Scalable Transformer for PDE Surrogate Modeling

Zijie Li, Dule Shu, Amir Barati Farimani
Carnegie Mellon University
Mechanical Engineering Department
{zijieli, dules}@andrew.cmu.edu & barati@cmu.edu

Abstract

Transformer has shown state-of-the-art performance on various applications and has recently emerged as a promising tool for surrogate modeling of partial differential equations (PDEs). Despite the introduction of linear-complexity attention, applying Transformer to problems with a large number of grid points can be numerically unstable and computationally expensive. In this work, we propose Factorized Transformer (FactFormer), which is based on an axial factorized kernel integral. Concretely, we introduce a learnable projection operator that decomposes the input function into multiple sub-functions with one-dimensional domain. These sub-functions are then evaluated and used to compute the instance-based kernel with an axial factorized scheme. We showcase that the proposed model is able to simulate 2D Kolmogorov flow on a 256×256 grid and 3D smoke buoyancy on a $64 \times 64 \times 64$ grid with good accuracy and efficiency. The proposed factorized scheme can serve as a computationally efficient low-rank surrogate for the full attention scheme when dealing with multi-dimensional problems.

1 Introduction

Various physics processes are modeled by partial differential equations (PDEs), from the interaction between atoms in molecular systems to large-scale cosmological phenomena. Solving PDEs advances the understanding of complex physical phenomena, enabling people to make accurate predictions, and make informed decisions across a wide range of scientific and engineering disciplines. Numerical solvers provide a practical way to simulate and predict PDEs since many PDEs are often difficult to solve analytically. Most numerical solvers divide the continuous domain into a discretized grid and reduce the continuous differential equations to algebraic equations via methods like finite difference/element/volume methods or spectral method. Despite the theoretical guarantees behind them, their practical realization of specific problems can pose challenges that require careful expertise to overcome, such as a sufficient understanding of the underlying physics, or a fine-tailored mesh that resolves the necessary spatio-temporal scales. The interest in developing user-friendly and efficient PDE solvers, along with the success of deep learning models in many other areas [13, 44, 51, 103], has facilitated the emergence of neural-network-based PDE solvers, where the neural network can be used to parameterize the solution function of the target equation [96], or to approximate the solution operator[69, 76]. Compared to many numerical solvers, neural PDE solvers appear to be more tolerant with coarse discretization [104], and can be applied without explicit meshing [96]. In addition, knowing the underlying equations are not strictly necessary for neural PDE solvers, which gives them the potential to simplify and accelerate the process of physics simulation based on PDEs.

Among various neural network designs, attention-based models (Transformer) [115] have become state-of-the-art for a wide array of applications [13, 18, 27, 51], which gives rise to a recent surge of interest in applying the Transformer to PDE modeling [16, 29, 35, 41, 43, 54, 67, 73, 82, 86]. By viewing the input sequence as a function sampled on a discretization grid, attention can be interpreted as a learnable kernel integral [17, 35, 54, 60] or a learnable Galerkin projection [16], and the sequence-to-sequence Transformer [115] can be modified correspondingly to be better suited for

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

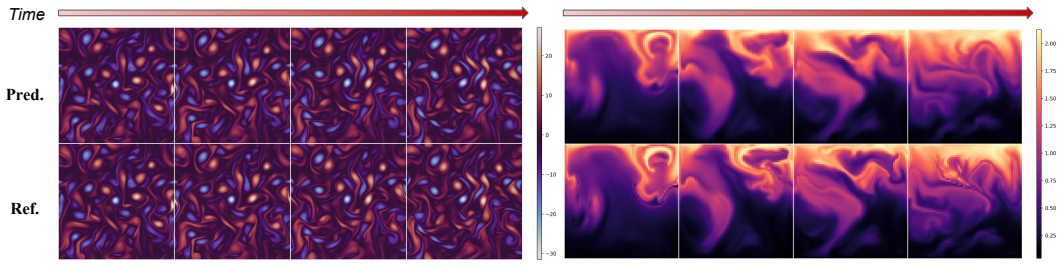


Figure 1: Model's prediction (pred.) and reference ground truth (ref.). **Left**: 2D Kolmogorov flow on 256×256 grid; **Right**: 3D smoke buoyancy on $64 \times 64 \times 64$ grid (zOy cross-section is shown).

PDE modeling [16, 35, 43, 54, 67, 74, 86]. In these works, attention is typically applied to every grid point in the domain to exploit both the local and non-local structure of the system, and therefore a linear-complexity variant of attention is usually necessary. As the number of grid points grows exponentially with respect to the number of dimensions, this results in a very large attention matrix that computes the interaction between every pair of the grid points (despite this attention matrix is not evaluated explicitly in linear attention). Consequently, cascading a deep stack of attention layers introduces instability and relatively high computational cost on high-resolution grid. To alleviate these issues and improve the scalability of Transformer in PDE modeling, we propose a modified attention mechanism. Our model is inspired by the kernel integral viewpoint of softmax-free attention, with a factorized integration scheme motivated by the inherent low-rank structure of dot-product kernel matrix. More specifically, we propose a multi-dimensional factorized kernel integral with each kernel function in the integral having only single-dimensional domains. To calculate these axial kernels, we propose a learnable integral operator that is able to project the input function with high-dimensional domain into a set of sub-functions with single-dimensional domain. The computation of each axial kernel is quadratic with respect to the number of grid points along that the corresponding axis but does not grow with the number of dimensions, which alleviates the curse of dimensionality in standard attention. With the modified attention mechanism, our proposed model can scale up to multi-dimensional problems with a large number of grid points and achieve competitive performance compared to state-of-the-art models. Moreover, we show that our factorized attention mechanism can reduce the computational cost and improve stability compared to softmax-free linear attention.[1]

2 Related works

Neural PDE solver Based on the emphases of model design, neural PDE solvers can generally be divided into the following groups. The first group of work focuses on using neural networks with mesh-specific architecture design (such as convolutional layers for uniform mesh, or a graph layer for irregular mesh) to learn the spatial and/or temporal correlation of the PDE data [10, 38, 48, 64, 65, 75, 88, 91, 92, 100, 104, 109, 114, 116]. With input-target data collected, the training process can be conducted without the knowledge of underlying PDEs. This can be appealing when the physics of the system is unknown or partially known, such as large-scale climate modeling [61, 83, 90, 97]. The second group of work, namely the Physics-Informed Neural Networks (PINNs)[15, 40, 42, 52, 77, 87, 96, 106, 127], treat neural networks as a parametrization of the underlying solution function. PINNs incorporate the knowledge of the governing equations into the construction of loss function, which includes the residual of the PDE, the consistency with given boundary condition and initial condition. Unlike the previous group of works, PINNs do not necessarily need input-target data and can be trained solely based on equation loss. The third group of works, often referred to as the neural operator, focuses on learning a mapping between the function spaces[5, 8, 9, 16, 36, 43, 50, 54, 60, 68, 70, 71, 76, 78, 86]. Neural operator has the generalization capability within a family of PDE and can potentially be adapted to different discretization without retraining. DeepONet [76] proposes a practical realization of the universal operator approximation theorem [21]. Concurrent work graph neural operator [69] proposes a learnable kernel integral to approximate the solution operator of parametric PDEs and the follow-up work Fourier Neural Operator (FNO) [68] achieves excelling accuracy and efficiency on certain types of problems. Broadly speaking, the

[1] Code for this project is available at: https://github.com/BaratiLab/FactFormer.

operator learning can be conducted upon different types of function bases, such as the Fourier bases [34, 58, 59, 68, 95, 110, 121], wavelet bases [36], learned bases in an attention layer [16, 67], or based on approximation of the Green's function [7, 108]. The training of neural operators can also be combined with the principle of PINNs to yield a more physically consistent prediction [72, 117]. Our model is closely related to the neural operator, as the major building blocks in our proposed model are a learnable projection operator and a learnable kernel integral operator.

In addition to direct surrogate modeling, neural networks can also be combined with numerical solvers to improve their accuracy and efficiency. For example, using a trained neural network to correct the error of the solver on the fly [3, 28, 56, 89, 113], or doing offline high-fidelity reconstruction [24, 30, 49, 66, 102].

Transformer for Physics Simulation The Transformer model [115] have gained outstanding popularity in natural language modeling [13, 25], imagery data processing [27] and beyond [51]. In the field of physics simulation, Transformer has drawn increasing research interest as a surrogate model for simulation, with its modeling capability demonstrated both as a neural PDE solver [16, 32, 35, 41, 43, 48, 54, 67, 82, 86] and as a pure data-driven model in the absence of a known governing PDE [14, 20, 31, 83]. The dot-product attention can be considered as an approximation of an integral transform with a non-symmetric learnable kernel function [16, 17, 35, 54, 60, 122], which relates Transformer to other popular operator learning models such the FNO [68]. We will expand the discussion of Transformer under the kernel viewpoint in Section 3.

Efficient Transformer Following the introduction of Transformer [115], various works have investigated ways of reducing the computational cost of standard scaled-dot product attention. The first line of work seeks to remove the softmax and make use of matrix associativity to derive linear complexity attention [23, 53, 101], which has also been explored for PDE modeling[16, 43, 67]. The second line of work tries to approximate the dot product between query and key matrix by exploiting the low-rank structure of it [4, 22, 45, 55, 120, 123, 126]. Our work is related to the first group of works with a softmax-free design, but still calculates the dot product between query and key first. Among the second line of work, Axial Transformer is closely related to our work, as both works have explored conducting attention in an axial fashion. However, the derivation of attention matrix is different in the two works (see Section 3.2 for detailed comparison). More generally, the exploitation of the multi-dimensional tensor structure in our proposed model can be related to tensor factorization methods [57, 85] and their applications in various deep learning models [58, 63, 80, 84, 124].

3 Method

3.1 Attention mechanism

Standard attention Given three sets of vectors, namely the queries $\{\mathbf{q}_i\}_{i=1}^{N_q}$, keys $\{\mathbf{k}_i\}_{i=1}^{N_k}$, and values $\{\mathbf{v}_i\}_{i=1}^{N_v}$ (assuming $N_k = N_v$), attention mechanism [2, 33, 79, 115] dynamically computes a weighted average of the values: $\mathbf{z}_i = \sum_{j=1}^{N_v} h(\mathbf{q}_i, \mathbf{k}_j)\mathbf{v}_j$, where $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i \in \mathbb{R}^{1\times d}$, $h(\cdot)$ is the weight function that determines the contribution of a specific value to the final output. An example of $h(\cdot)$ is the scaled-dot product with softmax [115]: $h(\mathbf{q}_i, \mathbf{k}_j) = \exp\left(\mathbf{q}_i\mathbf{k}_j^T/\tau\right)/\sum_s \exp\left(\mathbf{q}_i\mathbf{k}_s^T/\tau\right)$, and τ is usually chosen as $\tau = \sqrt{d}$. The queries/keys/values are usually obtained from inputs via learnable projection. In self-attention, all of them are computed from the same source as follow:

$$\mathbf{q}_i = \mathbf{u}_i W_q, \mathbf{k}_i = \mathbf{u}_i W_k, \mathbf{v}_i = \mathbf{u}_i W_v, \tag{1}$$

where $\mathbf{u}_i \in \mathbb{R}^{1\times d_{\text{in}}}$ is the input vector and $\{W_q, W_k, W_v\} \in \mathbb{R}^{d_{\text{in}}\times d}$ are learnbale projection matrices. In cross-attention, queries are derived from one input while keys and values are derived from another.

Attention as learnable integral Under the hood of PDE modeling, the input sequence to the attention layer can be viewed as the sampling of input function on the discretization grid [16, 54, 60, 67]. Kovachki et al. [60] propose that the scaled-dot product attention [115] can be viewed as a special case of a Neural Operator [60], where the attention amounts to the Monte Carlo approximation of the learnable kernel integral. Cao [16] further proposes two interpretations of softmax-free attention. The first is to view attention as the Fredholm integral equation of the second kind with a learnable asymmetric dot-product kernel, and the second is to view it as a Peterov-Galerkin projection with learnable basis function. The softmax-free attention proposed by Cao [16] is later extended in OFormer [67], where Rotary Positional Encoding (RoPE) [105] is introduced to modulate the dot product and can be viewed as another special case of the kernel integral in Neural Operator style.

In this work, we continue on adopting the learnable kernel integral viewpoint of attention and view each channel of the hidden feature map as the sampling of a specific function on the discretization grid. Given query/key/value matrix $\{Q, K, V\} \in \mathbb{R}^{N\times d}$, their row vectors: $\mathbf{q}_i/\mathbf{k}_i/\mathbf{v}_i$, correspond to the sampling of a set of functions $\{q_l(\cdot), k_l(\cdot), v_l(\cdot)\}_{l=1}^d$ on grid point x_i, where $\{x_i\}_{i=1}^N$ discretizes the underlying domain. As a more concrete example, the l-th column (channel) of $\mathbf{q}_i$, represents the sampling of function $q_l(\cdot)$ on a grid point, i.e. $(\mathbf{q}_i)^l = q_l(x_i)$. Furthermore, softmax-free attention is equivalent to the numerical quadrature of a kernel integral:

$$(\mathbf{z}_i)^l = \sum_{s=1}^{N} w_s(\mathbf{q}_i \cdot \mathbf{k}_s)(\mathbf{v}_s)^l \approx \int_\Omega \kappa\left(x_i, \xi\right) v_l(\xi) d\xi, \tag{2}$$

where $\mathbf{z}_i$ is the output vector, $\kappa\left(x, \xi\right) = \sum_{l=1}^d q_l(x)k_l(\xi)$ is an instance-based kernel and w_s is the quadrature weight. Understanding attention from the perspective of the kernel has been an active topic of research [17, 23, 111, 122]. The theoretical approximation power of different kernel integrals has also been analyzed under the context of PDE learning [35, 54, 60].

