Transformer for Partial Differential Equations' Operator Learning

Anonymous authors
Paper under double-blind review

Abstract

Data-driven learning of partial differential equations' solution operators has recently emerged as a promising paradigm for approximating the underlying solutions. The solution operators are usually parameterized by deep learning models that are built upon problem-specific inductive biases. An example is a convolutional or a graph neural network that exploits the local grid structure where functions' values are sampled. The attention mechanism, on the other hand, provides a flexible way to implicitly exploit the patterns within inputs, and furthermore, relationship between arbitrary query locations and inputs. In this work, we present an attention-based framework for data-driven operator learning, which we term Operator Transformer (OFormer). Our framework is built upon self-attention, cross-attention, and a set of point-wise multilayer perceptrons (MLPs), and thus it makes few assumptions on the sampling pattern of the input function or query locations. We show that the proposed framework is competitive on standard PDE benchmark problems and can flexibly be adapted to different types of grids. [1]

1 Introduction

Many of the phenomena in different fields of science and engineering are modeled by Partial Differential Equations (PDEs). Formulating and solving the governing PDEs help us understand, predict, and control complex systems in physics, engineering, etc. Various numerical schemes approach the problem of solving the PDEs via discretizing the solution space and solving a finite-dimensional problem. However, solving complex spatiotemporal PDEs demands a high-resolution discretization in both time and space which is computationally expensive.

A group of data-driven PDE solvers aim to directly learn the solution from the data observations without any prior knowledge of the underlying PDE. These methods usually rely on a supervised deep learning architecture which imposes an inductive bias that can suit the problem in hand. The use of convolutional layers for structured grids Khoo et al. (2021); Zhu & Zabaras (2018); Bhatnagar et al. (2019); Pant et al. (2021), and graph layers for learning unstructured local relations in a system de Avila Belbute-Peres et al. (2020); Iakovlev et al. (2021); Li & Farimani (2022); Li et al. (2022); Ogoke et al. (2021) are just two instances of these models. Learning such function mappings, however, is restricted to the specific resolution and the geometry observed in the training phase.

The pioneering work DeepONet (Lu et al., 2021a) introduces a new data-driven modeling paradigm that aims to learn the solution operator of PDEs and provides a practical realization of the general universal nonlinear operator approximation theorem (Chen & Chen, 1995). Concurrent work Graph Neural Operator (Anandkumar et al., 2020) proposes another operator learning architecture that leverages learnable iterative kernel integration. The promising features of data-driven operator learning, in particular the generalization capability within a family of PDE and the potential adaptivity to different discretizations, have given rise to a wide array of recent research works (Li et al., 2020b; 2021a; Wen et al., 2022; Rahman et al., 2022; Cao, 2021; Kissas et al., 2022; Lu et al., 2022; Cai et al., 2021; Bhattacharya et al., 2021; Nelsen & Stuart, 2021; Gupta et al., 2021). The training of data-driven operator can also be augmented with physics prior to regularize the

[1] Code is available at: `https://anonymous.4open.science/r/Operator-Transformer-7481`.

search space of optimal model, bridging the physics-informed machine learning (Raissi et al., 2019; Sun et al., 2020) with data-driven operator learning (Wang et al., 2021a; Li et al., 2021b; Wang et al., 2021b).

In practice, the data-driven operators are trained based on the finite-dimensional approximation of the input and output functions, where functions are sampled on some discretization grids. Many of the learned operators, despite showing promising performance on solving PDEs, are limited to have the same discretizations for the input and output Anandkumar et al. (2020); Li et al. (2020b), fixed input grids Lu et al. (2021a) or to have equi-spaced grid points Li et al. (2021a); Gupta et al. (2021); Cao (2021). The attention mechanism (Vaswani et al., 2017) has the potential to address this issue by providing a flexible learning framework, where we can have unstructured grids and different discretizations for input and output points, that potentially allows querying output function at arbitrary locations. In this work, we introduce an end-to-end operator learning framework building upon linear attention mechanism (Cao, 2021) and simple point-wise MLPs, which makes few assumptions of the discretization or geometrical constraints on either the input points or output points. Concretely, we design a cross-attention module that decouples the input and output grids and a latent time-marching architecture that enables building scalable and robust models for time-dependent PDEs. Combining these modules with an appropriate way to encode the positional information, we propose a flexible framework with competitive learning capacity.

2 Related works

Learned PDE Solvers In the past decade, there have been a number of studies on applying deep learning architectures for solving PDEs Lu et al. (2021b); Karniadakis et al. (2021); Brunton et al. (2020). Physics-informed methods Raissi et al. (2019); Sun et al. (2020); Sirignano & Spiliopoulos (2018); E & Yu (2018) exploit the PDE supervision to approximate the solution (usually parameterized by a neural network) in an aggregated space-time domain. They are accurate and generally mesh-agnostic, yet requires retraining across different instances of a PDE (e.g. change of coefficient). Without knowledge of the PDE, the solutions can also be learned purely from the data. A common strategy for such task is to first spatially encode the data and then use various schemes to evolve in time. Convolutional layers Stachenfeld et al. (2022); Wang et al. (2020); Bhatnagar et al. (2019); Kochkov et al. (2021), kernels Saha & Mukhopadhyay (2021), and graph-based layers de Avila Belbute-Peres et al. (2020); Iakovlev et al. (2021); Li & Farimani (2022); Li et al. (2022) are some of the common architectures for learning the spatial relations in a PDE. In this study, we show that transformers can be pliable yet effective spatial encoders.

Time-marching in the latent space Learning the temporal component of the spatio-temporal PDEs can be challenging. A common and straightforward approach is to follow an Encoder-Process-Decoder (EPD) scheme to map the input solution at time t to the solution at next time step Brandstetter et al. (2022); Stachenfeld et al. (2022); Pfaff et al. (2020); Sanchez-Gonzalez et al. (2020); Pant et al. (2021); Hsieh et al. (2019). An alternative approach which can significantly reduce the computational complexity and memory usage is to propagate the dynamics in the latent space Lee & Carlberg (2021); Wiewel et al. (2019). Therefore, once the encoding from observation space to the latent space is done, the system can evolve in time using Recurrent Neural Networks (RNNs) such as LSTM Wiewel et al. (2019) or even linear propagators based on the assumptions of learning Koopman operator Morton et al. (2018); Li et al. (2020a); Lusch et al. (2018); Takeishi et al. (2017); Pan & Duraisamy (2020). In this work, we design a latent time-marching architecture that brings about scalability of the model's attention encoder and allows the model to unroll for the future time steps during training.

Neural Operators Supported by theoretical universal approximation guarantees Chen & Chen (1995), DeepONet Lu et al. (2021a) introduces an general architecture for operator learning, which can also be integrated with prior knowledge of the system Wang et al. (2021a;b). In another group of works, the infinite-dimensional solution operators are approximated by iterative learnable integral kernels Anandkumar et al. (2020). Such kernel can be parameterized by message-passing (Gilmer et al., 2017) neural networks Anandkumar et al. (2020); Li et al. (2020b) or convolution in the Fourier domain Li et al. (2021a). Kovachki et al. (2021) further provide a theoretical analysis of the approximation capacity of kernel-integral-based learning framework. Fourier Neural Operator (FNO) Li et al. (2021a) and its variants Li et al. (2021b); Guibas et al. (2022); Tran et al. (2021) have shown promising results in various applications Pathak et al.

(2022); Wen et al. (2022). These integral kernels can also be learned in the multiwavelet domain Gupta et al. (2021); Tripura & Chakraborty (2022). Furthermore, other than Fourier bases or multiwavelet bases, the bases needed for approximating the solution operator can be learned and composed via attention Cao (2021).

Transformer for physical system After the groundbreaking success in natural language processing Vaswani et al. (2017); Devlin et al. (2018), attention has also been demonstrated as a promising tool for various other machine learning tasks Dosovitskiy et al. (2021); Tunyasuvunakool et al. (2021); Veličković et al. (2017). Following the success in these fields, several previous works (Geneva & Zabaras, 2022; Han et al., 2022; Shao et al., 2022; Cao, 2021; Kissas et al., 2022) have explored using attention to model and simulate physical system, which can generally be divided into two orthogonal lines. In the first group, attention layers are used to capture the structures and patterns lie in the PDEs' spatial domain Cao (2021); Kissas et al. (2022); Shao et al. (2022). On the contrary, in the second group, the attention is used to model the temporal evolution of the system while the spatial encoding is done by other mechanisms like CNNs or GNNs Geneva & Zabaras (2022); Han et al. (2022). Our model lies in the former group where attention-based layers are used for encoding the spatial information of the input and query points, while the time marching is performed in the latent space using recurrent MLPs.

3 Method

3.1 Attention mechanism

Standard attention The standard attention (Vaswani et al., 2017; Bahdanau et al., 2014; Graves et al., 2014; Luong et al., 2015) is an operation defined upon three sets of vectors, namely the query vectors $\{\mathbf{q}_j\}$, key vectors $\{\mathbf{k}_j\}$, and value vectors $\{\mathbf{v}_j\}$. Concretely, for each query $\mathbf{q}_i$, the corresponding output $\mathbf{z}_i$ is calculated as a weighted sum (assuming number of vectors in each set is n):

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij}\mathbf{v}_j, \quad \alpha_{ij} = \frac{\exp\left(h(\mathbf{q}_i, \mathbf{k}_j)\right)}{\sum_{s=1}^{n} \exp\left(h(\mathbf{q}_i, \mathbf{k}_s)\right)}. \tag{1}$$

The most common choice for the weight function $h(\cdot)$ is the scaled dot-product (Vaswani et al., 2017): $h(\mathbf{q}_i, \mathbf{k}_j) = (\mathbf{q}_i \cdot \mathbf{k}_j)/\sqrt{d}$, with d being the dimension of vectors $\mathbf{q}_i$ and $\mathbf{k}_j$. The scaled dot-product can be written in matrix representation as: $\mathbf{Z} = \text{softmax}\left(\mathbf{Q}\mathbf{K}^T/\sqrt{d}\right)\mathbf{V}$, where $\mathbf{z}_i, \mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$ are the i-th row vectors of matrices $\mathbf{Z}, \mathbf{Q}, \mathbf{K}, \mathbf{V}$ respectively. Due to the presence of the softmax, the multiplication $\mathbf{Q}\mathbf{K}^T$ must be evaluated explicitly, resulting in overall complexity of $O(n^2 d)$, which is prohibitively expensive when applying to long sequences.

