speed.

The initial condition is defined as a superposition of sinusoidal waves:

$$u(t = 0, x) = \sum_{i=1}^{N} A_i \sin(k_i x + \phi_i), \tag{70}$$

with $k_i = 2\pi n_i / L_x$ representing the wave numbers for randomly selected integers $n_i \in [1, n_{\text{max}}]$, amplitudes $A_i \sim \mathcal{U}(0, 1)$, phases $\phi_i \sim \mathcal{U}(0, 2\pi)$, and $L_x = 1$ denoting the domain length. With small probability, transformations such as taking the absolute value or applying a window function are applied to the initial field to increase diversity.

The spatial domain is $x \in (0, 1)$, and periodic boundary conditions are employed:

$$u(t, 0) = u(t, 1), \quad \partial_x u(t, 0) = \partial_x u(t, 1). \tag{71}$$

The numerical solution is obtained using a second-order upwind finite difference scheme in both space and time. This PDE setup serves as a benchmark for evaluating models' ability to learn and reproduce translational dynamics with minimal distortion or dispersion.

E.9 1D Compressible Fluid Dynamics.

The 1D compressible fluid dynamics (CFD) equations model the conservation of mass, momentum, and energy in a compressible fluid. Derived from the general compressible Navier–Stokes equations, they are used to simulate phenomena such as shock waves, rarefaction, and contact discontinuities. In the PDEBench benchmark, this setup includes various configurations such as inviscid flow, viscous flow, and shock-tube initial conditions. The governing equations are given by:

$$\begin{aligned} \partial_t \rho + \partial_x (\rho u) &= 0, \\ \partial_t (\rho u) + \partial_x \left(\rho u^2 + p \right) &= \partial_x \sigma, \\ \partial_t E + \partial_x \left[(E + p) u \right] &= \partial_x (u \sigma), \end{aligned} \tag{72}$$

where ρ is the density, u is the velocity, p is the pressure, $E = \epsilon + \frac{1}{2} \rho u^2$ is the total energy with internal energy $\epsilon = \frac{p}{\Gamma - 1}$, and σ is the viscous stress term defined as $\sigma = \eta \partial_x u$ for shear viscosity η. The ratio of specific heats is set to $\Gamma = 5/3$.

The benchmark includes multiple initial condition types:

- **Random field:** Initial ρ, u, and p fields generated as smooth random perturbations using a superposition of sine waves.

- **Shock tube:** A Riemann problem where left and right states (ρ, u, p) are sampled from uniform distributions with a sharp discontinuity at a random position.

The spatial domain is $x \in (0, 1)$ with two boundary condition types:

- **Periodic:** Fields wrap around the domain,
- **Out-going:** Ghost cells copy the nearest interior value to allow waves to exit.

The simulations are performed using a second-order HLLC Riemann solver with MUSCL reconstruction for inviscid cases, and central differencing for viscous terms. This PDE setup is challenging due to strong nonlinearity, shock formation, and sensitivity to initial and boundary conditions.

E.10 3D Compressible Fluid Dynamics.

The 3D compressible Navier–Stokes equations govern the motion of gases where density, pressure, and velocity fields evolve in space and time. This system models conservation of mass, momentum, and energy in three dimensions, making it essential for simulating realistic high-speed flows, turbulence, and shock dynamics. The equations are same as Eq. 57, but their variables are three dimensional, e.g. $\mathbf{v} \in \mathbb{R}^3$.

Numerical solutions are generated using a second-order HLLC scheme for the inviscid part and central differencing for viscosity. This 3D setting significantly increases the complexity of the benchmark, introducing challenges in both computational cost and physical realism for surrogate models.