Note that the above kernel does not explicitly depend on the spatial coordinates (x_i, ξ). For this work, we opt for a modified kernel formulation proposed in OFormer [67], which modulates the dot product kernel with relative position. Assuming the underlying spatial domain is 1-D (which is sufficient for our proposed model, see next subsection), given query and key vectors $\mathbf{q}_i, \mathbf{k}_j$ and their corresponding spatial coordinates x_i, x_j, RoPE [105] ($g(\cdot,\cdot) : \mathbb{R}^{1\times d} \times \mathbb{R} \mapsto \mathbb{R}^{1\times d}$) is defined as:

$$g\left(\mathbf{q}_i, x_i\right) = \mathbf{q}_i \mathbf{\Theta}\left(x_i\right), \quad g\left(\mathbf{k}_j, x_j\right) = \mathbf{k}_j \mathbf{\Theta}\left(x_j\right) \tag{3}$$
$$\text{where: } \mathbf{\Theta}\left(x_i\right) = \operatorname{Diag}\left(R_1(x_i), \ldots, R_{d/2}(x_i)\right), \quad R_l = \begin{bmatrix} \cos\left(\lambda x_i \theta_l\right) & -\sin\left(\lambda x_i \theta_l\right) \\ \sin\left(\lambda x_i \theta_l\right) & \cos\left(\lambda x_i \theta_l\right) \end{bmatrix},$$

and λ, θ_l are hyperparameters. θ_l is usually chosen as $10000^{-2(l-1)/d}, l \in \{1, 2, \ldots, d/2\}$ following Vaswani et al. [115] and Su et al. [105]. λ is a mesh-based weight that we heuristically set to 64 throughout most problems. The projection function $\Theta(\cdot) : \mathbb{R} \mapsto \mathbb{R}^d \times \mathbb{R}^d$ can explicitly modulate the dot product with relative position: $g\left(\mathbf{q}_i, x_i\right) g\left(\mathbf{k}_j, x_j\right)^T = \mathbf{q}_i \Theta(x_i - x_j) \mathbf{k}_j^T$, thanks to the following property of rotation matrix: $R_l(x_i)R_l(x_j)^T = R_l(x_i - x_j)$.

To summarize, we will adopt attention mechanism in the following form for the proposed model (with modification discussed in the next subsection):

$$Z = w\tilde{Q}\tilde{K}^T V, \tag{4}$$

where $\tilde{\square}$ denotes a matrix whose row vectors are RoPE encoded as in (3), e.g., $\tilde{Q}_i = g(\mathbf{q}_i, x_i)$, w is the quadrature weight with a typical choice of $1/N$ for uniform quadrature rule, Z is the output matrix. The query/key/value matrix $Q/K/V$ is derived from the input via learnable projections defined in (1). The matrix product $\tilde{Q}\tilde{K}^T$ evaluates the kernel function $\kappa(\cdot,\cdot)$ on the discretization grid $\{x_i\}_{i=1}^N$.

3.2 Multidimensional factorized attention

Compared to the standard scaled dot product attention that has quadratic complexity with respect to the length of the input sequence, the attention in (4) can enjoy a linear complexity by making use of the associativity of matrix multiplication (calculate $\tilde{K}^T V$ first). In PDE modeling, the length of the input sequence is equal to the number of points on the underlying discretization grid. Assuming the n-dimensional domain is discretized by $S_1 \times S_2 \times \ldots \times S_n = N$ points, the softmax-free attention in (4) will compute the kernel in (2) with the dot product of Q and K, which are N by d matrices with N usually much larger than d. The kernel matrix computed is by design low-rank as it is the product of two tall and thin matrices. Meanwhile, attending a large number of grid points to each other can be unstable and the linear attention has a complexity that is quadratic to the channel dimension d, which can limit the scalability of the model in terms of its width. To improve the numerical stability and reduce the computational cost of the aforementioned attention mechanism, we propose a simple yet efficient way to modify the kernel integral discussed in the previous section which is motivated by the low-rank structure of attention. Essentially, our model computes the kernel integral in an axial factorized manner instead of convolving over all the grid points in the domain.

For the following discussion, we will use tensor notation [57] to describe the operation. We assume the data is represented on a uniform Eulerian grid and can be treated as n-*way* tensor

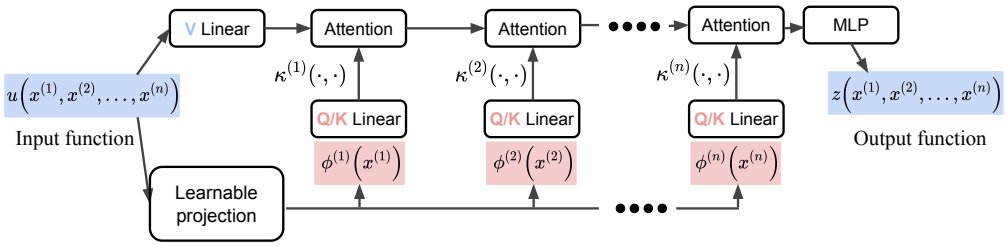


Figure 2: Schematic of the factorized kernel attention. **Upper path**: the input is transformed into the *Value* via a linear transformation. **Lower path**: the input is first projected into multiple sub-functions with a one-dimensional domain. These sub-functions are then used to derive the *Query* and *Key* on each axis, and their dot products form the kernel function of the corresponding axis. The *Value* is iteratively updated by the kernel integral transform along each axis and finally sent to an MLP.

$U \in \mathbb{R}^{S_1 \times S_2 \times \ldots \times S_n}$[2]. The product of it with a matrix $W \in \mathbb{R}^{J \times S_m}$ across the m-th mode will result in a tensor of shape $S_1 \times \ldots \times S_{m-1} \times J \times S_{m+1} \times \ldots \times S_n$, whose elements are defined as:

$$(U \times_m W)_{i_1 i_2 \ldots i_{m-1} j i_{m+1} \ldots i_n} = \sum_{i_m=1}^{S_m} U_{i_1 i_2 \ldots i_m \ldots i_n} W_{j i_m}. \tag{5}$$

Learnable projection The first major component of the proposed framework is a set of learnable integral operators $\{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(n)}\}$ that projects the input function $u : \mathbb{R}^n \mapsto \mathbb{R}^d$ into a set of functions with one-dimensional domain $\{\phi^{(1)}, \phi^{(2)}, \ldots, \phi^{(n)}\} \in \mathbb{R} \mapsto \mathbb{R}^d$, which is defined as:

$$\begin{aligned} \phi^{(m)}(x_i^{(m)}) &= \mathcal{G}^{(m)}(u)(x_i^{(m)}) \\ &= h^{(m)}\left(w \int_{\Omega_1} \cdots \int_{\Omega_n} \gamma^{(m)}\left(u\left(\xi_1, \ldots, \xi_{m-1}, x_i^{(m)}, \xi_{m+1}, \ldots, \xi_n\right)\right) d\xi_1 \ldots d\xi_{m-1} d\xi_{m+1} \ldots d\xi_n\right), \end{aligned} \tag{6}$$

where $h^{(m)}(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}^d$ and $\gamma^{(m)}(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}^d$ are pointwise learnable functions and $w = 1/(L_1 L_2 \cdots L_{m-1} L_{m+1} \cdots L_n)$, with L_m being the size of domain Ω_m discretized by $\{x_i^{(m)}\}_{i=1}^{S_m}$. In practice, we implement $h^{(m)}$ as a three-layer multi-layer perception (MLP) similar to the feedforward network in Transformer [115] and $\gamma^{(m)}$ as a simple linear transformation. When the underlying grid is uniform, (6) simply amounts to first transforming the input with pointwise learnable functions $\gamma^{(m)}(\cdot)$, applying mean pooling over all but the m-th spatial dimension, and then applying another pointwise learnable function $h^{(m)}$.

Factorized kernel integral Equipped with the above projection module, we now introduce our factorized kernel integral scheme. More specifically, we propose to use the following integral to replace the kernel integral in (2):

$$\begin{aligned} & z\left(x_{i_1}^{(1)}, x_{i_2}^{(2)}, \ldots, x_{i_n}^{(n)}\right) \\ &= \int_{\Omega_1} \kappa^{(1)}(x_{i_1}^{(1)}, \xi_1) \int_{\Omega_2} \kappa^{(2)}(x_{i_2}^{(2)}, \xi_2) \cdots \int_{\Omega_n} \kappa^{(n)}(x_{i_n}^{(n)}, \xi_n) v(\xi_1, \xi_2, \ldots, \xi_n) d\xi_1 d\xi_2 \ldots d\xi_n, \end{aligned} \tag{7}$$

where kernels $\{\kappa^{(1)}, \ldots, \kappa^{(n)}\} : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$ are computed based on projected single-dimensional function along each axis, $v(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}^d$ is derived from the input function u via linear transformation (just as the value in standard attention). Next, we will discuss how the above kernel integral is implemented in practice. Using the learnable projection operator defined in (6), we can obtain $\{\hat{U}^{(1)}, \ldots, \hat{U}^{(n)}\}$ ($\hat{U}^{(m)} \in \mathbb{R}^{S_m \times d}$) from the input $U \in \mathbb{R}^{S_1 \times \ldots \times S_n \times d}$, where the i_m-th row of $\hat{U}$ is the evaluation of projected function $\phi^{(m)}(\cdot)$ at x_{i_m}: $\hat{U}_{i_m}^{(m)} = \phi^{(m)}(x_{i_m})$. Then we apply linear transformation on them to obtain the query/key matrix just as standard attention: $Q^{(m)} = \hat{U}^{(m)} W_q^{(m)}, K^{(m)} = \hat{U}^{(m)} W_k^{(m)}$, where $\{W_q^{(m)}, W_k^{(m)}\} \in \mathbb{R}^{d \times d}$ are learnable matrices. The query and key are used to compute the kernel matrix $A^{(m)} \in \mathbb{R}^{S_m \times S_m}$:

$$A^{(m)} = w_m \tilde{Q}^{(m)} \left(\tilde{K}^{(m)}\right)^T, \tag{8}$$

[2]In practice we often have an additional mode for the channel, resulting in a *$(n+1)$-way* tensor.

where w_m is the mesh weight, $\tilde{\square}$ denotes the RoPE encoded matrix as discussed in (4), the i-th row and j-th column of $A^{(m)}$ represents the kernel value $\kappa^{(m)}(x_i^{(m)}, x_j^{(m)})$. Despite (8) has a quadratic complexity with respect to the grid size S_m, this is an affordable cost for most of the problems where the axial grid size S_m is mostly between 64 to 512. Meanwhile, the value $V \in \mathbb{R}^{S_1 \times \ldots \times S_n \times d}$ is derived from the input via a linear transformation (i.e. $(m+1)$-th mode product): $V = U \times_{n+1} W_v$, where $W_v \in \mathbb{R}^{d \times d}$ is again a learnable matrix. The overall factorized kernel integral is numerically approximated with the following tensor-matrix product (Figure 2):

$$Z = \text{Att}(U) = V \times_1 A^{(1)} \times_2 A^{(2)} \times \ldots \times_n A^{(n)}. \tag{9}$$

In (9), the computation of all kernel is of complexity $O(S_1^2 d + S_2^2 d + \ldots + S_n^2 d)$, and the time complexity of a single tensor-matrix product $V \times_m A^{(m)}$ is $O(N S_m d)$. After evaluating the tensor-matrix product, the output tensor Z will be sent to a pointwise feedforward network $f(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}^d$. To sum up, the update protocol of a single layer in our proposed factorized Transformer is defined as follows:

$$U \leftarrow f\left(\text{IN}\left(\text{Att}\left(U\right)\right)\right) + U, \tag{10}$$

where $\text{Att}(\cdot)$ is the attention from (9), $\text{IN}(\cdot)$ is instance normalization [112] that normalizes each channel instance-wise.

It is worth pointing out that the axial factorized kernel proposed here shares some similarities with the Axial Transformer proposed in Ho et al. [45], but has two significant differences despite the connection. Firstly, Axial Transformer reduces the computational cost by constraining the context of attention along each axis (e.g. a pixel can only attend to other pixels on the same row), which amounts to moving all but one axis to the batch dimension. In this way computing the axial kernel matrix is of $O(N S_m d)$ complexity (recall $N = S_1 \times \ldots \times S_n$) instead of $O(S_m^2 d)$ as in our model. And its overall computation of attention is relatively more expensive due to the presence of softmax. Secondly, the decomposition in Axial Transformer is not layer-wise. For example, in the first layer, the attention is conducted in a row-wise manner and then the second block will conduct attention in a column-wise manner, whereas our model decomposes attention along all axes into a tensor-matrix product within every layer. We provide an illustrative example in Figure 27 of the Appendix.

3.3 Training techniques

In this subsection, we will discuss several techniques used for training the model (including baselines) in our numerical experiments. In general, these techniques aim to alleviate the compounding error of autoregressive neural PDE solvers when applied to time-dependent PDEs.