Attention without softmax In natural language processing and computer vision tasks, the i-th row vector of query/key/value matrices, $\mathbf{q}_i/\mathbf{k}_i/\mathbf{v}_i$, are usually viewed as the latent embedding of a token (e.g. a word or an image patch). Other than the "row-wise" interpretation of attention, Cao (2021) proposes that each column in the query/key/value matrices, can potentially be interpreted as the evaluation of a learned basis function at each point. For instance, $\mathbf{V}_{ij}$ (also the i-th element of the j-th column vector: $(\mathbf{v}^j)_i$) can be viewed as the evaluation of the j-th basis function on the i-th grid point x_i, i.e. $\mathbf{V}_{ij} = v_j(x_i)$, and similarly for the matrices $\mathbf{Q}, \mathbf{K}$ (with each column represent the sampling of basis function $q_j(\cdot), k_j(\cdot)$). Facilitated by the learnable bases interpretation, Cao (2021) proposes two types of attention that can be viewed as the numerical quadrature of different forms of integrals:

$$\text{Fourier type:} \quad (\mathbf{z}_i)_j = \frac{1}{n}\sum_{s=1}^{n} (\mathbf{q}_i \cdot \mathbf{k}_s)(\mathbf{v}^j)_s \approx \int_{\Omega} \kappa\left(x_i, \xi\right) v_j(\xi) d\xi, \tag{2}$$

$$\text{Galerkin type:} \quad (\mathbf{z}^j)_i = \sum_{l=1}^{d} \frac{(\mathbf{k}^l \cdot \mathbf{v}^j)}{n} (\mathbf{q}^l)_i \approx \sum_{l=1}^{d} \left(\int_{\Omega} (k_l(\xi) v_j(\xi)) d\xi\right) q_l(x_i), \tag{3}$$

where Ω denotes the spatial domain that is discretized by n points $\{x_i\}_{i=1}^n$, and the dot product $\mathbf{q}_i \cdot \mathbf{k}_s$ in equation 2 approximates the learnable kernel function $\kappa\left(x_i, \xi\right)$. The approximation capability of Fourier type

attention and Galerkin type attention are studied theoretically at (Choromanski et al., 2020) and (Cao, 2021) respectively. Inspired by the Gram-Schmidt process, Cao (2021) heuristically chooses layer normalization (Ba et al., 2016) to normalize $\mathbf{q}_i, \mathbf{k}_s$ in equation 2 and $\mathbf{k}^l, \mathbf{v}^j$ in equation 3. But if we further exploit the basis function property of these column vectors, it is also reasonable to impose non-learnable instance normalization (Ulyanov et al., 2016) on each column vector such that their L_2 norm equals to one, i.e. $\left|\left|\mathbf{v}^j\right|\right|_2 = 1$. In practice, we find this slightly improves the model's performance, and thus we adopt instance normalization throughout the model. Equipped with the normalization schemes, equation 2 and equation 3 in matrix multiplication form write as:

$$\text{Fourier type:} \quad \mathbf{Z} = \frac{1}{n}\widehat{\mathbf{Q}}\widehat{\mathbf{K}}^T\mathbf{V}; \quad \text{Galerkin type:} \quad \mathbf{Z} = \frac{1}{n}\mathbf{Q}(\widehat{\mathbf{K}}^T\widehat{\mathbf{V}}), \tag{4}$$

where $\widehat{\cdot}$ denotes a column-wise normalized (via instance normalization) matrix. The above integral-based attention mechanisms serve as simple but powerful building blocks for PDE operator learning model. The softmax-free property allows both Fourier type and Galerkin type to be computed efficiently in the form (omitting the normalization scheme): $\mathbf{Z} = \mathbf{Q}(\mathbf{K}^T\mathbf{V})/n$, since matrix multiplication is associative.

Cross-attention While the aforementioned attention mechanisms are flexible with respect to the discretization of input domain, the fact that $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are essentially different linear projections of the same input feature embedding $\mathbf{H}$ makes the input $\mathbf{H}$ and output $\mathbf{Z} = \mathbf{Q}(\mathbf{K}^T\mathbf{V})/n$ locked to the same fixed grid $\{x_i\}_{i=1}^n$. To decouple the output domain from the input domain and allow for arbitrary query locations, we extend the above attention mechanisms from self-attention to cross-attention, where the query matrix $\mathbf{Q}$ is encoded from a different input. To be more specific, the i-th row vector $\mathbf{q}_i$ in the query matrix $\mathbf{Q}$ is the latent encoding of query point y_i, where $\{y_i\}_{i=1}^m$ denotes the discretization of the output domain and is not necessarily same as the input grid points $\{x_i\}_{i=1}^n$. Hence, leveraging the cross-attention mechanism, we can query at arbitrary locations which are independent of the input grid points. If we adopt the learned bases interpretation from previous section, the cross-attention mechanism can also be viewed as a weighted sum of three sets of basis functions: $\{k_l(\cdot), v_l(\cdot), q_l(\cdot)\}_{l=1}^d$ with $k_l(\cdot), v_l(\cdot)$ defined on the input discretization grid $D(x) : \{x_i\}_{i=1}^n$ and $q_l(\cdot)$ defined on the query discretization grid $D(y) : \{y_i\}_{i=1}^m$, which serves as the finite-dimensional approximation space's basis for the final target function:

$$z_j(y_{i'}) = \sum_{l=1}^{d} \frac{\sum_{i=1}^n k_l(x_i)v_j(x_i)}{n} q_l(y_{i'}). \tag{5}$$

If we drop the point-wise feed forward network (FFN) after the attention layer, which is a MLP with two layers (omitting the bias term): $\mathbf{x} \leftarrow \sigma\left(\mathbf{x}\mathbf{W}^{(1)}\right)\mathbf{W}^{(2)}$, where $\mathbf{W}^{(1)}, \mathbf{W}^{(2)} \in \mathbb{R}^{d\times d}$ are learnable weight matrices and $\sigma(\cdot)$ is a non-linear activation function, and then directly make $z_j(y_{i'})$ as output, the cross attention in the equation 5 is a special case of the universal operator approximator proposed in DeepONet (Lu et al., 2021a) with $\sum_{i=1}^n k_l(x_i)v_j(x_i)/n$ as the branch net and $q_l(y_{i'})$ as the trunk net. But in practice, we find that appending FFN after attention improves the model's performance and a propagating MLP is necessary for time-dependent PDEs (see Figure 2). More broadly speaking, the query matrix $\mathbf{Q}$, whose column space is spanned by the learnable query basis functions $\{q_l(\cdot)\}_{l=1}^d$, loosely resembles the induced points in Set Transformer (Lee et al., 2018) and the latent array in Perceiver (Jaegle et al., 2021).

3.2 Model architecture

Encoder The encoder comprises three components (Figure 1), an input encoder $\phi^{\mathcal{X}}(\cdot)$ that takes the input function's sampling $a(x_i)$ and coordinates of sampling positions $\{x_i\}_{i=1}^n$ as input features, a query encoder $\phi^{\mathcal{Y}}(\cdot)$ that takes the coordinates of query locations $\{y_i\}_{i=1}^m$ as input features, and a cross-attention module to pass system information from input locations to the query locations.

The input encoder first uses a point-wise MLP that is shared across all locations to lift input features into high-dimensional encodings $\mathbf{f}^{(0)}$ and the encodings are then fed into stacked self-attention blocks. The update protocol inside each self-attention block is similar to the standard Transformer (Vaswani et al., 2017):

$$\mathbf{f}^{(l')} = \text{LayerNorm}\left(\mathbf{f}^{(l)} + \text{Attn}(\mathbf{f}^{(l)})\right), \quad \mathbf{f}^{(l+1)} = \text{LayerNorm}\left(\mathbf{f}^{(l')} + \text{FFN}(\mathbf{f}^{(l')})\right), \tag{6}$$

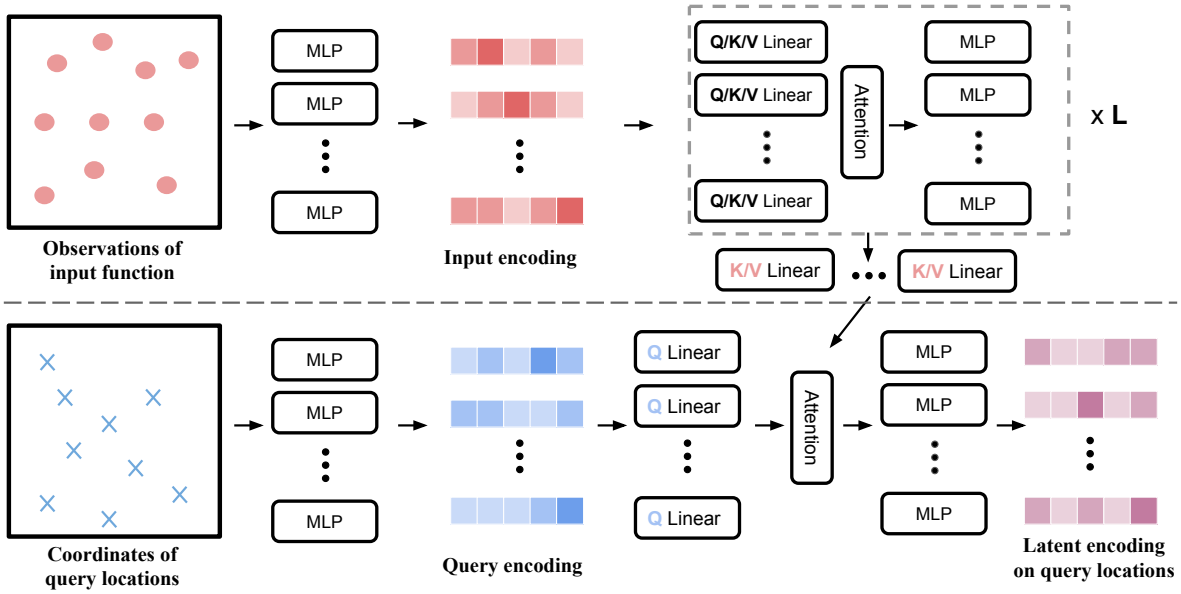


Figure 1: Attention-based encoder architecture. **Top row**: Input encoder that encodes input function information based on locations where input function is sampled and the sampled function value. **Bottom row**: Query encoder that encodes the coordinates of query locations and uses encoded coordinates to aggregate information from input encoding via cross-attention.

where $\text{Attn}(\cdot)$ denotes the linear attention mechanisms discussed in the previous sub-section, $\text{LayerNorm}(\cdot)$ denotes the layer normalization (Ba et al., 2016). When input and output data are not normalized, we drop the LayerNorm to allow the scaling to propagate through layers.

The query encoder consists of a shared point-wise MLP, whose first layer is a random Fourier projection layer (Tancik et al., 2020). The random Fourier projection $\gamma(\cdot)$ using Gaussian mapping is defined as:

$$\gamma(\mathbf{Y}) = [\cos(2\pi\mathbf{Y}\mathbf{B}), \sin(2\pi\mathbf{Y}\mathbf{B})], \tag{7}$$

where $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n]^T$, $\mathbf{y}_i$ is the Cartesian coordinates of the i-th query point, $\mathbf{B} \in \mathbb{R}^{d_1 \times d_2}$ (with d_1 the dimension of input coordinates and d_2 the output dimension) is a matrix with elements sampled from Gaussian distribution $\mathcal{N}(0, \sigma^2)$ with predefined σ. The random Fourier projection $\gamma(\cdot)$ allows the coordinate-based neural network to learn high-frequency patterns (Mildenhall et al., 2020), which has also been introduced into physics-informed machine learning in Wang et al. (2021c).