Latent marching It is proposed in the Li et al. [67] that a simple pointwise learnable function $\varepsilon(\cdot, \cdot) \in \mathbb{R}^d \times \mathbb{R}_{>0} \mapsto \mathbb{R}^d$ can be used to propagate dynamics in the latent space with a fixed time interval Δt: $z\left(x, t + \Delta t\right) = z\left(x, t\right) + \varepsilon\left(z\left(x, t\right), t\right)$, where z is the output of the final attention layer. In practice, ε is implemented as a pointwise MLP and is efficient to compute. Leveraging this technique, with one call to the neural solver, we can forward the state for multiple time steps (by marching in the latent space for k steps), thus reducing the total number of calls by a ratio of k. This is in principle similar to the *Temporal Bundling* technique proposed in Message-Passing Neural PDE solver (MP-PDE) [12], yet different in practical realization. In MP-PDE, the multi-timestep prediction is implemented as first predicting the difference in time $\{d_1, d_2, \ldots, d_k\}$ and then adding them to the input u_0 by a forward Euler scheme in the physical space: $\hat{u}_k = u_0 + d_k \Delta t$. In this work, we opt for the latent marching to predict multi-timesteps as the forward Euler scheme (in the physical space) is less stable for fluid problems with relatively large time step sizes.

Pushforward Neural PDE solvers are observed to be unstable for time-dependent problems. A small error or perturbation that occurs at the beginning of neural PDE solvers' prediction, can easily result in an unbounded rollout error. While there is hardly a universal method for guaranteeing their stability, a wide array of techniques have been proposed to improve the stability of neural PDE solvers, such as adding physics constraints [72, 118, 119], rollout training [68] or adding random-walk noise [91, 100, 104]. For this work, we adopt the *pushforward* technique from MP-PDE, which amounts to rolling out the model for two steps during training and then letting the gradient only flows through the last step. This allows training the model on error-corrupted samples and promotes the stability of the model. From a practical perspective, this is straightforward to implement and also computationally much cheaper than standard rollout training.

4 Experiment

In this section, we will investigate our proposed model numerically on several challenging problems. Furthermore, we compare our model against softmax-free attention [16, 67]. The baseline models we compared against are Fourier Neural Operator (FNO) [68], Factorized Fourier Neural Operator (F-FNO) [110] and Dilated ResNet (Dil-ResNet) [44, 125]. FNO has been shown to have good accuracy on a wide range of PDE problems and is computationally very efficient owing to the Fast Fourier Transformation (FFT). F-FNO factorizes the spectral convolution in FNO into separate spectral convolution along different axes and adopt an improved residual connection formulation like Transformer [115]. Dil-ResNet is recently introduced by Stachenfeld et al. [104] to learn the coarse-grained dynamics of turbulent flow and has demonstrated state-of-the-art performance across several problems. We adopt the implementation of Dil-ResNet with group normalization from PDEArena [37]. On 2D steady-state problem where linear attention's computational cost is affordable, we also include the result from Galerkin Transformer [16], which uses CNN to project the function onto a coarse grid and applies linear attention on the coarse grid. The implementation details of the proposed model and baselines are available in Section A, B of the Appendix.

4.1 Benchmark problems

We first apply our model to three fluid-like systems, where the underlying physics patterns are sensitive to the spatiotemporal scale that discretization can resolve, and typically require fine discretization for classical numerical solvers. In these problems, the neural PDE solver is trained to predict the next frame (or multiple frames if using latent marching) given a context of previous frames. The number of context frames of Kolmogorov flow and isotropic turbulence is set to 10 following [68, 72], and 4 for smoke buoyancy similar to [37]. We also consider a well-known steady-state problem-2D Darcy flow, which has been studied in many of the previous works. Below we provide a brief description of each problem we studied. More details can be found in Section E of the Appendix.

2D Kolmogorov flow The first example is 2D Kolmogorov flow governed by incompressible Navier-Stokes equation with a periodic boundary condition. The Reynolds number *Re* determines how turbulent the system will be. We adopt the setting of forced turbulence following Kochkov et al. [56] and generate the data by using the pseudo-spectral method to simulate fluid flow with Reynolds number $Re = 1000$. The objective is to predict the vorticity ω of the flow field within an interval $[t_0, t_0 + T]$, where $T = 1$s and t_0 is a random starting point in the sequence. We use a spatial grid of 256×256 and temporal discretization of $\Delta t = 0.0625$s (therefore 1s corresponds to 16 frames) to train and evaluate the model.

3D isotropic turbulence The second example is 3D isotropic turbulence governed by incompressible Navier-Stokes equation with a periodic boundary condition. The major difference from the first example is that the vortex stretching term is non-zero for three-dimensional flow. We use the 3D spectral simulator from Mortensen and Langtangen [81], which simulates the forced turbulence described in Lamorgese et al. [62]. For generating the dataset, we simulate a system of Taylor Reynolds number $Re_\lambda = 84$ [62]. The objective is to predict the pressure p and velocity $\mathbf{u}$ from $t = 0.5$ to $t = 1$s (10 frames). The model is trained and evaluated on a $60 \times 60 \times 60$ spatial grid with $\Delta t = 0.05$s.

3D smoke buoyancy The third example is 3D buoyancy-driven flow, which depicts smoke volume rising in a closed domain. A similar system in 2D formulation has been studied in several previous works [11, 113]. The underlying governing equation is the incompressible Navier-Stokes equation coupled with an advection equation. The boundary condition for the smoke field is Dirichlet while the boundary condition for the flow field is Neumann. The advection equation describes the motion of smoke, which is transported along the flow field. We modify the solver from [37] that is implemented in *phiflow*[46] to generate the data, with buoyancy factor set to 0.5 and viscosity $\nu = 0.003$. The objective is to predict the scalar density field of smoke d and velocity of flow $\mathbf{u}$ from $t = 3$ to $t = 15$s (16 frames). The model is trained and evaluated on a $64 \times 64 \times 64$ spatial grid with $\Delta t = 0.75$s. To account for non-periodic boundary conditions, we pad the domain for FNO variants and DilResNet following the original works. For FactFormer, we append a simple CNN block after the model, which comprises 3-by-3 convolutional layers with zero padding.

2D Darcy flow In addition to the above time-depedendent systems, the fourth example is 2D steady-state problem from Li et al. [68]. Given the diffusion coefficient, the model predicts the steady-state flow field. The boundary condition is also Dirichlet so we adopt settings for all models similar to the 3D smoke problem.

4.2 Results and discussion

For all the models, we study two protocols of training. The first is Latent Marching with Pushforward (denote as **LM**). The second is simply Autoregressive (denote as **AR**), where the model is rolled out for two steps during training. For LM models, each call to the model will output k future steps. On 2D Kolmogorov flow/3D smoke buoyancy, k is set to 4, and 2 for 3D isotropic turbulence. We interleave pushforward training with standard per-step training for LM models. The relative L^2 norm is used to train and measure the error of each model following Li et al. [68]. The sequence-wise averaged error and the frame-wise error at the end frame are reported in Table 1, 2, 3. We also report the time cost of simulating a sequence and the number of parameters for each model. The frame-wise error trends are shown in Figure 6, 7, 8, 9, 10 in the Appendix. The visualization of predicted samples are provided in the Section F of Appendix.

We observe that Dil-ResNet has a slightly better per-frame fitting capability compared to the other models on 3D flow problems. As shown in the loss trend plots, it starts at a lower error compared to other models. This coincides with the observation in Stachenfeld et al. [104] where Dil-ResNet's performance is strong on 3D fluid problems. On 2D flow, F-FNO has the best accuracy compared to other models. Interestingly, FactFormer can catch up with Dil-ResNet on 2D Kolmogorov flow and 3D smoke buoyancy when the time duration becomes longer. Yet for shorter-term prediction - 3D isotropic turbulence, Dil-ResNet still has the best final accuracy. This suggests that the accuracy of long-term prediction can potentially benefit from exploiting the global structure that lies in the input. Nonetheless, compared to Dil-ResNet, FactFormer offers superior efficiency as indicated by the inference time (time cost of simulating a sequence). Since the training time is roughly proportional to the model forward time, on 3D problems Dil-ResNet generally takes 3-4 times longer to train. In terms of different training strategies, we find that AR models are less stable than multi-step training (LM) and computationally more expensive as it requires more calls to the neural solver. Despite the average error varies case by case, LM models' error generally accumulates slower on the problems we studied, whereas AR models quickly blow up in some cases.

Lastly, while Dil-ResNet has shown good accuracy for 3D flow problems, its performance is highly dependent on the training discretizations. As shown in the Figure 3, without changing model architecture, its evaluation errors increases significantly when the resolution increases, while Transformer-based models and FNO models' performance are roughly invariant to the resolution. This highlights a major difference between CNN-based models and neural operators.

Model	FNO2D		F-FNO2D		Dil-ResNet		FactFormer	
	AR*	LM	AR*	LM	AR	LM	AR	LM
ω avg. error	0.3177	0.2978	**0.1486**	0.2453	0.8156	0.1655	0.8835	0.1734
ω final error	0.4423	0.4567	**0.2811**	0.3861	1.1692	0.3051	1.0963	0.3017
Inf. time (s)	0.73	0.81	0.86	1.01	4.69	1.78	3.14	1.38
# params (M)	85.1		3.7		2.4		3.5	

Table 1: Evaluation results of 2D Kolmogorov flow. A batch size of 10 is used for inference. LM models predict 4 steps with each call to the model. Total prediction length is 16 steps. **AR***: Since for 2D problem FNO variants can afford to rollout more steps during training, AR FNO rollout for 12 steps, AR F-FNO rollout for 6 steps, whereas other AR models rollout for 2 steps during training. For model that has complex parameters, each `cfloat` parameter count as two paramaters.

Model	FNO3D		F-FNO3D		Dil-ResNet		FactFormer	
	AR	LM	AR	LM	AR	LM	AR	LM
p avg. error	0.8080	0.4634	0.3151	0.3264	**0.1725**	0.1778	0.2989	0.2545
p final error	1.1285	0.6522	0.4250	0.4159	0.2573	**0.2448**	0.4407	0.3431
u avg. error	0.3967	0.3382	0.2298	0.2303	**0.1143**	0.1250	0.1775	0.1670
u final error	0.6561	0.4735	0.2799	0.2850	0.1675	**0.1671**	0.2594	0.2218
Inf. time (s)	1.01	0.91	2.77	1.37	12.67	6.89	2.68	1.31
# params (M)	509.8		3.0		6.9		5.1	

Table 2: Evaluation results of 3D isotropic turbulence. A batch size of 4 is used for inference. LM models predict 2 steps with each call to the model. Total prediction length is 10 steps.

Model	FNO3D		F-FNO3D		Dil-ResNet		FactFormer	
	AR	LM	AR	LM	AR	LM	AR	LM
d avg. error	0.1607	0.1344	0.1038	0.1236	**0.0843**	0.0999	0.1017	0.0942
d final error	0.1775	0.1287	0.1415	0.1219	0.1070	0.1062	0.1693	**0.0941**
$\mathbf{u}$ avg. error	0.5198	0.4255	0.3419	0.3713	**0.2378**	0.2747	0.3537	0.2592
$\mathbf{u}$ final error	1.0245	0.6718	0.8655	0.6146	0.5372	0.5023	0.7881	**0.4482**
Inf. time (s)	3.19	1.47	6.35	2.75	27.49	6.94	5.61	2.62
# params (M)	509.8		3.0		6.9		4.6	

Table 3: Evaluation results of 3D smoke buoyancy. A batch size of 4 is used for inference. LM models predict 4 steps with each call to the model. Total prediction length is 16 steps.

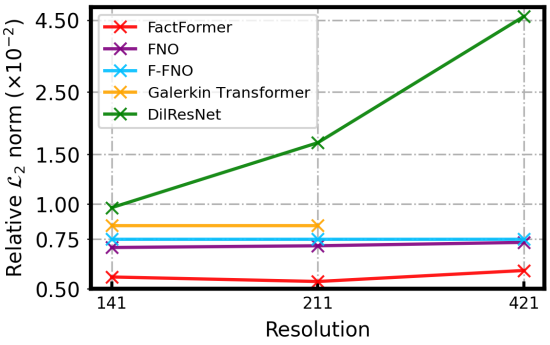


Figure 3: Error on 2D Darcy flow with different training resolutions. Galerkin Transformer's result is taken from the original paper [16].

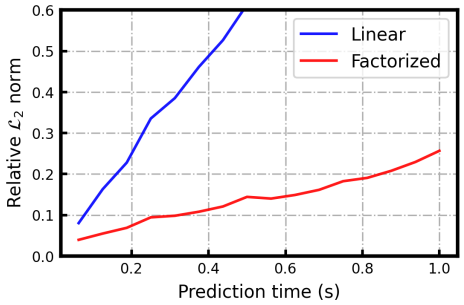


Figure 4: Error trend of different Transformer models on 2D Kolmogorov flow with 128×128 grid.

Model	Avg. rel. L^2 Norm	Fwd.		Fwd. + Bwd.	
		Enc. time (s)	Prop. time (s)	Time (s)	Mem. (MB)
Factorized attention	0.1529	0.0202	0.0112	0.0954	5217
Linear attention (`matmul`)	0.5853	0.1370		0.3013	12029
Linear attention (`einsum`)		0.1333		0.2938	12029

Table 4: Comparison between factorized and linear attention on their forward/backward computational cost, Mem. denotes the peak memory usage. The benchmark is carried out using PyTorch 1.8.2 on an RTX 3090, with a batch size of 4. **Enc. time**: the time spent on obtaining the latent encoding, primarily includes attention layers and feedforward layers after each attention layer; **Prop. time**: the time used to propagate dynamics in the latent space with a 3-layer MLP.

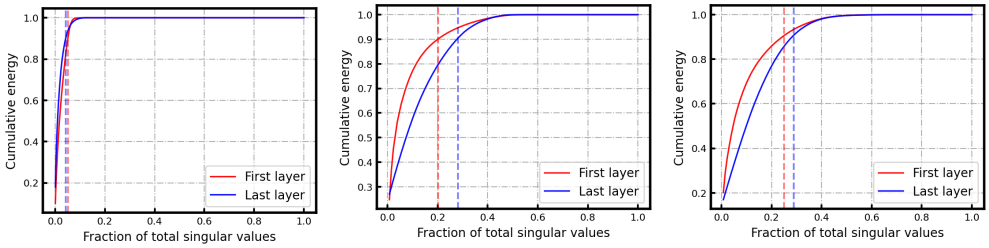
4.3 Comparison against full attention

In this subsection, we will present an ablation study of the proposed factorized attention mechanism with softmax-free attention (denoted as "linear attention") previously applied to PDE modeling[16, 67]. More specifically, we employ the attention from Li et al. [67] (in the form of (4)) to replace factorized attention in (10), with $\tilde{K}, V$ normalized column-wise via instance normalization, e.g. $||V_{\cdot,j}||_2 = 1$. To accommodate for the memory cost of linear attention, we further downsample the 2D Kolmogorov flow discussed in the last subsection to a 128×128 grid and train both linear and factorized attention models on it (with latent marching and pushforward trick).

Comparison of performance We compare the accuracy and computational cost of the two attention mechanisms in Figure 4 and Table 4. While in principle full attention could have better approximation capacity than factorized attention, in practice we find that it performs worse than factorized attention on this problem we studied. Specifically, its rollout is less stable and results in a degraded accuracy. We hypothesize that this is due to the instability of iteratively calculating the attention matrix of a large size, as rolling out the prediction requires recursively calling the model multiple times. In

addition to the accuracy improvement, the benchmark on computation empirically demonstrates the computational efficiency improvement of factorized attention over linear attention. We provide more detailed comparison between factorized attention and linear attention in the Section D of Appendix, where we observed consistent efficiency improvement with different grid sizes and model sizes.

Pattern of attention matrices We also investigate the structure of different attention matrices. By construction, when using softmax-free attention to compute the kernel integral in (2), the kernel matrix $A = QK^T$ is going to have a low-rank structure since $\text{rank}(A) \leq \min(\text{rank}(Q), \text{rank}(K))$, $Q, K \in \mathbb{R}^{N \times d}$ and usually$N >> d$. After training, we compute the attention matrices based on 100 samples and conduct singular value decomposition (SVD) on them. We define the total energy of the spectrum as the sum of singular values $E = \sum_i \sigma_i$, where σ_i is the i-th singular value and report the normalized cumulative energy histogram $b_k = \sum_{i=1}^{k} \sigma_i / \sum_i \sigma_i$ in Figure 5a, 5b, 5c. For each layer, the histogram is averaged across the attention matrices of all heads. We observe that for linear attention, its rank is relatively low as less than 5% of the singular values capture over 90% of the total energy, which is similar to the trend observed from previous works studying the rank of standard softmax-attention [6, 26, 120]. Note that the spectrum of linear attention is based on a truncated SVD and therefore its rank will be even lower if a full SVD is performed. The highly low-rank structure of the full attention matrix hints the potential to approximate with or decomposed into smaller and more compact matrices, and our proposed factorized scheme is one example.



(a) Full attention matrix A (b) Attention matrix $A^{(1)}$ (c) Attention matrix $A^{(2)}$

Figure 5: Spectrum of different attention matrices. **(a)**: Attention matrices from softmax-free attention. **(b), (c)**: Attention matrices for axis x and y from FactFormer. The vertical line indicates the fraction of singular values that capture 90% of the total energy. Since the full attention matrix A has a relatively large size (16384×16384). Therefore its spectrum is computed via TruncatedSVD [39] with top 1024 components truncated.

5 Conclusion

In this work, we propose an end-to-end Transformer for PDE modeling, which features a learnable projection operator and a factorized kernel integral. We demonstrate that the proposed model balances efficiency and accuracy well, making it a promising and scalable solution for PDE surrogate modeling. However, the proposed attention mechanism is still not free from the curse of dimensionality. The computation of the factorized kernel integral requires evaluating the function on all $S_1 \times S_2 \times \ldots \times S_m$ grid points. A future direction could be extending the factorization scheme to a more efficient tensor decomposition format like tensor-train. The proposed model currently exploits the uniform structure of the underlying grids and use mean pooling when doing projection, but non-uniform quadrature weight will be necessary when applying to non-uniform grids. It is also observed that the proposed model and other neural PDE solvers can be unstable due to the error accumulation when solving time-dependent systems.

Acknowledgement

This work is supported by the National Science Foundation under Grant No. 1953222. The authors would like to thank the annonymous reviewers and area chair for their efforts and valuable feedback during the reviewing process. The authors would like to thank Dr. Shuhao Cao from University of Missouri - Kansas City for the comments regarding the backpropagation of feedforward layer and attention layer in Transformer. The authors would also like to thank Zhi Ye for the suggestions on benchmarking the computational cost of the model.

References

[1] *Large-Scale Dynamics and Transition to Turbulence in the Two-Dimensional Kolmogorov Flow*, pages 374–396. doi: 10.2514/5.9781600865831.0374.0396. URL https://arc.aiaa.org/doi/abs/10.2514/5.9781600865831.0374.0396.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL https://arxiv.org/abs/1409.0473.

[3] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

[4] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[5] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M. Stuart. Model reduction and neural networks for parametric pdes, 2021.

[6] Srinadh Bhojanapalli, Ayan Chakrabarti, Himanshu Jain, Sanjiv Kumar, Michal Lukasik, and Andreas Veit. Eigen analysis of self-attention and its reconstruction from partial computation, 2021.

[7] Nicolas Boullé, Seick Kim, Tianyi Shi, and Alex Townsend. Learning green's functions associated with time-dependent partial differential equations. *Journal of Machine Learning Research*, 23(218):1–34, 2022.

[8] Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K Gupta. Clifford neural layers for pde modeling. *arXiv preprint arXiv:2209.04934*, 2022.

[9] Johannes Brandstetter, Max Welling, and Daniel E Worrall. Lie point symmetry data augmentation for neural PDE solvers. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 2241–2256. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/brandstetter22a.html.

[10] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

[11] Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K. Gupta. Clifford neural layers for pde modeling, 2023.

[12] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers, 2023.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[14] Salva Rühling Cachay, Peetak Mitra, Haruki Hirasawa, Sookyung Kim, Subhashis Hazarika, Dipti Hingmire, Phil Rasch, Hansi Singh, and Kalai Ramea. Climformer–a spherical transformer model for long-term climate projections.

[15] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12):1727–1738, 2021.

[16] Shuhao Cao. Choose a transformer: Fourier or galerkin. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24924–24940. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/d0921d442ee91b896ad95059d13df618-Paper.pdf.

[17] Shuhao Cao, Peng Xu, and David A. Clifton. How to understand masked autoencoders, 2022.

[18] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[19] Gary J. Chandler and Rich R. Kerswell. Invariant recurrent solutions embedded in a turbulent two-dimensional kolmogorov flow. *Journal of Fluid Mechanics*, 722:554–595, 2013. doi: 10.1017/jfm.2013.122.

[20] Ashesh Chattopadhyay, Mustafa Mustafa, Pedram Hassanzadeh, and Karthik Kashinath. Deep spatial transformers for autoregressive data-driven forecasting of geophysical turbulence. In *Proceedings of the 10th international conference on climate informatics*, pages 106–112, 2020.

[21] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995. doi: 10.1109/72.392253.

[22] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019.

[23] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2022.

[24] Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics*, 36(4):1–14, jul 2017. doi: 10.1145/3072959.3073643. URL https://doi.org/10.1145%2F3072959.3073643.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[26] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth, 2021.

[27] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[28] Gideon Dresdner, Dmitrii Kochkov, Peter Norgaard, Leonardo Zepeda-Núñez, Jamie A. Smith, Michael P. Brenner, and Stephan Hoyer. Learning to correct spectral methods for simulating turbulent flows, 2022.

[29] Antonio H de O Fonseca, Emanuele Zappala, Josue Ortega Caro, and David van Dijk. Continuous spatiotemporal transformers. *arXiv preprint arXiv:2301.13338*, 2023.

[30] Kai Fukami, Koji Fukagata, and Kunihiko Taira. Super-resolution reconstruction of turbulent flows with machine learning. *Journal of Fluid Mechanics*, 870:106–120, 2019. doi: 10.1017/jfm.2019.238.

[31] Zhihan Gao, Xingjian Shi, Hao Wang, Yi Zhu, Yuyang Bernie Wang, Mu Li, and Dit-Yan Yeung. Earthformer: Exploring space-time transformers for earth system forecasting. *Advances in Neural Information Processing Systems*, 35:25390–25403, 2022.

[32] Nicholas Geneva and Nicholas Zabaras. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, 2022.

[33] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. URL https://arxiv.org/abs/1410.5401.

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

[35] Ruchi Guo, Shuhao Cao, and Long Chen. Transformer meets boundary value inverse problems. *arXiv preprint arXiv:2209.14977*, 2022.

[36] Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations, 2021.

[37] Jayesh K. Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling, 2022.

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

[39] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, 2010.

[40] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34): 8505–8510, 2018.

[41] Xu Han, Han Gao, Tobias Pfaff, Jian-Xun Wang, and Li-Ping Liu. Predicting physics in mesh-reduced space with temporal attention. *arXiv preprint arXiv:2201.09113*, 2022.

[42] Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. Physics-informed machine learning: A survey on problems, methods and applications, 2023.

[43] Zhongkai Hao, Chengyang Ying, Zhengyi Wang, Hang Su, Yinpeng Dong, Songming Liu, Ze Cheng, Jun Zhu, and Jian Song. Gnot: A general neural operator transformer for operator learning. *arXiv preprint arXiv:2302.14376*, 2023.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[45] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers, 2020. URL https://openreview.net/forum?id=H1e5GJBtDr.

[46] Philipp Holl, Vladlen Koltun, and Kiwon Um. phiflow: A differentiable pde solving framework for deep learning via physical simulations.

[47] Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5): 2002–2041, 2020.

[48] Steeven JANNY, Aurélien Bénéteau, Madiha Nadri, Julie Digne, Nicolas THOME, and Christian Wolf. EAGLE: Large-scale learning of turbulent fluid dynamics with mesh transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=mfIX4QpsARJ.

[49] Chiyu "Max" Jiang, Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A. Tchelepi, Philip Marcus, Mr Prabhat, and Anima Anandkumar. Meshfreeflownet: A physics-constrained deep continuous space-time super-resolution framework. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2020. doi: 10.1109/SC41405.2020.00013.

[50] Pengzhan Jin, Shuai Meng, and Lu Lu. Mionet: Learning multiple-input operators via tensor product. *SIAM Journal on Scientific Computing*, 44(6):A3490–A3514, 2022.

[51] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

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

[53] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

[54] Georgios Kissas, Jacob Seidman, Leonardo Ferreira Guilhoto, Victor M. Preciado, George J. Pappas, and Paris Perdikaris. Learning operators with coupled attention, 2022.

[55] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

[56] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), may 2021. doi: 10.1073/pnas.2101784118. URL https://doi.org/10.1073%2Fpnas.2101784118.

[57] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500, aug 2009. ISSN 0036-1445. doi: 10.1137/07070111X. URL https://doi.org/10.1137/07070111X.

[58] Jean Kossaifi, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, and Anima Anandkumar. Multi-grid tensorized fourier neural operator for high resolution PDEs, 2023. URL https://openreview.net/forum?id=po-oqRst4Xm.

[59] Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators, 2021.

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

[61] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Alexander Pritzel, Suman Ravuri, Timo Ewalds, Ferran Alet, Zach Eaton-Rosen, Weihua Hu, Alexander Merose, Stephan Hoyer, George Holland, Jacklynn Stott, Oriol Vinyals, Shakir Mohamed, and Peter Battaglia. Graphcast: Learning skillful medium-range global weather forecasting, 2022.

[62] A. G. Lamorgese, D. A. Caughey, and S. B. Pope. Direct numerical simulation of homogeneous turbulence with hyperviscosity. *Physics of Fluids*, 17(1), 12 2004. ISSN 1070-6631. doi: 10.1063/1.1833415. URL https://doi.org/10.1063/1.1833415. 015106.

[63] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition, 2015.

[64] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids, 2019.

[65] Zijie Li and Amir Barati Farimani. Graph neural network-accelerated lagrangian fluid simulation. *Computers & Graphics*, 103:201–211, 2022. ISSN 0097-8493. doi: https://doi.org/10.1016/j.cag.2022.02.004. URL https://www.sciencedirect.com/science/article/pii/S0097849322000206.

[66] Zijie Li, Tianqin Li, and Amir Barati Farimani. TPU-GAN: Learning temporal coherence from dynamic point cloud sequences. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=FEBFJ98FKx.

[67] Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=EPPqt3uERT.

[68] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[69] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations, 2020.

[70] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations, 2020.

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

[72] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations, 2023.

[73] Xinliang Liu, Bo Xu, and Lei Zhang. Ht-net: Hierarchical transformer based operator learning model for multiscale pdes. *arXiv preprint arXiv:2210.10890*, 2022.

[74] Xinliang Liu, Bo Xu, and Lei Zhang. Mitigating spectral bias for the multiscale operator learning with hierarchical attention, 2023.

[75] Winfried Lötzsch, Simon Ohler, and Johannes Otterbach. Learning the solution operator of boundary value problems using graph neural networks. In *ICML 2022 2nd AI for Science Workshop*, 2022. URL https://openreview.net/forum?id=4vx9FQA7wiC.

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

[77] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228, 2021.

[78] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, apr 2022. doi: 10.1016/j.cma.2022.114778. URL https://doi.org/10.1016%2Fj.cma.2022.114778.

[79] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015. URL https://arxiv.org/abs/1508.04025.

[80] Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Dawei Song, and Ming Zhou. A tensorized transformer for language modeling, 2019.