At the cross-attention module, by attending the latent encoding of the last (L-th) attention layer's output $\mathbf{f}^{(L)}$ of input function with the learned encoding $\mathbf{z}^{(0)}$ of query locations, we pass the information of input function to the query points. The cross-attention process is defined as:

$$\mathbf{z}' = \mathbf{z}' + \text{Cross-Attn}(\mathbf{z}^{(0)}, \mathbf{f}^{(L)}), \quad \mathbf{z} = \mathbf{z}' + \text{FFN}(\mathbf{z}'). \tag{8}$$

Compared to equation 6, we drop the LayerNorm at each sub-layer to preserve the scale (variance) of $\mathbf{z}^{(0)}$.

Latent propagator for time-dependent system A direct way to approach the time-dependent problem would be by extending the degrees of freedom and introducing temporal coordinates for each point as Raissi et al. (2019), which requires the model to learn the entire space-time solution at once. As analyzed in Krishnapriyan et al. (2021), for certain problems such as reaction-diffusion equation, this could lead to poor performance. In addition, this usually requires the model to have larger number of parameters in order to model the extra degree of freedom (Li et al., 2021a; Gupta et al., 2021).

Another direction would be using a time-marching scheme to autoregressively solve the entire state space with a fixed time discretization (resembles the *Method of lines* (Schiesser, 1991)), which is adopted by many neural-network based solvers as it is easier to train. A common architecture of learned time-marching solvers

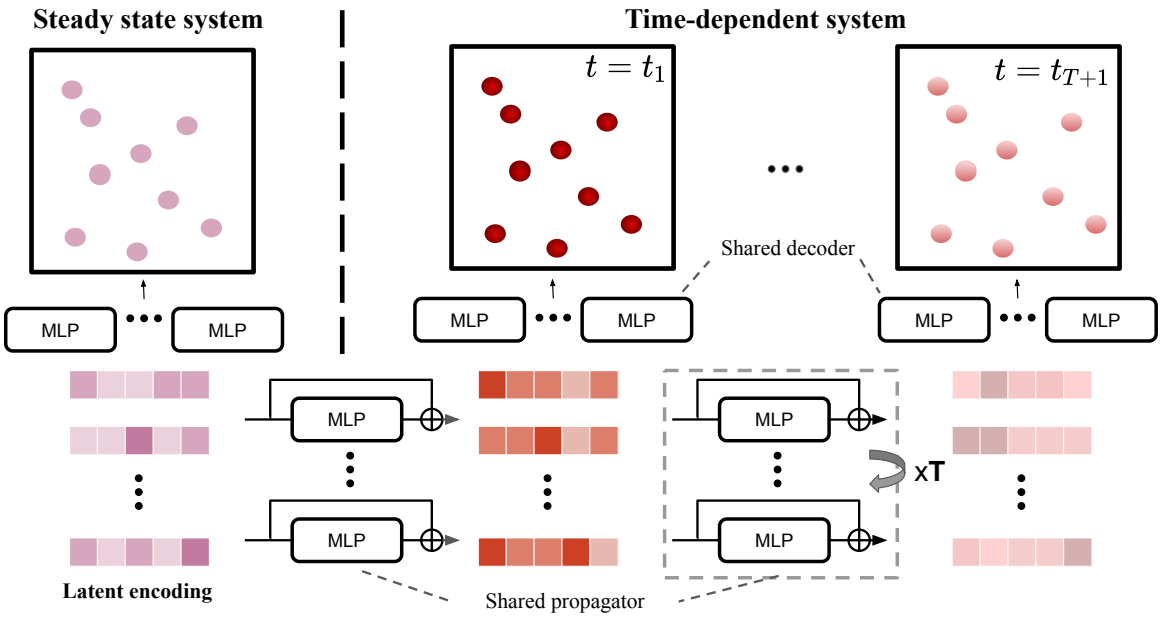


Figure 2: Decoder and propagator architecture. Given the latent encoding, for steady state system, a point-wise MLP is used to map the latent encoding $\mathbf{z}$ to the output function value. For time-dependent system, a point-wise MLP with skip connection is used to march states temporally in the latent space.

is the *Encoder-Processor-Decoder* (EPD) architecture (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020; Brandstetter et al., 2022; Stachenfeld et al., 2022), which takes state variable u_t as input, project it into latent space, then processes the encoding with neural networks and finally decodes it back to the observable space: $\hat{u}^{t+\Delta t}$. Such model is trained by minimizing the per-step prediction error, with input u^t and target $u^{t+\Delta t}$ sampled from ground truth data. In fact, if we unroll the learned per-step solver $f_{\text{NN}}(\cdot)$ to predict the full trajectory ($\hat{u}^1 = f_{\text{NN}}(u^0), \hat{u}^2 = f_{\text{NN}}(\hat{u}^1), \ldots$), the unrolled solver can be viewed as a recurrent neural network (RNN) with no hidden states, and thus the standard per-step training pipeline is essentially a recast of the *Teacher Forcing* (Williams & Zipser, 1989). Therefore, the learned per-step solvers face similar problems as RNNs trained with Teacher Forcing, the accumulated prediction error leads to the shift of the input data distribution and thus makes model perform poorly. Such data distribution shift can be alleviated by injecting noise during training (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2020; Stachenfeld et al., 2022). Alternatively, Brandstetter et al. (2022) propose to unroll the solver for several steps and let the gradient flow through the last step to train the model on the shifted data. For operator learning problem where time horizon is fixed, Li et al. (2021a) push this further by fully unrolling the trajectory during training. However, fully unrolled training has a memory complexity of $O(tn)$ (with t the length of time horizon and n the number of model parameters), which can be prohibitively expensive if the model is large.

Here we approach the space-time modeling as a sequence-to-sequence (Sutskever et al., 2014) learning task and propose a recurrent architecture to propagate the state in the time dimension. Different from the standard EPD model, we forward the dynamics in the latent space instead of the observable space, and thus encoder only needs to be called once throughout the whole process, which reduces the memory usage and allows scalable fully unrolled training. Given the latent encoding $\mathbf{z}$ from equation 8, we set it as the initial condition $\mathbf{z}^0$ and recurrently forward the latent state with a propagator $N(\cdot)$ that predicts the residuals (He et al., 2015) between each step: $\mathbf{z}^{t+1} = N(\mathbf{z}^t) + \mathbf{z}^t$. There are many suitable choices for parametrizing the propagator (e.g. self-attention, message-passing layer), but in practice, we found a point-wise MLP shared across all the query locations and all the time steps suffice, which indicates that the our model can effectively approximate the original time-dependent PDE with a discrete latent ODE.

After reaching target time step in the latent space, we use another point-wise MLP to decode the latent encoding $\mathbf{z}^t$ back to the output function $u(\cdot, t)$.

Relative positional encoding The attention mechanism is generally position-agnostic if no positional information is present in the input features. Galerkin/Fourier Transformer (Cao, 2021) approaches this problem by concatenating the coordinates into input features and latent embedding inside every attention head, and adds spectral convolutional decoder (Li et al., 2021a) on top of the attention layers. Instead of using convolutional architecture which is usually tied to the structured grid, we use Rotary Position Embedding (RoPE) (Su et al., 2021) to encodes the positional information, which is originally proposed for encoding word's relative position.

Given the 1D Cartesian coordinate x_i of a point and its d-dimensional embedding vector $\mathbf{q}_i$, the RoPE $\psi(\cdot)$ is defined as:

$$\psi\left(\mathbf{q}_i, x_i\right) = \mathbf{\Theta}\left(x_i\right)\mathbf{q}_i, \quad \mathbf{\Theta}\left(x_i\right) = \mathrm{Diag}(\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_{d/2}), \tag{9}$$

$$\text{where:} \quad \mathbf{R}_l = \begin{bmatrix} \cos\left(\lambda x_i \theta_l\right) & -\sin\left(\lambda x_i \theta_l\right) \\ \sin\left(\lambda x_i \theta_l\right) & \cos\left(\lambda x_i \theta_l\right) \end{bmatrix}, \tag{10}$$

where $\mathrm{Diag}(\cdot)$ denotes stacking sub-matrices along the diagonal, λ is wavelength of the spatial domain (e.g. $\lambda = 2048$ for a spatial domain discretized by 2048 equi-distant points), θ_l is set to $10000^{-2(l-1)/d}, l \in \{1, 2, \ldots, d/2\}$ following (Vaswani et al., 2017; Su et al., 2021). Leveraging the property of the rotation matrix, the above positional encoding injects the relative position information when attending a query vector $\mathbf{q}_i$ with a key vector $\mathbf{k}_j$: $\psi\left(\mathbf{q}_i, x_i\right) \cdot \psi\left(\mathbf{k}_j, x_j\right) = (\mathbf{\Theta}(x_i)\mathbf{q}_i) \cdot (\mathbf{\Theta}(x_j)\mathbf{k}_j) = \mathbf{\Theta}(x_i - x_j)(\mathbf{q}_i \cdot \mathbf{k}_j)$. RoPE can be extended to arbitrary degrees of freedom as proposed in (Biderman et al., 2021) by splitting the query and key vectors. Suppose the spatial domain is 2D, the attention of query $\mathbf{q}_i$ with coordinate $\mathbf{x}_i = (\alpha_i, \beta_i)$ and key $\mathbf{k}_j$ with $\mathbf{x}_j = (\alpha_j, \beta_j)$ write as:

$$\begin{aligned} &\psi\left(\mathbf{q}_i, \mathbf{x}_i\right) \cdot \psi\left(\mathbf{k}_j, \mathbf{x}_j\right) = \\ &\psi_1\left((\mathbf{q}_i)_{:d/2}, \alpha_i\right) \cdot \psi_1\left((\mathbf{k}_j)_{:d/2}, \alpha_j\right) + \psi_2\left((\mathbf{q}_i)_{d/2:}, \beta_i\right) \cdot \psi_2\left((\mathbf{k}_j)_{d/2:}, \beta_j\right), \end{aligned} \tag{11}$$

where $(\mathbf{q}_i)_{:d/2}$ denotes the first $d/2$ dimensions of vector $\mathbf{q}_i$ and $(\mathbf{q}_i)_{d/2:}$ denotes the rest $d/2$ dimensions, similarly for $\mathbf{k}_j$, and $\psi_1(\cdot), \psi_2(\cdot)$ is the 1D RoPE as defined in equation 9.

Training setting The general training framework of this work is similar to the previous data-driven operator learning models (Gupta et al., 2021; Li et al., 2021a; Cao, 2021; Anandkumar et al., 2020; Lu et al., 2021a). Given the input function $a \in \mathcal{A}$, and target function $u \in \mathcal{U}$, our goal is to learn an operator $\mathcal{G}_\theta : \mathcal{A} \mapsto \mathcal{U}$. The model parameter θ is optimized by minimizing the empirical loss $\mathbb{E}_{a\sim\mu}\left[\mathcal{L}(\mathcal{G}_\theta(a), u)\right]$, where μ is a measure supported on $\mathcal{A}$, and $\mathcal{L}(\cdot)$ is the appropriate loss function. In practice, we evaluate this loss numerically based on the sampling of $\{a^{(j)}_{D(x)}, u^{(j)}_{D(y)}\}_{j=1}^N$ on discrete grid points $D(x) : \{x_i\}_{i=1}^n, D(y) : \{y_i\}_{i=1}^m$, and choose $\mathcal{L}(\cdot)$ as relative $\mathcal{L}_2$ norm: $\frac{1}{B}\sum_{j=1}^{B} \frac{||\hat{u}^{(j)} - u^{(j)}||_2}{||u^{(j)}||_2}$, given B samples with $\hat{u}^{(j)}$ being the model prediction. Different from several previous works (Li et al., 2021a; 2020b; Cao, 2021; Gupta et al., 2021), $D(x), D(y)$ does not need to be the same discretization of the PDE spatial domain Ω and can vary across different training samples $\{a^{(j)}_{D(x)}, u^{(j)}_{D(y)}\}$.

4 Experiment

In this section, we first present the benchmark results on the standard neural operator benchmark problems from Li et al. (2021a) and compare our model with several other state-of-the-art operator learning frameworks, including Galerkin/Fourier Transformer (G.T. / F.T.) from (Cao, 2021) and Multiwavelet-based Operator (MWT) (Gupta et al., 2021). Then we showcase the model's application to irregular grids where above frameworks cannot be directly applied to, and compare model to a graph neural network baseline (Lötzsch et al., 2022). We also perform an analysis on the model's latent encoding. Finally, we present an ablation study of key architectural choices.

The full details of model architecture on different problems and training procedure are provided in the Appendix A. More ablation study of hyperparameter choices are carried out in Appendix C.

4.1 Benchmark problems

Below we provide a brief description of investigated PDEs. Full details of these equations' definition and dataset generation are provided in the Appendix D.

Problems on uniform grid The first kind of problems considered include 1D viscous Burgers' equation, 2D Darcy flow, and 2D imcompressible Navier-Stokes equation. In these problems, the data are represented on a uniform equi-spaced grid, where discrete Fourier transformation (FNO, G.T./F.T.) and wavelet transformation (MWT) can be efficiently applied to.

The objective solution operators $\mathcal{G}$ aimed to learn are defined as (T is the time horizon, u denotes solution function of interest, a denotes the coefficient function):

$$\textit{Burgers'}: \quad \mathcal{G} : u(x,t)|_{t=0} \mapsto u(x,t)|_{t=1}, \quad x \in (0,1), \tag{12}$$

$$\textit{Darcy flow}: \quad \mathcal{G} : a(x) \mapsto u(x), \quad x \in (0,1)^2, \tag{13}$$

$$\textit{Navier-Stokes}: \quad \mathcal{G} : u(x,t)|_{t\in[0,10]} \mapsto u(x,t)|_{t\in(10,T]}, \quad x \in (0,1)^2. \tag{14}$$

On Navier-Stokes equation, following prior work (Li et al., 2021a), we adopt 3 datasets NS1, NS2, NS3 with different viscosities, and we generate a more challenging and diverse dataset (denoted as NS-mix) where ν is not constant, and it is uniformly sampled: $\nu \sim \mathcal{U}([0.4, 1]) \times 10^{-4}$, corresponding to Reynolds numbers roughly ranging from 200 to 500. For the size of each Navier-Stokes dataset, NS2-full contains 9800/200 (train/test) samples; NS-mix is a larger dataset consisting of 10000/1000 samples; each of the rest datasets contains 1000/200 samples. For Burgers' equation and Darcy flow, the dataset splitting is same as in Cao (2021).

Problems on non-uniform grid Other than uniform grid, the second kind of problems considered are based on unstructured non-uniform grid. We first investigate the 2D Poisson equation of magnetic/electric field on different geometries, the objective is to find the solution of field given boundary conditions. We use the shape extrapolation dataset from (Lötzsch et al., 2022). The training set contains grids of four shapes (square, circle with or without hole, L-shape) with 8000 samples, while testing set has 2000 samples consisting of U-shape grids that model has never seen during training. This experiment tests model's generalization capability to different solution domain. In addition, we study the 2D time-dependent compressible flow around the cross-section of airfoils using the dataset from Pfaff et al. (2020). In this problem, the underlying grid is highly irregular, as the distance between two closest points ranges from $2e-4$ to 3.5. The dataset contains 1000/100 sequences of different inflow speed (Mach number) and angles of attack.

The objective solution operators $\mathcal{G}$ with respect to these problems are defined as:

$$\textit{Poisson}: \quad \mathcal{G} : \{u(x)|_{x\in\partial\Omega}, f(x)\} \mapsto u(x), \quad x \in \Omega, \tag{15}$$

$$\textit{Airfoil}: \quad \mathcal{G} : \mathbf{u}(x,t)|_{t\in[0,0.576]} \mapsto \mathbf{u}(x,t)|_{t\in(0.576,4.800]}, \quad x \in \Omega, \tag{16}$$

where Ω is the solution spatial domain and $f(x)$ is the source term of the Poisson equation. For Poisson equation we predict the field vector in addition to the scalar potential [2]. For compressible flow we predict velocity, density and pressure jointly.

4.2 Results and discussion

For benchmark results of all baselines, we report the evaluation results from original paper when applicable. When not applicable, we train and evaluate the model following the suggested setting in the original paper, and mark the results with "*". The best version of Galerkin/Fourier Transformer is reported at each problem. Details of baseline models' implementation and comparison of computing budget can be found in the Appendix B.

[2]The electric field $\mathbf{E}$ is defined as the negative gradient of potential: $\mathbf{E} = -\nabla u$. For magnetic field $\mathbf{B}$, it is defined as the curl of magnetic potential (scalar in this problem): $\mathbf{B} = \nabla \times u$. In the network we use a separate head to predict them.

Performance on equi-spaced grid The comparison results with several state-of-the-art baselines are shown in Table 1, 2, 3. We observe that for small data regime (NS2-part, NS3) and problems with relatively smooth target function (viscid Burgers', Darcy), OFormer generally does not show superiority over other operator learning-frameworks. Architectures like spectral convolution (in FNO, Galerkin/Fourier Transformer) are baked in helpful inductive bias when target is smooth and periodic. As shown in Table 2, equipped with a smooth activation function, FNO+ outperforms all other methods significantly, as the target function $u(\cdot, 1)$ is relatively smooth due to the presence of diffusion term, and has a periodic characteristic in the spatial domain. For more complex instances like Navier-Stokes equation, we find that OFormer has strong performance in data sufficient regime, where it is on par with MWT on NS2-full with substantially less parameters. This highlights the learning capacity of attention-based architecture, which leverages flexible basis learned from data. Moreover, our model performs similarly under different resolutions, indicating its robustness to different resolution. (Contour plot of the prediction and corresponding error can be found in Appendix E)

Data settings		Relative L_2 norm				
Case	ν, T	FNO-2D	FNO-3D	MWT	G.T.*	OFormer
NS1	$1 \times 10^{-3}, 50$	0.0128	0.0086	**0.0062**	0.0098	0.0104
NS2-part	$1 \times 10^{-4}, 30$	0.1559	0.1918	0.1518	**0.1399**	0.1708
NS2-full	$1 \times 10^{-4}, 30$	0.0834	0.0820	0.0667	0.0792	**0.0646**
NS3	$1 \times 10^{-5}, 20$	0.1556	0.1893	0.1541	**0.1340**	0.1697
NS-mix*	$[0.4, 1] \times 10^{-4}$, 30	0.1650	0.1654	**0.1267**	0.1462	0.1400
# of parameters (M)		0.41/2.37	6.56	179.18	1.56	1.85

Table 1: Benchmark on 2D Navier-Stokes equation with fixed 64×64 grid. For large dataset (NS-mix) we use a larger version of FNO-2D by increasing modes and width. We adopt Galerkin Transformer (G.T.) with a larger hidden dimension ($64 \rightarrow 96$) to match the size of other models.

Res.	Relative L_2 norm ($\times 10^{-3}$)				
	FNO*	FNO+*	MWT	F.T.	OFormer
512	3.15	**0.72**	1.85	1.14	1.37
2048	3.07	**0.72**	1.86	1.12	1.26
8192	2.76	**0.70**	1.78	1.07	1.52

Table 2: Benchmark on 1D Burgers' equation with different resolution. FNO uses ReLU (Nair & Hinton, 2010) whereas FNO+ uses GELU. Both FNO variants remove normalization compared to the original paper.

Res.	Relative L_2 norm ($\times 10^{-2}$)		
	FNO	G.T.	OFormer
141	1.09	**0.84**	1.24
211	1.09	**0.84**	1.29

Table 3: Benchmark on 2D Darcy flow with different resolution. MWT's result is not reported since its implementation requires resolution to be power of 2 and thus it is trained on a different data.

Application to irregular grids The comparison between OFormer and GNN baseline on the 2D Poisson problem are shown in Table 4. Both OFormer and GNN have the capability to learn and extrapolate to new geometries with reasonable accuracy under mesh augmentation. However, we can observe that OFormer outperforms GNN by a margin when no mesh augmentation is applied. This highlights an essential difference of the learning mechanism between OFormer and local graph convolution-based architecture. GNN that exploits local mesh information such as edge distance and node connection is more prone to overfit on the training geometries, while OFormer, leveraging global attention and not using any explicit graph information, is more robust under the change of local mesh topology. Other than grid with relatively uniform point distribution, OFormer is also applicable to highly irregular meshes. We showcase that OFormer is able to predict the temporal evolution of compressible flow around the airfoil (exemplar visualization is shown at Figure 4, more visualization results can be found in Appendix E). The test root mean squared error (at region closed to airfoil) of the predicted momentum is 15.477, with the average magnitude of momentum at this region is 153.456.

Another important property of OFormer is that it can handle variable number of input/query points. Here we showcase an application of such property. During training, we randomly drop some of the input points with a ratio of $r \sim \mathcal{U}([0, 0.5])$, but still query at full resolution. we test the model by randomly sampling 25% of the grid points in the spatial domain, which is 150% of the maximum dropping ratio during training. As shown in Figure 5, the model can still predict reasonably accurate results on a full resolution grid. (More quantitative results can be found in Appendix C)

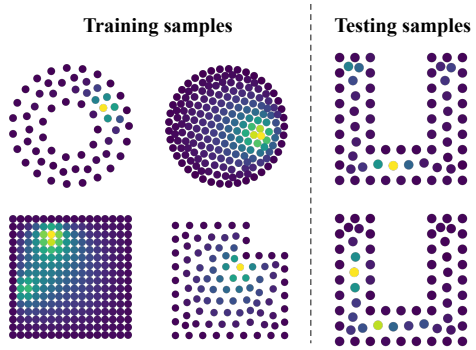


Figure 3: Visualization of training/testing samples in 2D Poisson problem. Testing set contains new geometries that never appeared in the training set.