[81] Mikael Mortensen and Hans Petter Langtangen. High performance python for direct numerical simulations of turbulent flows. *Computer Physics Communications*, 203:53–65, jun 2016. doi: 10.1016/j.cpc.2016.02.005. URL https://doi.org/10.1016%2Fj.cpc.2016.02.005.

[82] Tan Nguyen, Minh Pham, Tam Nguyen, Khai Nguyen, Stanley Osher, and Nhat Ho. Fourierformer: Transformer meets generalized fourier integral theorem. *Advances in Neural Information Processing Systems*, 35:29319–29335, 2022.

[83] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K. Gupta, and Aditya Grover. Climax: A foundation model for weather and climate, 2023.

[84] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks, 2015.

[85] I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. doi: 10.1137/090752286. URL https://doi.org/10.1137/090752286.

[86] Oded Ovadia, Adar Kahana, Panos Stinis, Eli Turkel, and George Em Karniadakis. Vito: Vision transformer-operator. *arXiv preprint arXiv:2303.08891*, 2023.

[87] Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[88] Pranshu Pant, Ruchit Doshi, Pranav Bahl, and Amir Barati Farimani. Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations. *Physics of Fluids*, 33 (10):107101, oct 2021. doi: 10.1063/5.0062546. URL https://doi.org/10.1063%2F5.0062546.

[89] Jaideep Pathak, Mustafa Mustafa, Karthik Kashinath, Emmanuel Motheau, Thorsten Kurth, and Marcus Day. Using machine learning to augment coarse-grid computational fluid dynamics simulations. *arXiv preprint arXiv:2010.00072*, 2020.

[90] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, Pedram Hassanzadeh, Karthik Kashinath, and Animashree Anandkumar. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators, 2022.

[91] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks, 2021.

[92] Lukas Prantl, Benjamin Ummenhofer, Vladlen Koltun, and Nils Thuerey. Guaranteed conservation of momentum for learning particle-based fluid dynamics, 2022.

[93] Alfio Quarteroni and Alberto Valli. *Domain decomposition methods for partial differential equations*. Number BOOK. Oxford University Press, 1999.

[94] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf.

[95] Md Ashiqur Rahman, Zachary E. Ross, and Kamyar Azizzadenesheli. U-no: U-shaped neural operators, 2023.

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

[97] Stephan Rasp, Peter D. Dueben, Sebastian Scher, Jonathan A. Weyn, Soukayna Mouatadid, and Nils Thuerey. WeatherBench: A benchmark data set for data-driven weather forecasting. *Journal of Advances in Modeling Earth Systems*, 12(11), nov 2020. doi: 10.1029/2020ms002203. URL https://doi.org/10.1029%2F2020ms002203.

[98] Robert S. Rogallo. Numerical experiments in homogeneous turbulence. 1981.

[99] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[100] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020.

[101] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. Efficient attention: Attention with linear complexities, 2020.

[102] Dule Shu, Zijie Li, and Amir Barati Farimani. A physics-informed diffusion model for high-fidelity flow field reconstruction. *Journal of Computational Physics*, 478:111972, 2023.

[103] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587): 484–489, 2016.

[104] Kimberly Stachenfeld, Drummond B. Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned coarse models for efficient turbulence simulation, 2022.

[105] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2022.

[106] Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

[107] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020.

[108] Jingwei Tang, Vinicius C Azevedo, Guillaume Cordonnier, and Barbara Solenthaler. Neural green's function for laplacian systems. *Computers & Graphics*, 107:186–196, 2022.

[109] Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020.

[110] Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators, 2023.

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

[112] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[113] Kiwon Um, Robert Brand, Yun Raymond Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Advances in Neural Information Processing Systems*, 33:6111–6122, 2020.

[114] Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1lDoJSYDH.

[115] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[116] Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 1457–1466, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403198. URL https://doi.org/10.1145/3394486.3403198.

[117] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science Advances*, 7(40): eabi8605, 2021. doi: 10.1126/sciadv.abi8605. URL https://www.science.org/doi/abs/10.1126/sciadv.abi8605.

[118] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets, 2021.

[119] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks, 2022.

[120] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[121] Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

[122] Matthew A Wright and Joseph E Gonzalez. Transformers are deep infinite-dimensional non-mercer binary kernel machines. *arXiv preprint arXiv:2106.01506*, 2021.

[123] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention, 2021.

[124] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification, 2017.

[125] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2016.

[126] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2021.

[127] Min Zhu, Handi Zhang, Anran Jiao, George Em Karniadakis, and Lu Lu. Reliable extrapolation of deep neural operators informed by physics or sparse observations. *Computer Methods in Applied Mechanics and Engineering*, 412:116064, 2023.

A Model implementation details

The major hyperparameters are listed in Table 5.

Hyperparameter	2D Kolmogorov	3D turbulence	3D smoke	2D Darcy flow
Hidden dimension	128	128	128	128
Depth	4	4	3	3
Heads	8	6	6	12
Kernel dimension	128	192	192	128
Input encoder	2D Conv	2D Conv	2D Conv	MLP
Output decoder	MLP	MLP	MLP	MLP

Table 5: Major hyperparameters for FactFormer

Hidden dimension indicates the number of channels in the latent space. *Depth* denotes the number of attention layers. Before entering each attention layer, we modulate the latent encoding with positional encoding: $z_i \leftarrow z_i + \psi(x_i)$, where x_i is the Cartesian coordinate of latent encoding z_i, $\psi : \mathbb{R}^n \mapsto \mathbb{R}^d$ is a random Fourier feature mapping [94, 107] with a learnable linear transformation. *Kernel dimension* is the dimension of each head, which is equivalent to d_k, the number of function bases used to compute the kernel: $\kappa(x, \xi) = \sum_l^{d_k} q_l(x) k_l(\xi)$. We train the model with AdamW optimizer and cyclic learning rate scheduler with a maximum learning rate $3e-4$, similar to prior Transformer-for-PDE works [16, 43, 67].

For 3D smoke buoyancy and 2D Darcy flow problem, we append a CNN block after attention layers to better account for the boundary values. The CNN block has a U-shape arrangement with 4 CNN layers, with all layers using a kernel size of 3 and padding size of 1 (pad with zeros). The first CNN layer has a stride of 2, while other layers have a stride of 1. The stride-2 convolution will downsample the data by half, so nearest upsampling is applied between the second and third CNN layers to recover the spatial resolution.

On top of every model (including baselines we will discuss in the next section), we use a 2D convolutional layer to compress the temporal dimension if it is a time-dependent problem. Concretely, we first reshape the input into (N, T_{in}) where N is the number of spatial grid points and T_{in} is the number of input frames. Then we apply 2D convolution filters of size $(1, T_{\text{in}})$ to compress the temporal dimension to 1. At the bottom of every model, we use a three-layer MLP to project the latent encoding back to variables of interest such as pressure and velocities. In addition, we adopt a curriculum training strategy for all latent marching models, where we only march for 1 step at the beginning of the training and don't do any pushforward. Then we gradually increase the latent marching steps throughout the training and apply pushforward when the model has been trained for around 6% of the total epochs.

B Baseline implementation details

In this section, we provide the full details of baseline models, namely FNO, F-FNO, Dil-ResNet, and linear attention Transformer.

For Fourier Neural Operator, the implementation is taken from Li et al. [72]'s official implementation: https://github.com/neuraloperator/physics_informed. And for Factorized Fourier Neural Operator the implementation is taken from https://github.com/alasdairtran/fourierflow. We add group normalization before the final fully-connected layer. We use a hidden size of 96, a mode number of 12 for 3D problems, 24 for 2D turbulence, 20 for Darcy flow, and a layer number of 4.

For Dil-ResNet, we adopt the implementation from Gupta and Brandstetter [37]:https://github.com/microsoft/pdearena. Compared to Gupta and Brandstetter [37] and Stachenfeld et al. [104], we simplify the setting by truncating the number of layers inside each block, where we use dilation layers $[1, 3, 8, 3, 1]$ ($[1, 2, 4, 2, 1]$ is chosen for the 3D problem as it has slightly better performance) inside each block instead of $[1, 2, 4, 8, 4, 2, 1]$. The primary reason is that without truncation, the training on 3D problems will take over a week (and cannot fit into a single A6000 GPU for 2-step

rollout training), which is significantly slower than other models. In summary, we use 3 residual blocks (each with 5 CNN layers of width 128), an MLP-based or 2D convolution-based (for convolution in temporal domain) encoder/decoder, with group normalization inserted between every residual block. The implementation of the linear attention Transformer follows OFormer's [67] attention implementation https://github.com/BaratiLab/OFormer, with a Galerkin style normalization scheme [16]. Other hyperparameters are kept the same as the hyperparameters listed in Table 5 - 2D Kolmogorov. For non-Transformer models, they are trained with Adam optimizer and decay learning rate from $5e-4$ to $5e-6$ throughout the training.

All the experiments are carried out using PyTorch 1.8 except for FNO/F-FNO experiment, which uses PyTorch 1.13 for optimizing complex-valued parameters. We train all LM models for 100k iterations and AR models for 64k iterations of gradient updates.

C Visualization of error trend

This section includes the average frame-wise error trend for the time-dependent systems we have investigated.

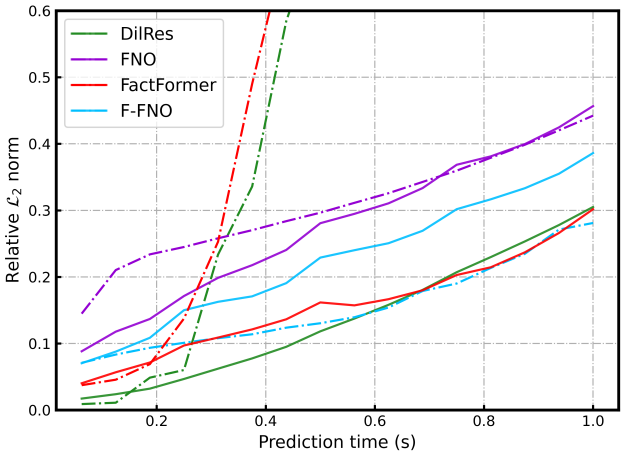


Figure 6: Error trend of vorticity ω on 2D Kolmogorov flow.
Dashed line: AR; **Solid line**: LM

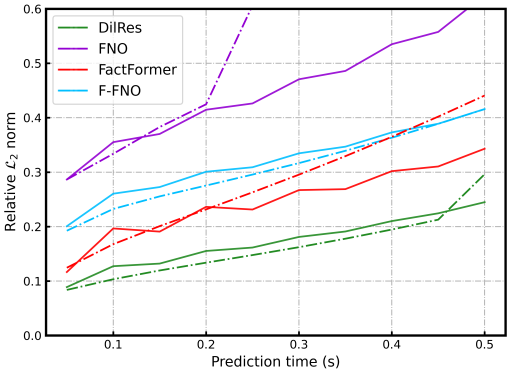


Figure 7: Error trend of pressure p on 3D isotropic turbulence.

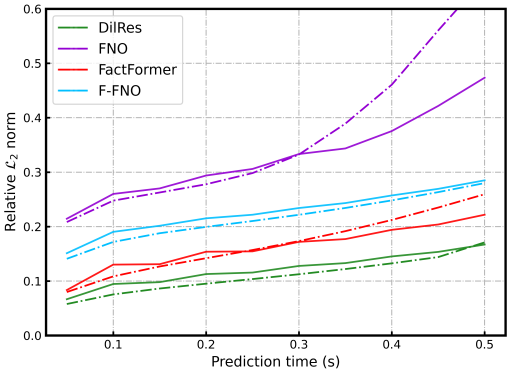


Figure 8: Error trend of velocity $\mathbf{u}$ on 3D isotropic turbulence.

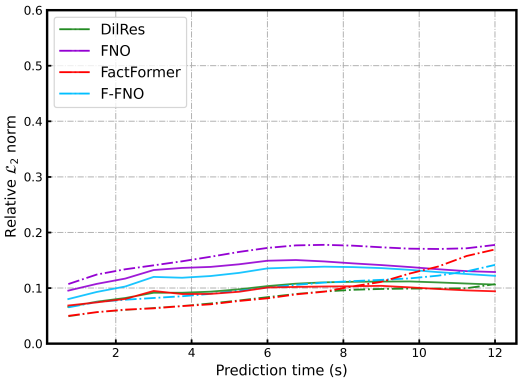


Figure 9: Error trend of marker field d on 3D smoke buoyancy.

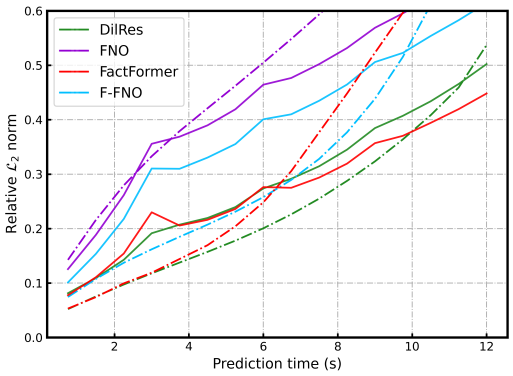


Figure 10: Error trend of velocity field $\mathbf{u}$ on 3D smoke buoyancy.

D Further ablation study

This section includes further study and numerical experiments on the proposed model.