Mesh aug.	Model	Mean squared error ($\times 10^{-3}$)			
		Electric		Magnetic	
		Potential	Field	Potential	Field
No	OFormer	**0.155**	**1.512**	**0.111**	**1.274**
	GNN	0.327	2.723	0.161	2.640
Yes	OFormer	0.019	**0.325**	0.018	**0.198**
	GNN	**0.006**	1.821	**0.006**	1.338

Table 4: Benchmark on electrostatics and magnetostatics' Poisson equation. The train/evaluation protocol is adopted from (Lötzsch et al., 2022). Mesh augmentation comprises a set of random transformation on the training meshes, including varying node density and size of hole/cutout.

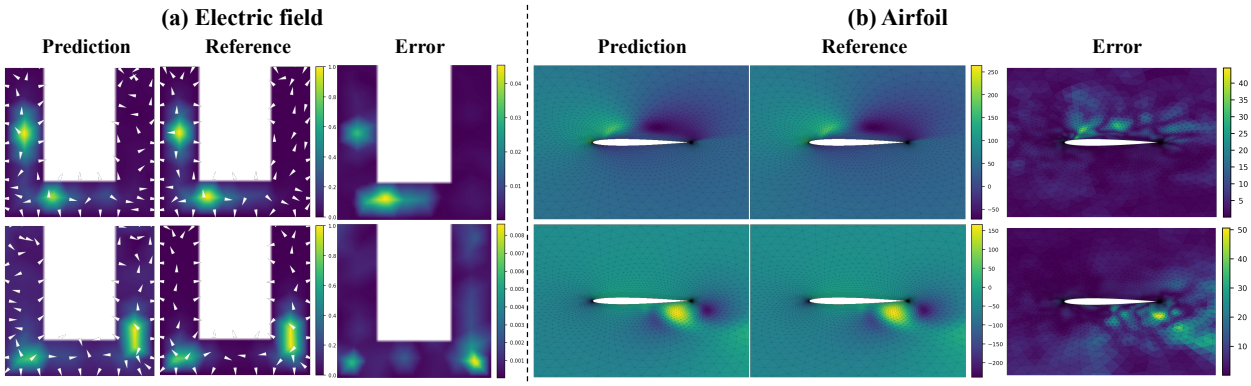


Figure 4: Samples of model's prediction, reference simulation result and absolute error on: **(a)** Electric potential and corresponding field; **(b)** Velocity field around airfoil, top/bottom: x/y component.

Learned spatio-temporal representation To inspect OFormer's ability to learn meaningful representations of the PDEs, we perform an analysis on our model trained on a dataset of Navier-Stokes equation where each sample has a random coefficient of $\log(\nu) \sim \mathcal{U}([-3, -5.3])$, which resulted in ν falls in the range of $(5e-6, 1e-3)$. The model is trained to predict the future time steps, where it has never been informed of the coefficient values. After training, we examine the latent space of the input encoder.

A single d-dimensional representation vector $\mathbf{f}_{\text{avg}}^{(k)}$ of k-th sample is obtained by applying average pooling on the encodings $\mathbf{f}_{\text{avg}}^{(k)} = \text{Avg}(\{\mathbf{f}_i^{(k)}\}_{i=1}^n)$ over the input points. First, we project the representation vectors of all training samples into a 2-dimensional space using Principal Component Analysis (PCA). We can observe the evident trend of colors w.r.t. ν in Figure 6(a). Furthermore, we train a linear regression model on the latent vectors from the training samples, and then we use it to predict for the ν on testing samples. The Figure 6(b) show that a linear model can effectively map the latent representations to the viscosity coefficients which indicates the model's capability of finding compact representations such that a nonlinear mapping has become linear.

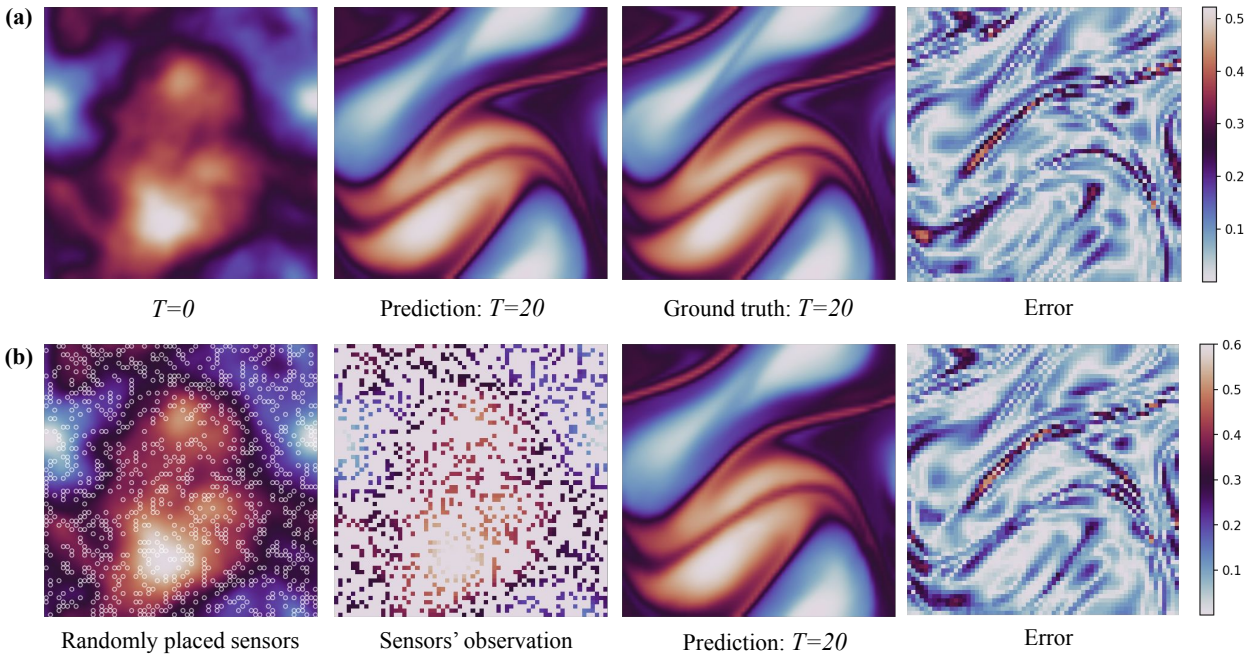


Figure 5: **(a)** Model's prediction on dense uniform input points; **(b)** Model's prediction on sparse randomly sampled input points (25% of all grid points).

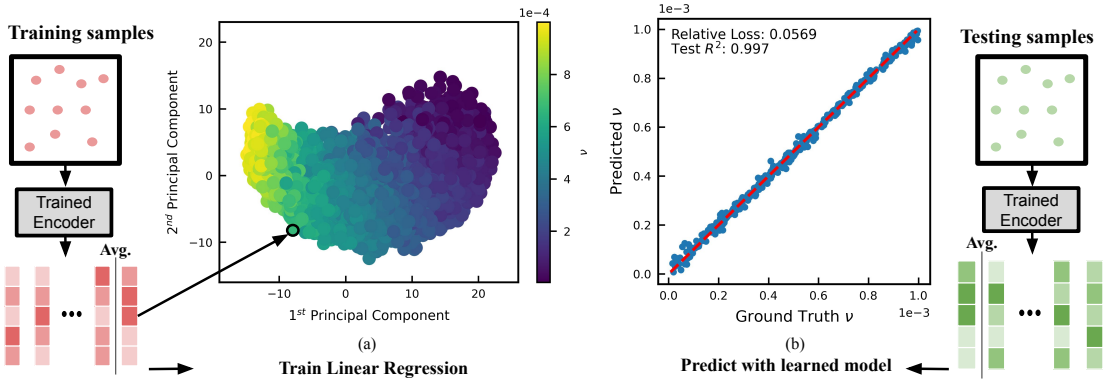


Figure 6: Learned representations of the model trained on Navier-Stokes equations with different viscosity coefficients. **(a)** The 2-dimensional PCA on the latent vectors. **(b)** A linear regression model trained on the latent vectors of training samples can accurately predict ν of testing samples.

4.3 Ablation study

We study the influence of the following factors on the model's performance: (i) The type of attention; (ii) Model's scaling performance; (iii) Positional encoding; (iv) Different normalization methods. The quantitative ablation results are shown in Figure 7.

First of all, we find that the performance of a specific type of attention is problem-specific, which signifies the importance of choosing normalization targets (query/key or key/value). In addition, we observe that on a scale-sensitive problem (data are not normalized for Burgers' equation to preserve the energy decaying property), InstanceNorm which normalizes each column vector $||\mathbf{v}^j||_2 = 1$, performs better than LayerNorm, further facilitating the bases interpretation of two linear attention mechanisms. Scaling up the model's size

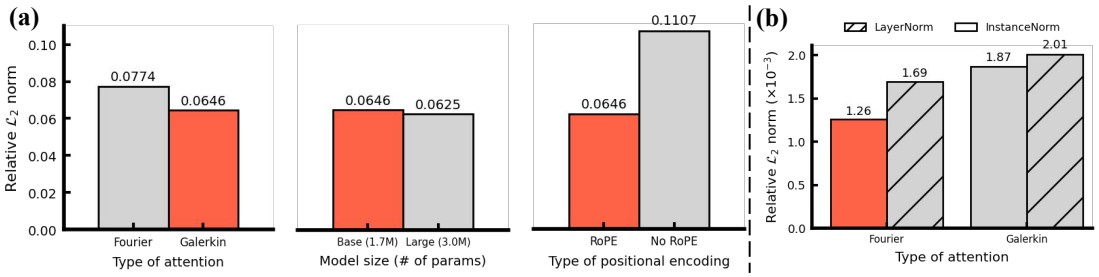


Figure 7: Ablation study on **(a)** NS2-full dataset; **(b)** 1D Burgers' equation with resolution 2048. Light color denotes option we adopted.

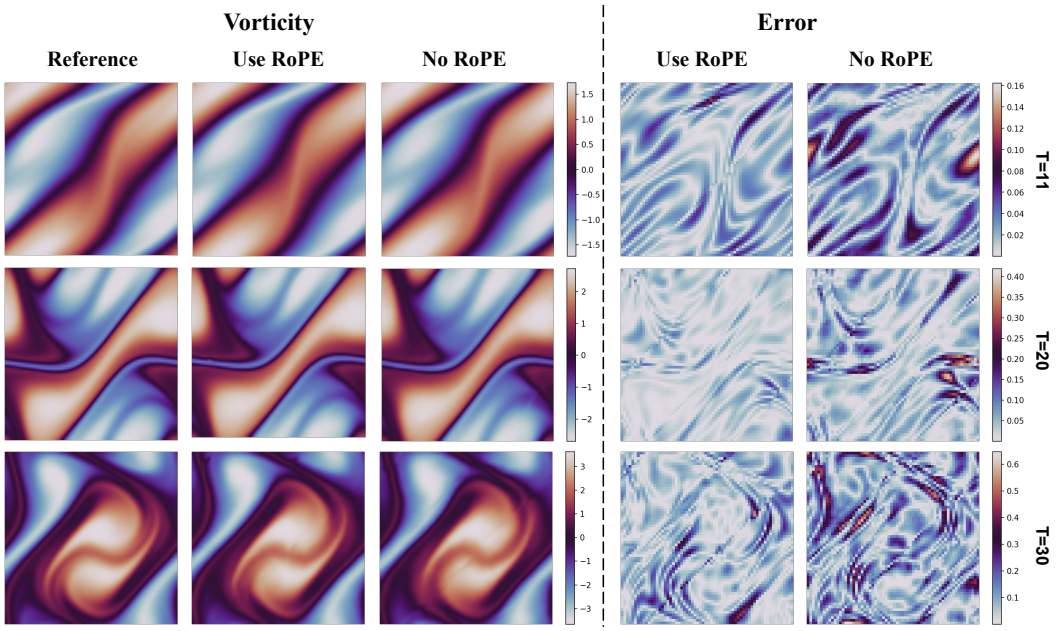


Figure 8: Influence of positional encoding, the predicted sequence is randomly chosen. Error is measured by absolute error.

(increasing the hidden dimension) can benefit the model's performance in a data sufficient regime. On NS-full dataset, the larger version of OFormer pushes the previous best result from 0.0667 (Gupta et al., 2021) to 0.0625. Lastly, we find that if RoPE is removed and replaced by concatenating coordinates into query and key vectors (after projection), the model's performance deteriorates significantly. As shown in Figure 8, without Rotary Position Embedding (RoPE), the model's prediction becomes relatively blurrier and generally has larger error. This signifies the importance of relative position information in PDE operator learning.

5 Conclusion

In this work, we introduce OFormer, a fully point-based attention architecture for PDEs' solution operator learning. The proposed model shows competitive performance on PDE operator learning and offers flexibility with input/output discretizations. We also showcase that a trained data-driven model can learn to generalize to diverse instances of PDEs without inputting system parameters (NS-mix) and learn meaningful representation for system identification. The major limitation is that given few assumptions are made on the grid structure, the model needs sufficient data to reach its optimal performance, and linear attention mechanism is still relatively computational expensive for higher resolution grids. In addition, just like most of the other data-driven models, the accumulated error of the model's prediction on long-term transient phenomena such as turbulent fluids is unneglectable, unlike numerical solvers which are usually guaranteed to be stable.

References

Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020. URL https://openreview.net/forum?id=fg2ZFmXFO3.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

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.

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, Aug 2019. ISSN 1432-0924. doi: 10.1007/s00466-019-01740-0. URL https://doi.org/10.1007/s00466-019-01740-0.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M. Stuart. Model Reduction And Neural Networks For Parametric PDEs. *The SMAI journal of computational mathematics*, 7:121–157, 2021. doi: 10.5802/smai-jcm.74. URL https://smai-jcm.centre-mersenne.org/articles/10.5802/smai-jcm.74/.

Stella Biderman, Sid Black, Charles Foster, Leo Gao, Eric Hallahan, Horace He, Ben Wang, and Phil Wang. Rotary embeddings: A relative revolution, 2021. [Online; accessed].

Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vSix3HPYKSU.

Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52(1):477–508, 2020. doi: 10.1146/annurev-fluid-010719-060214. URL https://doi.org/10.1146/annurev-fluid-010719-060214.

Shengze Cai, Zhicheng Wang, Lu Lu, Tamer A. Zaki, and George Em Karniadakis. Deepm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *Journal of Computational Physics*, 436:110296, 2021. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2021.110296. URL https://www.sciencedirect.com/science/article/pii/S0021999121001911.

Shuhao Cao. Choose a transformer: Fourier or galerkin. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ssohLcmn4-r.

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.

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, 2020. URL https://arxiv.org/abs/2009.14794.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks, 2016. URL https://arxiv.org/abs/1612.08083.

Filipe de Avila Belbute-Peres, Thomas D. Economon, and J. Zico Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, Mar 2018. ISSN 2194-671X. doi: 10.1007/s40304-018-0127-z. URL https://doi.org/10.1007/s40304-018-0127-z.

Nicholas Geneva and Nicholas Zabaras. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, feb 2022. doi: 10.1016/j.neunet.2021.11.022. URL https://doi.org/10.1016%2Fj.neunet.2021.11.022.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017. URL https://arxiv.org/abs/1704.01212.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

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

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Efficient token mixing for transformers via adaptive fourier neural operators. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=EXHG-A3jlM.

Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 24048–24062. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/c9e5c2b59d98488fe1070e744041ea0e-Paper.pdf.

Xu Han, Han Gao, Tobias Pffaf, Jian-Xun Wang, and Li-Ping Liu. Predicting physics in mesh-reduced space with temporal attention, 2022. URL https://arxiv.org/abs/2201.09113.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016. URL https://arxiv.org/abs/1606.08415.

Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural PDE solvers with convergence guarantees. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rklaWn0qK7.

Valerii Iakovlev, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time {pde}s from sparse data with graph neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=aUX5Plaq7Oy.

Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention, 2021. URL https://arxiv.org/abs/2103.03206.

George Karniadakis, Yannis Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. pp. 1–19, 05 2021. doi: 10.1038/s42254-021-00314-5.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021. doi: 10.1017/S0956792520000182.

Georgios Kissas, Jacob H. Seidman, Leonardo Ferreira Guilhoto, Victor M. Preciado, George J. Pappas, and Paris Perdikaris. Learning operators with coupled attention. *ArXiv*, abs/2201.01032, 2022.

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):e2101784118, 2021. doi: 10.1073/pnas.2101784118. URL https://www.pnas.org/doi/abs/10.1073/pnas.2101784118.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces, 2021. URL https://arxiv.org/abs/2108.08481.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 26548–26560. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/df438e5206f31600e6ae4af72f2725f1-Paper.pdf.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks, 2018. URL https://arxiv.org/abs/1810.00825.

Kookjin Lee and Kevin T. Carlberg. Deep conservation: A latent-dynamics model for exact satisfaction of physical conservation laws. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):277–285, May 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16102.

Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional koopman operators for model-based control. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=H1ldzA4tPr.

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.

Zijie Li, Kazem Meidani, Prakarsh Yadav, and Amir Barati Farimani. Graph neural networks accelerated molecular dynamics. *The Journal of Chemical Physics*, 156(14):144103, 2022. doi: 10.1063/5.0083060. URL https://doi.org/10.1063/5.0083060.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6755–6766. Curran Associates, Inc., 2020b. URL https://proceedings.neurips.cc/paper/2020/file/4b21cf96d4cf612f239a6c322b10c8fe-Paper.pdf.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=c8P9NQVtmnO.

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, 2021b. URL https://arxiv.org/abs/2111.03794.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, Mar 2021a. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL https://doi.org/10.1038/s42256-021-00302-5.

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, 2021b. doi: 10.1137/19M1274067. URL https://doi.org/10.1137/19M1274067.

Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.114778. URL https://www.sciencedirect.com/science/article/pii/S0045782522001207.

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.

Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, Nov 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-07210-0. URL https://doi.org/10.1038/s41467-018-07210-0.

Winfried Lötzsch, Simon Ohler, and Johannes S. Otterbach. Learning the solution operator of boundary value problems using graph neural networks. 2022. doi: 10.48550/ARXIV.2206.14092. URL https://arxiv.org/abs/2206.14092.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. URL https://arxiv.org/abs/2003.08934.

Jeremy Morton, Antony Jameson, Mykel J Kochenderfer, and Freddie Witherden. Deep dynamical modeling and control of unsteady fluid flows. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/2b0aa0d9e30ea3a55fc271ced8364536-Paper.pdf.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Nicholas H. Nelsen and Andrew M. Stuart. The random feature model for input-output maps between banach spaces. *SIAM J. Sci. Comput.*, 43:A3212–A3243, 2021.

Francis Ogoke, Kazem Meidani, Amirreza Hashemi, and Amir Barati Farimani. Graph convolutional networks applied to unstructured flow field data. *Machine Learning: Science and Technology*, 2(4):045020, sep 2021. doi: 10.1088/2632-2153/ac1fc9. URL https://doi.org/10.1088/2632-2153/ac1fc9.

Francisco Palacios, Juan Alonso, Karthikeyan Duraisamy, Michael Colonno, Jason Hicken, Aniket Aranake, Alejandro Campos, Sean Copeland, Thomas Economon, Amrita Lonkar, Trent Lukaczyk, and Thomas Taylor. Stanford University Unstructured (SU^2): An open-source integrated computational environment for multi-physics simulation and design. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics, 2013. doi: 10.2514/6.2013-287. URL https://doi.org/10.2514/6.2013-287.

Shaowu Pan and Karthik Duraisamy. Physics-informed probabilistic learning of linear embeddings of nonlinear dynamics with guaranteed stability. *SIAM Journal on Applied Dynamical Systems*, 19(1):480–509, 2020. doi: 10.1137/19M1267246. URL https://doi.org/10.1137/19M1267246.

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, 2021. doi: 10.1063/5.0062546. URL https://doi.org/10.1063/5.0062546.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

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. URL https://arxiv.org/abs/2202.11214.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. 2020. doi: 10.48550/ARXIV.2010.03409. URL https://arxiv.org/abs/2010.03409.

Md Ashiqur Rahman, Zachary E. Ross, and Kamyar Azizzadenesheli. U-no: U-shaped neural operators, 2022. URL https://arxiv.org/abs/2204.11127.

M. Raissi, P. Perdikaris, and G.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. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Priyabrata Saha and Saibal Mukhopadhyay. A deep learning approach for predicting spatiotemporal dynamics from sparsely observed data. *IEEE Access*, 9:64200–64210, 2021. doi: 10.1109/ACCESS.2021.3075899.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020. URL https://arxiv.org/abs/2002.09405.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2013. URL https://arxiv.org/abs/1312.6120.

William E. Schiesser. The numerical method of lines: Integration of partial differential equations. 1991.

M. W. Scroggs, J. S. Dokken, C. N. Richardson, and G. N. Wells. Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes. *ACM Transactions on Mathematical Software*, 2022. doi: 10.1145/3524456. To appear.

Yidi Shao, Chen Change Loy, and Bo Dai. Sit: Simulation transformer for particle-based physics simulation, 2022. URL https://openreview.net/forum?id=DBOibe1ISzB.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.08.029. URL https://www.sciencedirect.com/science/article/pii/S0021999118305527.

Kim Stachenfeld, Drummond Buschman Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned simulators for turbulence. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=msRBojTz-Nh.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2021. URL https://arxiv.org/abs/2104.09864.

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. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2019.112732. URL https://www.sciencedirect.com/science/article/pii/S004578251930622X.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 1130–1140, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

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. URL https://arxiv.org/abs/2006.10739.

Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized fourier neural operators, 2021. URL https://arxiv.org/abs/2111.13802.

Tapas Tripura and Souvik Chakraborty. Wavelet neural operator: a neural operator for parametric partial differential equations, 2022. URL https://arxiv.org/abs/2205.02191.