Note on backpropagation Consider a single (softmax-free) attention block that consists of a attention layer and a two layer feedforward network (for simplicity we consider single-headed case):

$$U_1 = \text{Att}(U_0), \quad U_2 = \sigma(U_1 W_1) W_2, \tag{11}$$

where $U_0 \in \mathbb{R}^{N\times d}$ is the input, $W_1, W_2 \in \mathbb{R}^{d\times d}$ are learnable weights for the feedforward network. Given a loss function $l(\cdot) : \mathbb{R}^{N\times d} \mapsto \mathbb{R}$ (for example, mean squared error), define $\tilde{U_1} := U_1 W_1$, $\frac{\partial l}{\partial U_2} := h$, $\frac{\partial \sigma(\tilde{U_1})}{\partial \tilde{U_1}} := g$, the gradient for the weights and input in the feedforward network are ($\odot$ denotes element-wise multiplication):

$$\frac{\partial l}{\partial W_2} = \sigma(\tilde{U_1})^T h \tag{12}$$

$$\frac{\partial l}{\partial W_1} = U_1^T (h W_2^T \odot g) \tag{13}$$

$$\frac{\partial l}{\partial U_1} = (h W_2^T \odot g) W_1^T \tag{14}$$

For linear (softmax-free) dot-product attention: $U_1 = \text{Att}(U_0) = QK^T V$, where $Q = U_0 W_q, K = U_0 W_k, V = U_0 W_v$, and thus $U_1 = U_0 W_q W_k^T U_0^T U_0 W_v$, the gradient of weights are:

$$\frac{\partial l}{\partial W_q} = U_0^T \frac{\partial l}{\partial U_1} W_v^T U_0^T U_0 W_k, \tag{15}$$

$$\frac{\partial l}{\partial W_k} = U_0^T U_0 W_v (\frac{\partial l}{\partial U_1})^T U_0 W_q, \tag{16}$$

$$\frac{\partial l}{\partial W_v} = U_0^T U_0 W_k W_q^T U_0^T \frac{\partial l}{\partial U_1}, \tag{17}$$

where $U_0 \in \mathbb{R}^{N\times d}$ has appeared three times in each calculation and thus in backpropagation the computational cost of attention layer is generally more expensive than the feedforward layer where it involves more matrices that grow exponentially with respect to the spatial resolution.

Next we provide a comparison between the backpropagation of linear attention and the proposed factorized attention in scalar summation form. For simplicity, we only consider the attention layer. For linear attention, it can be written as:

$$Z_{i,c} = \sum_{m=1}^{d} Q_{i,m} (\sum_{j=1}^{N} K_{j,m} V_{j,c}), \tag{18}$$

where $Z_{i,c}$ denotes the i-th row and j-th column of matrix Z and similar for other matrices, $Q = XW_q, K = XW_k, V = XW_v$ and $X \in \mathbb{R}^{N\times d}$ is input. The gradient of parameters are computed as:

$$\frac{\partial Z_{i,c}}{\partial (W_q)_{r,s}} = X_{i,r}(\sum_{j=1}^{N} K_{j,s}V_{j,c}), \tag{19}$$

$$\frac{\partial Z_{i,c}}{\partial (W_k)_{r,s}} = Q_{i,s}(\sum_{j=1}^{N} X_{j,r}V_{j,c}), \tag{20}$$

$$\frac{\partial Z_{i,c}}{\partial (W_v)_{r,s}} = \begin{cases} \sum_{m=1}^{d} Q_{i,m}(\sum_{j=1}^{N} K_{j,m}X_{i,r}) & : \text{if } s = c \\ 0 & : \text{otherwise.} \end{cases} \tag{21}$$

For the proposed factorized attention, it can written as:

$$Z_{i,c} = \sum_{j_1=1}^{S_1}\sum_{j_2=1}^{S_2}\dots\sum_{j_n=1}^{S_n} A^{(1)}_{i_1,j_1}A^{(2)}_{i_2,j_2}\dots A^{(n)}_{i_n,j_n}V_{j,c}\,, \quad j := (j_1, j_2, \dots, j_n),\ i := i_1, i_2, \dots, i_n \tag{22}$$

where $A^{(m)} = Q^{(n)}(K^{(m)})^T$ ($A^{(m)} \in \mathbb{R}^{S_m\times S_m}, N = S_1 \times S_2 \times \dots \times S_n$) is the axial kernel matrix as defined in (8), $X^{(m)} \in \mathbb{R}^{S_m\times d}$ is the axial projection along m-th axis as defined in (6).

For W_v, its gradient is computed as:

$$\frac{\partial Z_{i,c}}{\partial (W_v)_{r,s}} = \begin{cases} \sum_{j_1=1}^{S_1}\sum_{j_2=1}^{S_2}\dots\sum_{j_n=1}^{S_n} A^{(1)}_{i_1,j_1}A^{(2)}_{i_2,j_2}\dots A^{(n)}_{i_n,j_n}X_{j,r} & : \text{if } s = c \\ 0 & : \text{otherwise.} \end{cases} \tag{23}$$

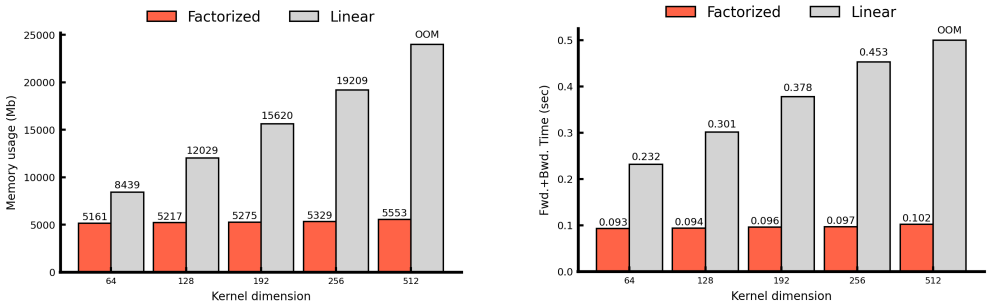
For $W_q^{(n)}, W_k^{(n)}$, their gradients are computed as:

$$\frac{\partial Z_{i,c}}{\partial (W_q^{(n)})_{r,s}} = X^{(n)}_{i_n,r}\sum_{j_1=1}^{S_1}\sum_{j_2=1}^{S_2}\dots\sum_{j_n=1}^{S_n} A^{(1)}_{i_1,j_1}A^{(2)}_{i_2,j_2}\dots A^{(n-1)}_{i_{n-1},j_{n-1}}V_{j,c}K^{(n)}_{j_n,s}, \tag{24}$$

$$\frac{\partial Z_{i,c}}{\partial (W_k^{(n)})_{r,s}} = Q^{(n)}_{i_n,s}\sum_{j_1=1}^{S_1}\sum_{j_2=1}^{S_2}\dots\sum_{j_n=1}^{S_n} A^{(1)}_{i_1,j_1}A^{(2)}_{i_2,j_2}\dots A^{(n-1)}_{i_{n-1},j_{n-1}}V_{j,c}X^{(n)}_{j_n,r}, \tag{25}$$

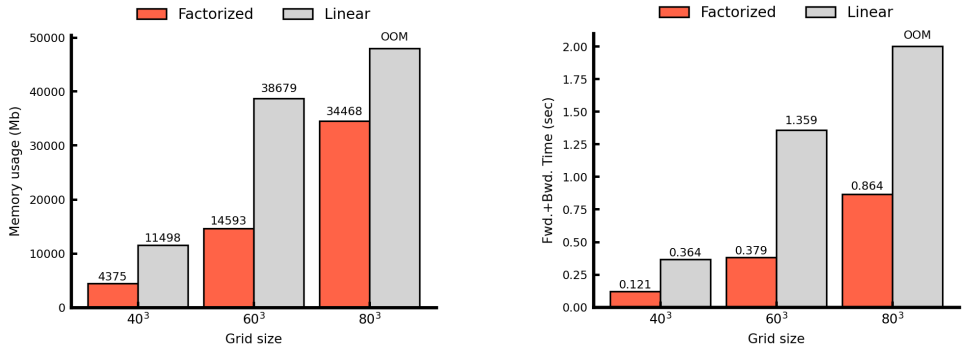
The major difference is that for factorized attention the summation is taken over each axis separately while for linear attention is taken over all N grid points.

Runtime comparison As discussed in Section A, the kernel dimension indicates how many function bases are used to evaluate the kernel and a larger kernel dimension is beneficial to the learning capacity of the model. As shown in Figure 11a, 11b, linear attention's training cost increases more significantly than the factorized attention as kernel dimension increases, since its complexity is quadratic with respect to kernel dimension. Factorized attention's computational efficiency can be further improved by reducing the spatial resolution, leveraging techniques such as learning the mapping in the latent space (similar to latent diffusion model [99]), multi-scale network architecture that resembles multigrid methods [35, 74], or domain decomposition [47, 93]. Furthermore, the training cost of factorized attention is also relatively lower than linear attention on 3D domain as shown in Figure 12a, 12b.



(a) Forward + backward peak memory usage. (b) Forward + backward time per iteration.

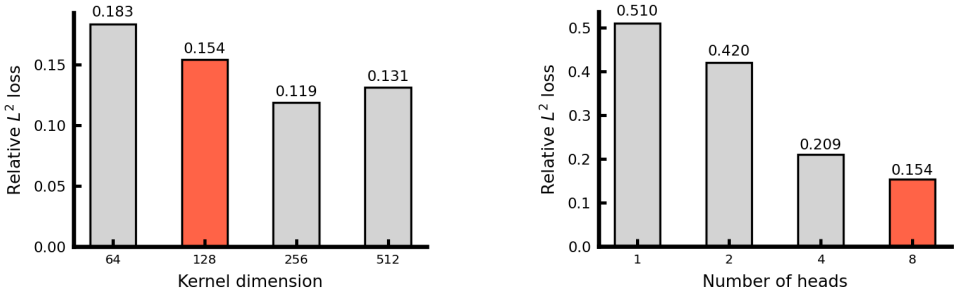
Figure 11: Benchmark of factorized attention and linear attention on 2D domain (with 128×128 grid) with varying kernel dimension. Benchmark is done on an RTX 3090 with PyTorch 1.8.2 and a batch size of 4. Hyperparameter setting is the same as in Table 5-2D Kolmogorov flow. "OOM" denotes out of memory.



(a) Forward + backward peak memory usage. (b) Forward + backward time per iteration.

Figure 12: Benchmark of factorized attention and linear attention on 3D domain with varying grid size. Benchmark is done on an A6000 with PyTorch 1.8.2 and batch size of 1. Hyperparameter setting is the same as in Table 5-3D isotropic turbulence.

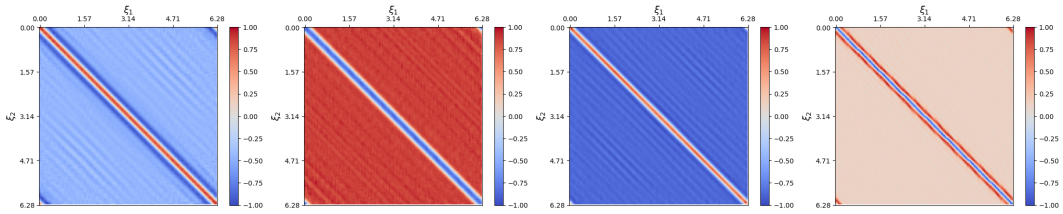
Model scaling performance We study the impact of the number of heads and size of kernel dimension on prediction loss. For each direction of hyperparameter search, we fix the value of other hyperparameters to that shown in Table 5. The ablation experiments are conducted on 2D Kolmogorov flow (sampled from a 128×128 grid) with a splitting different from the Evaluation Section in the main body of the paper. As shown in Figure 13b, the number of attention heads has a crucial impact on the final performance. The model's performance drops significantly when using fewer heads. This highlights the importance of multi-head mechanism in the factorized attention. In addition, we observe that the final accuracy of our model benefits from an increased kernel dimension (as shown in Figure 13a).



(a) Ablation on the kernel dimension. (b) Ablation on the number of heads.

Figure 13: Ablation study on the key hyperparameters. Red color denotes the final choice of hyperparameter. Experiments are carried out on the validation fold.

Visualization of learned kernels We visualize the learned kernel as shown in Figure 14. Due to the presence of Rotary positional encoding [105], all kernels have a stationary pattern (the kernel value $\kappa(\xi_1, \xi_2)$ depends only on the relative distance between two points, e.g. L^2 distance: $||\xi_1 - \xi_2||_2$). The kernel matrices also exhibits symmetric pattern despite the non-symmetric nature of dot product QK^T and all kernel matrices are diagonal dominated.



(a) Layer 1, Head 1: kernel of y axis (b) Layer 1, Head 1: kernel of x axis (c) Layer 4, Head 1: kernel of y axis (d) Layer 4, Head 1: kernel of x axis

Figure 14: Visualization of normalized attention kernel .

Influence of random seed We investigate the influence of random seeds by training the model with three different seeds. As shown in Figure 15, all models converge to the similar level of loss with marginal difference.

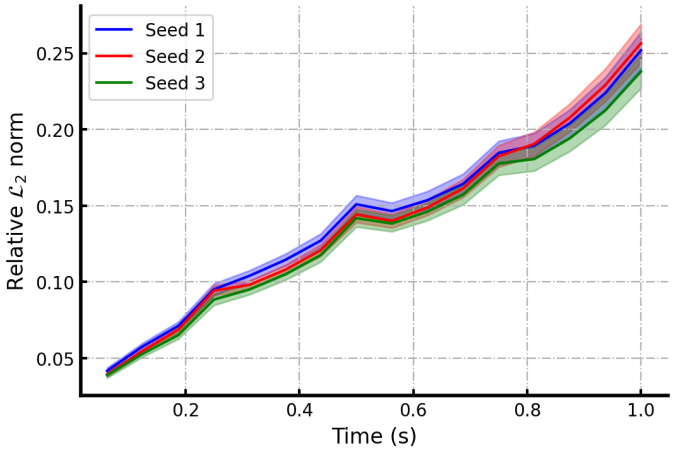


Figure 15: Averaged frame-wise loss trends. Each loss curve corresponds to a model initialized under a specific seed.