Kathryn Tunyasuvunakool, Jonas Adler, Zachary Wu, Tim Green, Michal Zielinski, Augustin Žídek, Alex Bridgland, Andrew Cowie, Clemens Meyer, Agata Laydon, Sameer Velankar, Gerard Kleywegt, Alex Bateman, Richard Evans, Alexander Pritzel, Michael Figurnov, Olaf Ronneberger, Russ Bates, Simon Kohl, and Demis Hassabis. Highly accurate protein structure prediction for the human proteome. *Nature*, 596: 1–9, 08 2021. doi: 10.1038/s41586-021-03828-1.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2016. URL https://arxiv.org/abs/1607.08022.

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 (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017. URL https://arxiv.org/abs/1710.10903.

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, pp. 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.

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, 2021a. doi: 10.1126/sciadv.abi8605. URL https://www.science.org/doi/abs/10.1126/sciadv.abi8605.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Improved architectures and training algorithms for deep operator networks, 2021b. URL https://arxiv.org/abs/2110.01654.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021c. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.113938. URL https://www.sciencedirect.com/science/article/pii/S0045782521002759.

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. ISSN 0309-1708. doi: https://doi.org/10.1016/j.advwatres.2022.104180. URL https://www.sciencedirect.com/science/article/pii/S0309170822000562.

Steffen Wiewel, Moritz Becher, and Nils Thürey. Latent space physics: Towards learning the temporal evolution of fluid flow. *Computer Graphics Forum*, 38, 2019.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270.

Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.04.018. URL https://www.sciencedirect.com/science/article/pii/S0021999118302341.

Appendix Summary

- Section A: Model implementation details;
- Section B: Baseline details;
- Section C: Further ablation study;
- Section D: Dataset details;
- Section E: Further results and visualization;

A Model implementation details

Below we provide the full implementation details of models used in different problems. All the models are implemented in PyTorch (Paszke et al., 2019).

For notation, we use $[d_1, d_2]$ to denote a linear/hidden layer with input dimension d_1 and output dimension d_2, which corresponds to nn.Linear(d1, d2) in PyTorch. For attention layer, $d \times h$ indicates that there are h attention heads each with a dimension of d.

A.1 Hyperparameter for equi-spaced grid problem

Problem	Input encoder top	SA hidden dim	SA FFN	# SA Block	Input encoder bottom
1D Burgers	[2, 96]	96×1	[96, 192, 96]	4	[96, 96]
2D Darcy flow	[3, 96]	96×4	[96, 192, 96]	4	[96, 256]
2D Navier-Stokes	[12, 128]	128×1	[128, 256, 128]	5	[128, 192]

Table 5: Architecture of input encoder. SA denotes self-attention, FFN denotes feed forward network.

Problem	Query encoder top	CA hidden dim	CA FFN	Query encoder bottom
1D Burgers	[1, 96, 96]	96×8	[96, 192, 96]	-
2D Darcy flow	[2, 256, 256]	256×4	[256, 512, 256]	-
2D Navier-Stokes	[2, 192, 192]	192×4	[192, 384, 196]	[192, 384]

Table 6: Architecture of query encoder. CA denotes cross-attention.

Problem	Propagator	Decoder	Total # of params (M)
1D Burgers	[96, 96, 96, 96]×3	[96, 48, 1]	0.67
2D Darcy flow	-	[256, 128, 64, 1]	2.51
2D Navier-Stokes	[384, 384, 384, 384]	[384, 192, 96, 1]	1.85

Table 7: Architecture of propagator and decoder. The number of parameters is the total number including input/query encoders. For 1D Burgers' equation we use 3 unshared MLPs for propagating the dynamics.

A.2 Hyperparameter for irregular grid problem

We use GELU (Hendrycks & Gimpel, 2016) as activation function for propagator and decoder, and opt for Gated-GELU (Dauphin et al., 2016) for FFN. On 1D Burgers, we train the model for 20k iterations using a batch size of 16. On Darcy flow, we train the model for 32K iterations using a batch size of 8. For small dataset on Navier-Stokes (consisting of 1000 training samples), we train the model for 32k iterations using a batch size of 16, and 128k iterations for large dataset (consisting of roughly 10k training samples). On Electrostatics/Magnetostatics we train the model for 32k iterations using a batch size of 16. On Airfoil train model for 50k iterations using a batch size of 10.

Problem	Input encoder top	SA hidden dim	SA FFN	# SA Block	Input encoder bottom
2D Electrostatics	[11, 64, 64]	64×1	[64, 128, 64]	2	[64, 64]
2D Magnetostatics	[11, 96, 96]	96×1	[96, 128, 96]	2	[96, 96]
2D Airfoil	[6, 128, 128]*	128×1	[128, 256, 128]	5	[128, 128]

Table 8: Architecture of input encoder. The input of 2D airfoil contains an additional time dimension of length t_{in}, so the very top layer is a Conv2D layer with kernel size $(t_{in}, 1)$ stride $(t_{in}, 1)$ and no padding.

Problem	Query encoder top	CA hidden dim	CA FFN	SA hidden dim	Query encoder bottom
2D Electrostatics	[3, 64, 64]	64×4	[64, 128, 64]	64×1	-
2D Magnetostatics	[3, 96, 96]	96×4	[96, 192, 96]	96×1	-
2D Airfoil	[2, 128, 128]*	128×4	[128, 256, 128]	128×1	[128, 256]

Table 9: Architecture of query encoder. On non-uniform grid, we add a self-attention layer after cross-attention layer. For 2D airfoil, we introduce a learnable embedding for each node type (boundary, open area, airfoil)

Problem	Propagator	Decoder	Total # of params (M)
2D Electrostatics	-	[64, 32, 32, 3]	0.17
2D Magnetostatics	-	[96, 48, 48, 3]	0.31
2D Airfoil	[384, 256, 256, 256]	[256, 128, 128, 4]	1.35

Table 10: Architecture of propagator and decoder. The number of parameters is the total number including input/query encoders. For airfoil, before inputting into propagator block, node type embedding are concatenated together with other features.

For training on the 2D Navier-Stokes and Airfoil problem, we adopt a curriculum strategy to grow the prediction time steps. During the initial stage, instead of predicting all the future states towards the end of time horizon, e.g. $u_{t_0}, u_{t_1}, \ldots, u_T$, we truncate the time horizon with a ratio $\gamma < 1$ (heuristically we choose $\gamma \approx 0.5$), train the network to predict $u_{t_0}, u_{t_1}, \ldots, u_{\gamma T}$. The motivation is that at the initial stage the training dynamics is less stable and gradient is changing violently, we truncate the time horizon to avoid stacking very deep recurrent neural network and thus alleviate the unstable gradient propagation. In practice we found this makes the training more stable and converge slightly faster.

As suggested in (Cao, 2021), for scaling-sensitive problem - the Burgers' equation, the initialization of the query/value projection layer has a great influence on the model's performance. We use orthogonal initialization (Saxe et al., 2013) with gain $1/d$ and add an additional constant $1/d$ to the diagonal elements (with d being the dimension of latent dimension at each head). The orthogonal weights have the benefit of preserving the norm of input and we found that with proper scaling it provides better performance than standard Xavier uniform initialization. Please refer to the Appendix C for the study on the influence of initialization.

For most of the problems we studied, we use instance normalization in the attention and observe slight improvement compared to layer normalization. However, for 2D Electrostatics/Magnetostatics' Poisson equation, we find that instance normalization is unstable as the grid points in this problem are much fewer (60 - 250) than other problems, so we adopt layer normalization for this problem.

B Baseline details

We use the open-sourced official implementation of Fourier Neural Operator (FNO)[3] (Li et al., 2021a), Multiwavelet Operator (MWT)[4] (Gupta et al., 2021) and Galerkin/Fourier Transformer (G.T./F.T.)[5] (Cao, 2021) to carry out the benchmark. The experiments on these baselines are carried out following the settings in the official implementation with minimal changes.

For FNO, when applying to 2D Navier-Stokes problem with relatively large number of data samples, we increase the number of modes in Fourier Transformation to 12 and increase the width (hidden dimension of network) to 32. This notably improves FNO-2D's performance. For G.T./F.T., on 2D Navier-Stokes problem we increase the hidden dimension size from 64 to 96, and extend the training epochs from 100 to 200 (150 on large dataset with 10000 training samples).

Below we provide a computational benchmark of different models' training and inference efficiency on 2D Navier-Stokes equation. The benchmark was conducted on a RTX-3090 GPU using PyTorch 1.8.1 (1.7 for MWT) and CUDA 11.0. The number of training samples is 10000, with time horizon $T = 30$, and the batch size is set to 10.

Model	Iters / sec	Memory (GB)	# of params (M)
FNO-2D	9.71	0.33	2.37
FNO-3D	10.74	0.42	6.56
MWT	1.67	10.95	179.18
G.T.	1.79	16.65	1.56
OFormer	1.89	15.93	1.85

Table 11: Training computational cost of different models.

Model	OFormer	G.T.	FNO-2D	FNO-3D	MWT	Pseudo-spectral	
						Fine	Coarse
Time (sec)	2.32	3.22	0.58	0.50	2.21	5735.74	3510.97

Table 12: Inference computational cost of different models and ground truth numerical solver. The time indicates how long does it take for each model to finish simulating 200 sequences (data processing time is excluded). The pseudo-spectral method is taken from (Li et al., 2021a), which is implemented in PyTorch. Fine grid has a resolution of 256×256 while coarse grid has a resolution of 64×64.

Note that as many traditional numerical solvers do not support GPU or not optimized for GPU computation, and they can trade-off accuracy for faster runtime (e.g. adopting a coarse grid and time step), so it is hard to conclude the exact acceleration of learned solvers on different problems, but in general we can observe that for time-dependent system learned solvers are highly efficient as they can tolerate a very large time step size (usually 3-4 orders of magnitudes compared to solver).

C Further ablation study

In this section we present further ablation study for the model.

Sparse reconstruction and prediction We study the model's performance under different discretization settings. At the training stage, if some of the input points (with probability $p = 0.2$ and dropping ratio $r \sim \mathcal{U}([0, 0.5])$) are randomly dropped, the model's performance are observed to deteriorate on most of the datasets. This indicates that while attention is flexible with respect to the input discretization, its best performances are usually obtained by exploiting the very specific grid structures. But for dataset that contains more diverse system parameters, like NS-mix and NS-mix2, the random dropping has little influence

[3] https://github.com/zongyi-li/fourier_neural_operator
[4] https://github.com/gaurav71531/mwt-operator
[5] https://github.com/scaomath/galerkin-transformer

on model's performance. While models trained with random dropping usually tend to perform worse than models trained with fixed grid points, they are much more robust on irregular discretization as shown in the Table 14 and can still generate reasonable prediction based on 25% of the input points.