E Dataset details

In this section we provide the details for each dataset.

2D Kolmogorov flow The incompressible Navier-Stokes equation under vorticity form reads as,

$$\begin{aligned} \frac{\partial \omega(\mathbf{x},t)}{\partial t} + \mathbf{u}(\mathbf{x},t)\cdot\nabla\omega(\mathbf{x},t) &= \frac{1}{Re}\nabla^2\omega(\mathbf{x},t) + f(\mathbf{x}), && \mathbf{x}\in(0,2\pi)^2, t\in(0,T], \\ \nabla\cdot\mathbf{u}(\mathbf{x},t) &= 0, && \mathbf{x}\in(0,2\pi)^2, t\in[0,T], \\ \omega(\mathbf{x},0) &= \omega_0(\mathbf{x}), && \mathbf{x}\in(0,2\pi)^2, \end{aligned} \tag{26}$$

where ω denotes vorticity, $\mathbf{u}$ denotes velocity, *Re* denotes the Reynolds number, $\mathbf{x} = (x_1, x_2)$ denotes the spatial coordinates and $f(\cdot)$ is the forcing term that is set to $f(\mathbf{x}) = -n\cos(nx_2) - 0.1\omega(\mathbf{x})$. The equation is periodic in all spatial directions. Compared to the cases discussed in Li et al. [72], we set the forcing factor n to 8 [1, 19] and introduce dragging force term $0.1\omega(\mathbf{x})$ as described in Kochkov et al. [56]. The initial condition ω_0 is sampled from a prescribed Gaussian random field

same as Li et al. [72]. The dataset consists of 100 trajectories for training and 20 trajectories for testing, with the length of each trajectory being 10 seconds and 160 frames.

We modify the pseudo-spectral solver (under Apache License 2.0) from https://github.com/neuraloperator/physics_informed/blob/master/solver/kolmogorov_flow.py to generate the data. The referenced direct numerical simulation is carried out with a spatial resolution of 2048×2048 and a temporal resolution of $1e-4$.

3D isotropic turbulence The incompressible Navier-Stokes equation for this problem is given as:

$$\begin{aligned} \frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} + \mathbf{u}(\mathbf{x},t)\cdot\nabla\mathbf{u}(\mathbf{x},t) = \nu\nabla^2\mathbf{u}(\mathbf{x},t) - \frac{1}{\rho}\nabla p(\mathbf{x},t) + \mathbf{f}(\mathbf{x}), \quad & \mathbf{x}\in(0,2\pi)^3, t\in(0,T], \\ \nabla\cdot\mathbf{u}(\mathbf{x},t) = 0, \quad & \mathbf{x}\in(0,2\pi)^3, t\in[0,T], \\ \mathbf{u}(\mathbf{x},0) = \mathbf{u}_0(\mathbf{x}), \quad & \mathbf{x}\in(0,2\pi)^3, \end{aligned} \tag{27}$$

where $\mathbf{u}$ denotes velocity, p denotes the pressure, ν is the viscosity parameter, $\mathbf{x} = [x_1, x_2, x_3]$ denotes the spatial coordinates and $f(\cdot)$ is the forcing term. The equation is periodic in all three spatial dimensions. The initialization of $\mathbf{u}$ and the forcing settings follow Rogallo [98] and Lamorgese et al. [62] respectively, with Taylor Reynolds number set to 84 [62]. The dataset consists of 1000 trajectories for training and 100 trajectories for testing, with the length of each trajectory being 1 second and 20 frames.

We use the spectral Galerkin solver (under GNU GPL license 3.0) from https://github.com/spectralDNS. The referenced simulation is carried out with a spatial resolution of $60 \times 60 \times 60$ and a temporal resolution of $0.005s$.

3D smoke buoyancy The governing equations for the 3D smoke buoyancy problem are incompressible Navier-Stokes equation (similar as above) coupled with advection equation:

$$\begin{aligned} \frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} + \mathbf{u}(\mathbf{x},t)\cdot\nabla\mathbf{u}(\mathbf{x},t) = \nu\nabla^2\mathbf{u}(\mathbf{x},t) - \frac{1}{\rho}\nabla p(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t), \quad & \mathbf{x}\in(0,L)^3, t\in(0,T], \\ \frac{\partial d(\mathbf{x},t)}{\partial t} + \mathbf{u}(\mathbf{x},t)\cdot\nabla d(\mathbf{x},t) = 0, \quad & \mathbf{x}\in(0,L)^3, t\in(0,T], \\ \nabla\cdot\mathbf{u}(\mathbf{x},t) = 0, \quad & \mathbf{x}\in(0,L)^3, t\in[0,T], \\ \mathbf{u}(\mathbf{x},0) = 0, \quad \mathbf{d}(\mathbf{x},0) = d_0(\mathbf{x}), \quad & \mathbf{x}\in(0,L)^3, \end{aligned}$$

where $\mathbf{f}(\mathbf{x},t) = [0, 0, \eta d(\mathbf{x},t)]$, η is the buoyancy factor, the velocity field $\mathbf{u}$ has a Dirichlet boundary condition: $\mathbf{u}(\mathbf{x},\cdot) = 0, \forall \mathbf{x}\in\partial\Omega$, and the scalar density field for smoke has a Neumann boundary condition: $\nabla d(\mathbf{x},\cdot) = 0, \forall \mathbf{x}\in\partial\Omega$. The initial condition $d_0(\mathbf{x})$ is a random field [3] with scaling of Fourier coefficient set to 15.0, smoothness factor set to 4.0. The length of the rectangular domain L is set to 8. The dataset consists of 2000 trajectories for training and 200 trajectories for testing, with the length of each trajectory being 15 seconds and 20 frames.

We modify the 2D solver (under MIT license) from https://github.com/microsoft/pdearena to generate the data. The solver applies an advection-project scheme. The referenced simulation is carried out with a spatial resolution of $64 \times 64 \times 64$ and a temporal resolution of $0.75s$.

2D Darcy flow The equation for the 2D Darcy flow is defined as:

$$\begin{aligned} -\nabla\cdot(a(x)\nabla u(x)) = f(x), \quad & x\in(0,1)^2, \\ u_0(x) = 0, \quad & x\in\partial(0,1)^2, \end{aligned} \tag{28}$$

where $f(x)$ is the forcing function that is set to constant 1. The coefficient function $a(x)$ is sampled from Gaussian Random Field with zero Neumann boundary condition. The data is generated via second-order finite difference solver on a 421×421 resolution grid. We use the pre-generated dataset from Li et al. [68] (under MIT license). The dataset consists of 1000 samples for training and 100 samples for testing.

[3] Implemented with *phiflow*'s Noise class, see:https://tum-pbs.github.io/PhiFlow/phi/field/

F Results visualization

In this section, we provide exemplary visualization of the model's prediction. For 3D problems, the cross-section at the middle of the first axis is shown.

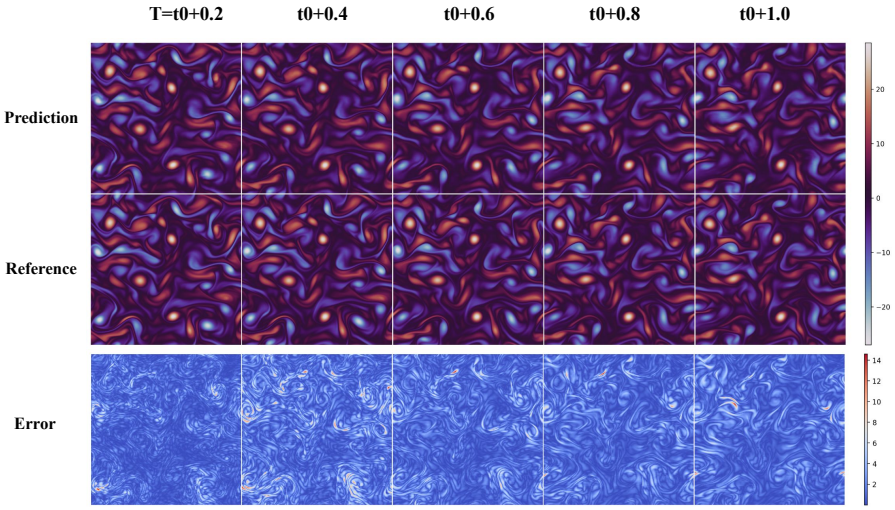


Figure 16: Sample 1 of 2D Kolmogorov flow.

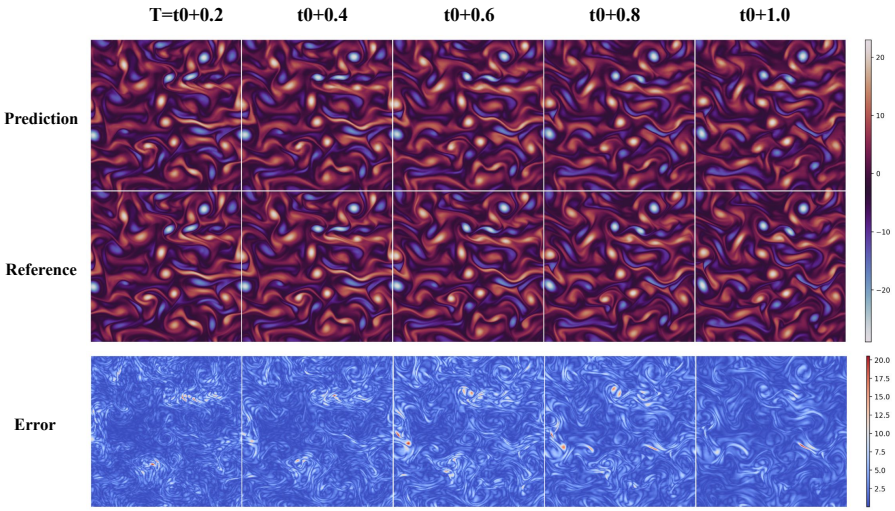


Figure 17: Sample 2 of 2D Kolmogorov flow.

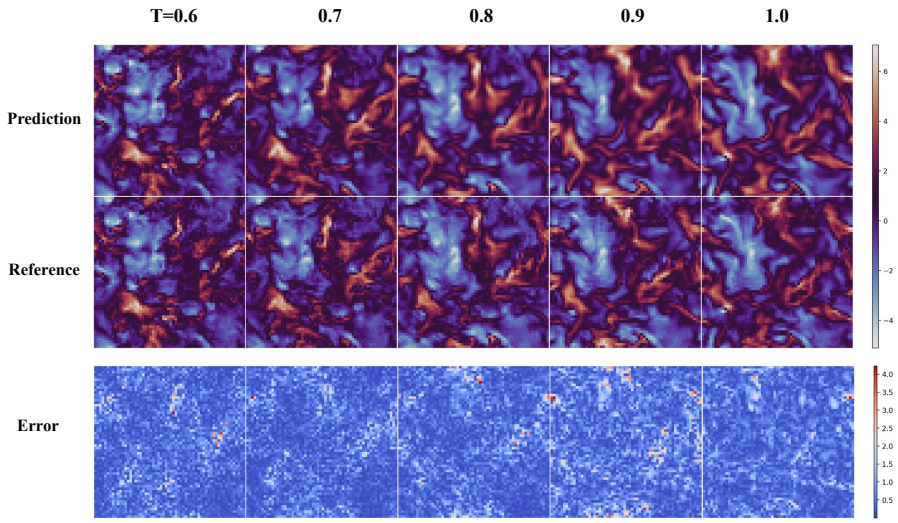


Figure 18: Pressure in 3D isotropic turbulence.

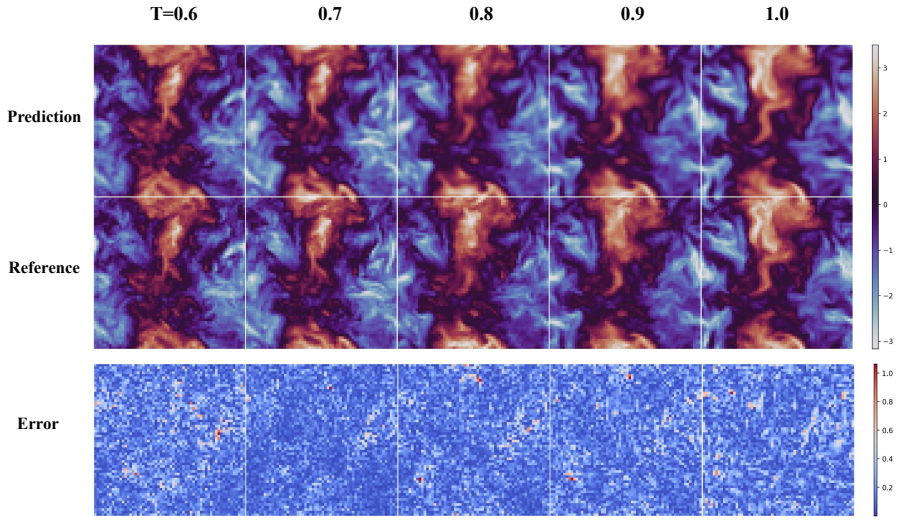


Figure 19: x-component of velocity in 3D isotropic turbulence.

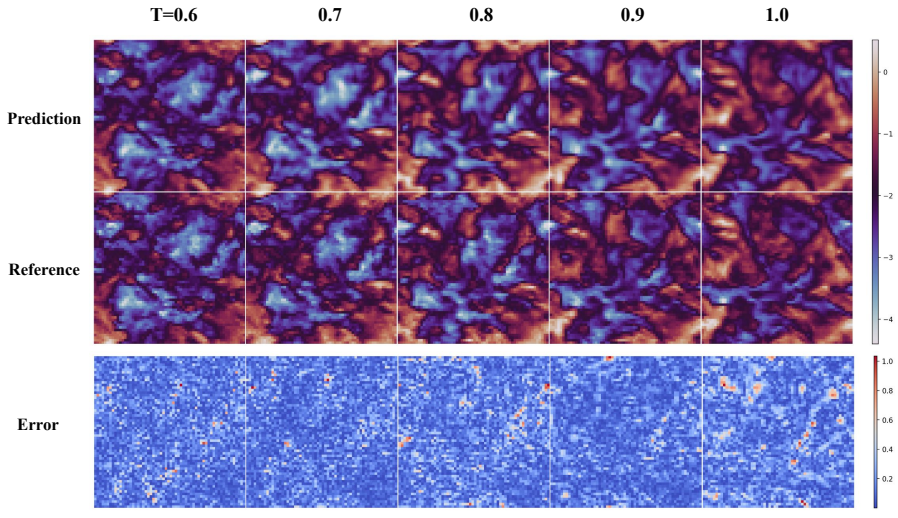


Figure 20: y-component of velocity in 3D isotropic turbulence.

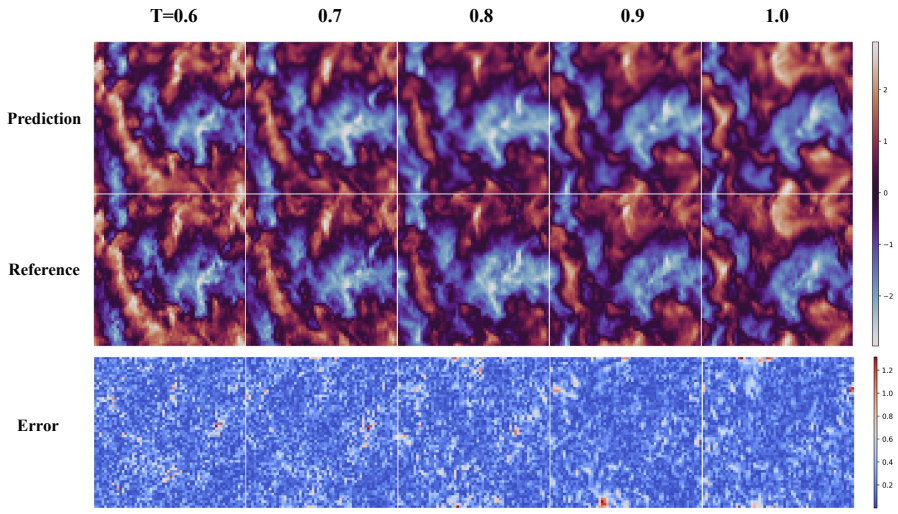


Figure 21: z-component of velocity in 3D isotropic turbulence.

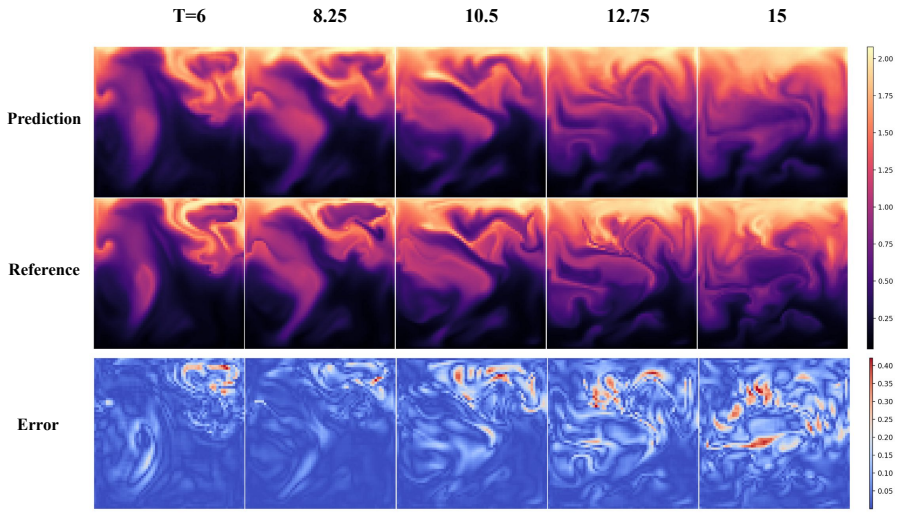


Figure 22: Smoke marker field in 3D smoke buoyancy.

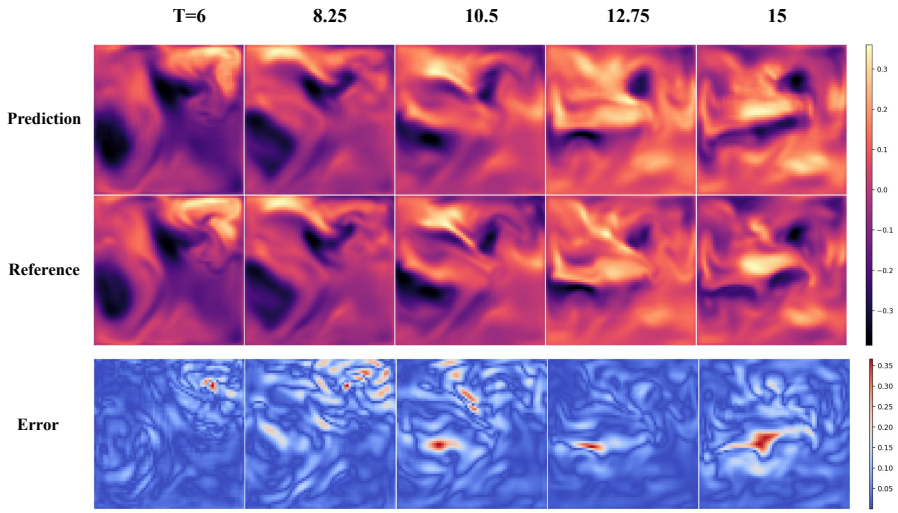


Figure 23: x-component of velocity in 3D smoke buoyancy.

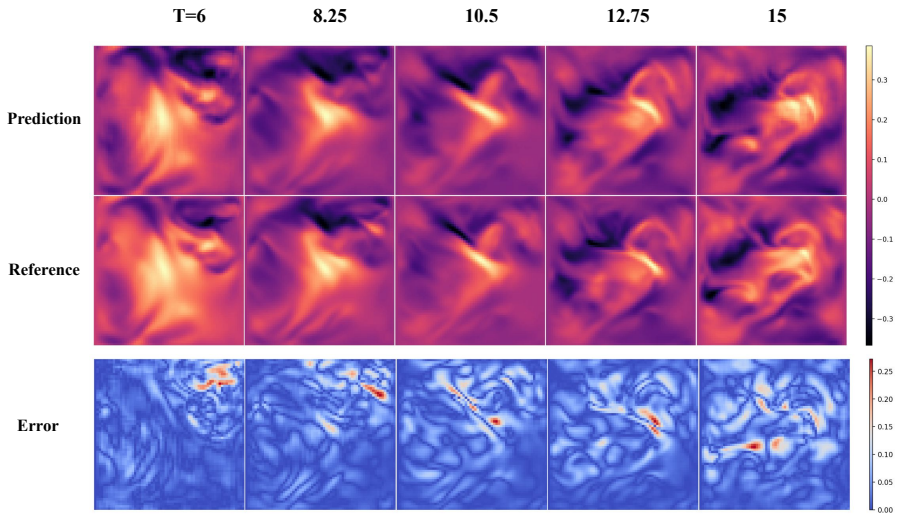


Figure 24: y-component of velocity in 3D smoke buoyancy.

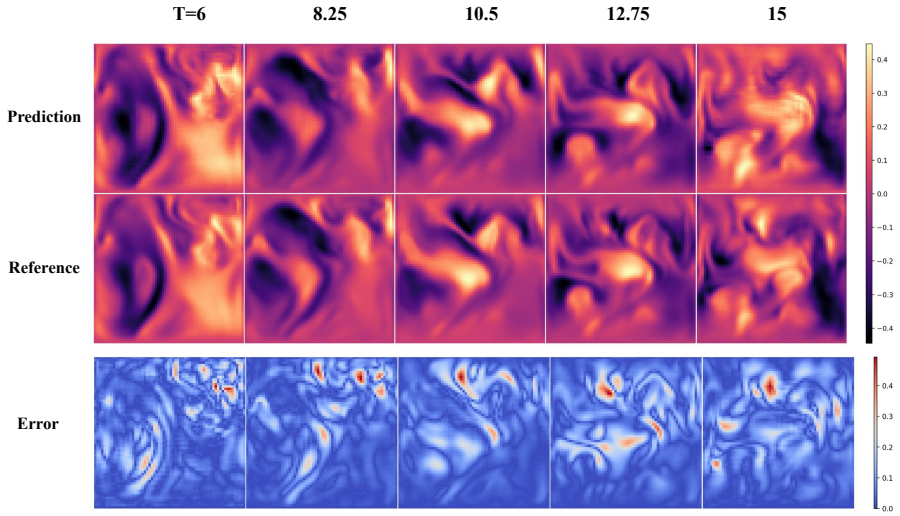


Figure 25: z-component of velocity in 3D smoke buoyancy.

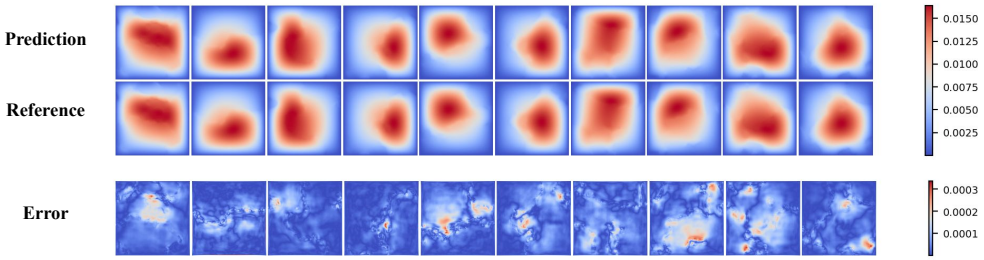


Figure 26: Flow field of 2D Darcy flow.

G Broader impact

The numerical simulation of PDEs is of extensive application in various fields, such as manufacturing, weather forecasting, and engineering design. Meanwhile, Transformer has shown promising performance on a wide range of data-driven applications including PDE modeling. Our work can help improve the stability and computational efficiency of the attention-based PDE surrogate models. Our experiments demonstrate that the proposed model serves as an efficient surrogate for numerical solvers, maintaining a balance between accuracy and efficiency, and thus pushing the Pareto front of accuracy-efficiency. However, as there exists a large variety of PDEs and each with very unique properties, there is no guarantee that one type of data-driven model can rule all. Additionally, just like most concurrent works on neural PDE solvers, the long-term stability of the proposed model cannot be guaranteed. Therefore it is important to acknowledge the limitations and potential risks associated with the application of neural PDE solvers.

Apart from enriching the existing architecture design choice of attention-based models, our work also has the potential to be combined with other neural PDE solvers design formulas (e.g., explicitly take into account the relationship between different output variables), or common neural network architectures (e.g., U-Net).

H Schematic of Axial Transformer and FactFormer

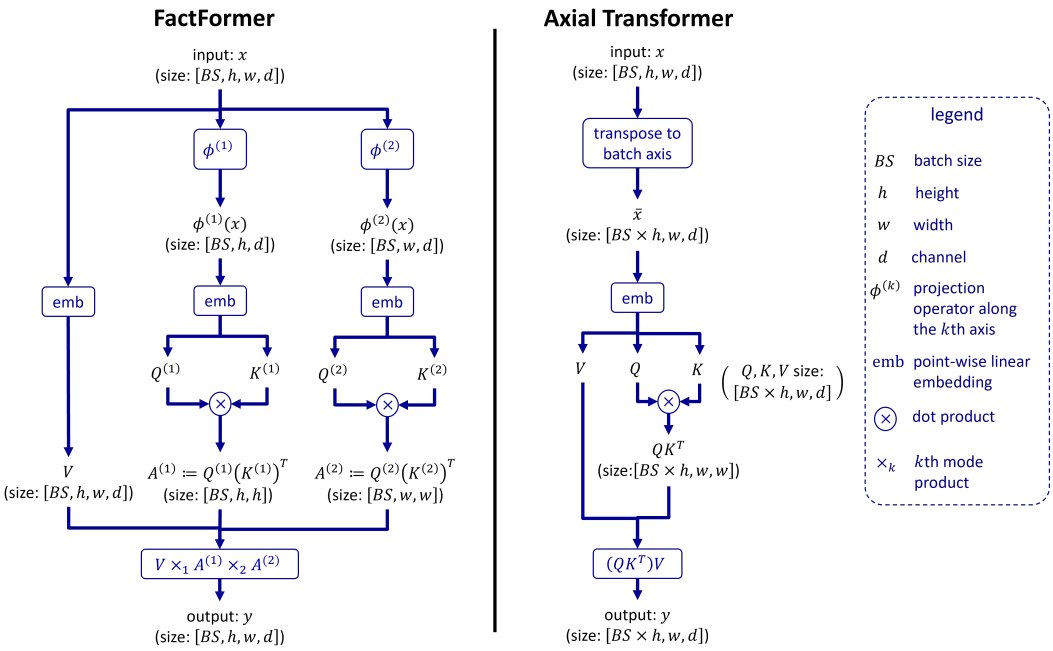


Figure 27: An illustrative example of FactFormer and Axial Transformer applying to 2D input data, with some details such as positional encoding, multi-head mechanism and softmax in Axial Transformer omitted for simplicity. For Axial Transformer, column-wise attention block is shown as an example.