Dataset	No random drop	Random drop	Performance diff.
NS1	0.0104	0.0133	0.0029 (27.9%)
NS2-part	0.1702	0.1773	0.0071 (4.2%)
NS2-full	0.0646	0.0672	0.0026 (4.0%)
NS3	0.1697	0.1747	0.0050 (2.9%)
NS-mix	0.1400	0.1413	0.0013 (0.9%)
1D Burgers	0.00126	0.00142	0.00016 (12.7%)

Table 13: Influence of randomly dropping some input grid points during the training process. No random drop indicates during training input and output grids are fixed at 64×64; random drop indicates that some are the input grid points are randomly dropped during training, while output grid is still fixed at 64×64. Error is measured by relative $\mathcal{L}_2$ norm. The resolution of Burgers' equation is 2048.

Training strategy	All	95%	75%	50%	25%
Random drop	0.0672	0.0687	0.0772	0.0939	0.1294
No random drop	0.0646	0.0795	0.1293	0.2065	0.3774

Table 14: Performance of models on NS2-full's test set with randomly sampled input points. Percentage on the top row indicates the ratio of sampled points' amount with respect to total number of all points. Error is measured by relative $\mathcal{L}_2$ norm on comparing full-resolution (64×64) prediction with ground truth.

Influence of initialization and positional encoding On Burgers' equation, to emulate the energy decaying scheme and preserve the magnitude of how much the energy has decayed, no data normalization or normalization layer (except for the instance normalization inside every attention layer) is used. While this improves model's performance, it also makes the model very sensitive to the initialization of projection matrix inside each attention layer, especially for the matrix where no instance normalization is applied (for Fourier type attention, it's the value matrix; for Galerkin type attention, it's the query matrix).

We modulate the initialization process of the weight of query (or value if using Fourier type attention) projection matrices ($\mathbf{W}_{(q)} \in \mathbb{R}^{d \times d}$) in Galerkin type attention as:

$$\mathbf{W}_{(q)} = \sigma \mathbf{B} + \delta \mathbf{I}, \tag{17}$$

where σ is a scaling factor which we set to $1/d$, $\mathbf{B}$ is the original weight matrix (initialized either via orthogonal initialization (Saxe et al., 2013) or Xavier uniform (Glorot & Bengio, 2010)), and similar to Cao (2021) we add diagonal bias $\delta \mathbf{I}$ to the matrix. The orthogonal initialization initializes a weight matrix that is norm preserving and projects the input feature vectors onto orthogonal embedding vectors. As shown in Figure 9, due to presence of skip connection: $\mathbf{z}' = \mathbf{z} + \mathrm{Attn}(\mathbf{z})$ and there is no normalization layer, both initialization methods blow up and have much worse performance when no scaling is applied. Orthogonal initialization tend to perform the best when both scaling (set gain to $1/d$) and diagonal bias are applied.

Influence of data OFormer's performance is highly correlated with the amount of data available, especially for relatively complicated problem like Navier-Stokes equation. As shown in Figure 10, the performance of the trained OFormer improves almost linearly as data size grow exponentially. At low data regime (500-5000), OFormer benefits significantly from increasing the data amount. This indicates that data is crucial for OFormer to unlock its optimal approximation power and also demonstrates OFormer's strong capacity to assimilate large data. However, this also hints a potential bottleneck of OFormer that it might not have satisfactory performance when there is limited data.

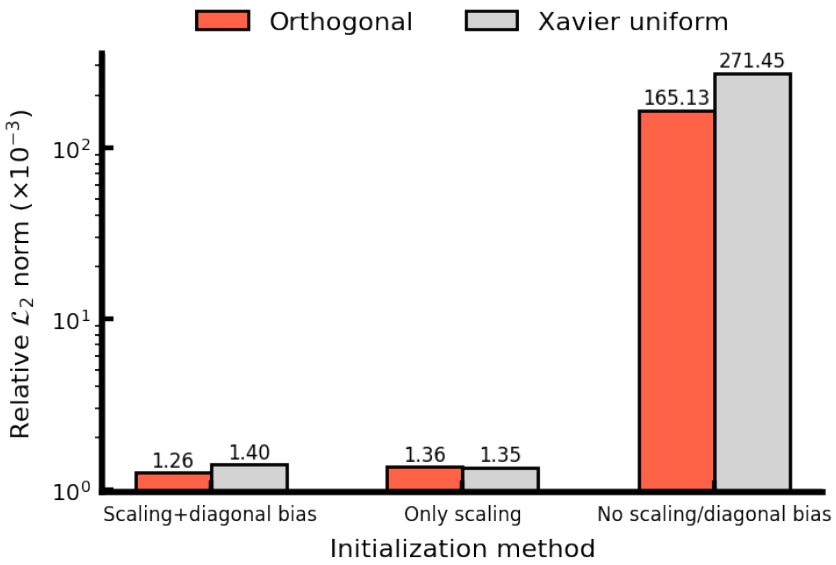


Figure 9: Influence of different initialization on scaling sensitive problem (1D Burgers' equation), using Fourier-type attention.

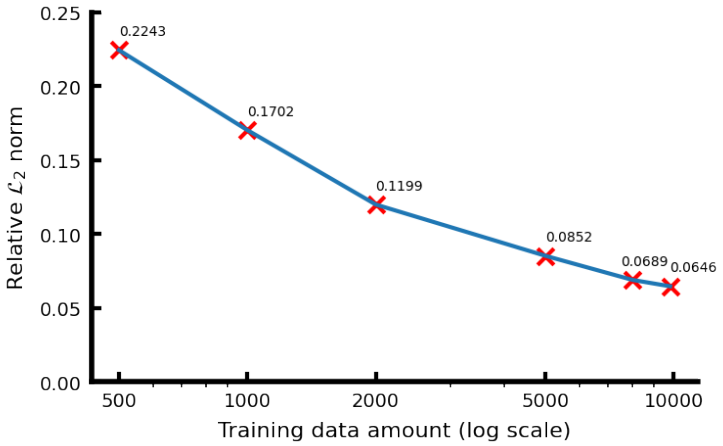


Figure 10: Performance of OFormer on NS2 when trained with different amount of data.

D Dataset details

D.1 Data generated on equi-spaced mesh

Following datasets[6] are used as benchmark in Li et al. (2021a); Cao (2021); Gupta et al. (2021), where the data is generated using equi-spaced simulation grid, and then downsampled to create dataset with different resolution.

Burgers' Equation The nonlinear 1D Burgers' equation with the following form is considered:

$$\begin{aligned} \partial_t u(x,t) &= -u\partial_x u(x,t) + \nu \partial_{xx} u(x,t), \quad x \in (0,1), t \in (0,1], \nu > 0, \\ u(x,0) &= u_0(x), \end{aligned} \tag{18}$$

[6]Dataset and numerical solver courtesy: `https://github.com/zongyi-li/fourier_neural_operator`

where ν is viscosity coefficient, the equation is under periodic boundary conditions, and initial condition $u_0(x)$ sampled from Gaussian Random Field prior. The viscosity coefficient is set to 0.1. The learning objective is an operator mapping from initial condition $u_0(x)$ to the solution at time $t = 1$, i.e., $\mathcal{G} : u_0(x) \mapsto u(x, 1)$. The data is generated via a split step method on 8192 resolution grid. At each time step, the state of the system is first advanced with the diffusion term and then the convection term (both calculated via Fourier transform), using the forward Euler scheme.

Darcy Flow The steady-state 2D Darcy flow equation considered here takes the form:

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x), \quad x \in (0,1)^2, \\ u_0(x) &= 0, \quad x \in \partial(0,1)^2, \end{aligned} \tag{19}$$

where $f(x)$ is the forcing function (set to constant 1) and we aim to learn the operator mapping diffusion coefficient $a(x)$ to the solution $u(x)$, i.e., $\mathcal{G} : a(x) \mapsto u(x)$. The coefficient function is sampled from a Gaussian Random Field prior with zero Neumann boundary conditions on the Laplacian kernel of the field. The data is generated via second-order finite difference solver on a 421×421 resolution grid.

Navier-Stokes Equation The vorticity form of 2D Navier Stokes equation for the viscous, incompressible fluid reads as:

$$\begin{aligned} \partial_t \omega(x,t) + \mathbf{u}(x,t) \cdot \nabla \omega(x,t) &= \nu \Delta \omega(x,t) + f(x), \quad x \in (0,1)^2, t \in (0,T], \\ \nabla \cdot \mathbf{u}(x,t) &= 0, \quad x \in (0,1)^2, t \in (0,T], \\ \omega(x,0) &= \omega_0(x), \quad x \in (0,1)^2, \end{aligned} \tag{20}$$

where ω is vorticity, $\mathbf{u}$ represents velocity field, and ν is the viscosity, source term is set as: $f(x) = 0.1\,(\sin(2\pi(x_1 + x_2)) + \cos(2\pi(x_1 + x_2)))$, and initial condition $\omega_0(x)$ sampled from a Gaussian random field. The equation is under periodic boundary conditions. The goal is to learn an operator mapping the initial states of the vorticity field to the future time steps, i.e., $\mathcal{G} : \omega|_{(0,1)^2 \times [0,10]} \mapsto \omega|_{(0,1)^2 \times (10,T]}$. Generally, lower viscosity coefficients ν result in more chaotic dynamics and makes the learning more challenging.

The data is generated by solving the stream-function ($\psi : u = \nabla \times \psi$) formulation of above equation using pseudo-spectral method on a 256×256 grid. At each time step, the stream function is first calculated by solving a Poisson equation ($\nabla^2 \psi = -\omega$). Then the non-linear term is calculated by differentiating voriticity in the Fourier space, dealiased and then multiplied with velocities in the physical space. Lastly, the Crank-Nicolson scheme is applied to update the state.

D.2 Data generated on non-equi-spaced mesh

Poisson equation The 2D Poisson equation for electrostatics and magnetostatics are defined as:

$$\begin{aligned} \textit{Electric}: \quad & -\nabla^2 U(x) = \frac{\rho(x)}{\epsilon(x)}, \quad x \in \Omega \\ \textit{Magnetic}: \quad & -\nabla^2 A_z(x) = \mu(x) I_z(x), \quad x \in \Omega \end{aligned} \tag{21}$$

where ρ is the charge density, ϵ is the permittivity of the material, U is the electric potential, μ is the permeability of the material, I_z is the z component of current density vector and A_z is the z component of magnetic potential. The goal is to learn the solution operator of the Dirichlet boundary value problem: $\mathcal{G} : \{U(x)|_{x \in \partial\Omega}, \rho(x), \epsilon(x)\} \mapsto U(x), x \in \Omega$ for electrostatics and $\mathcal{G} : \{A_z(x)|_{x \in \partial\Omega}, I_z(x), \mu(x)\} \mapsto A_z(x), x \in \Omega$ for magnetostatics. In the problem we studied, $\epsilon(x), \mu(x)$ are held constant.

The data is generated by solving the above equation using finite element method implemented with FEniCS library (Scroggs et al., 2022) [7]. The element used is a 2-degree triangle element. We use the pre-generated data from Lötzsch et al. (2022) for experiment.

[7] Dataset and numerical solver courtesy: `https://github.com/merantix-momentum/gnn-bvp-solver`

Euler equation of compressible flow The Euler equation of compressible flow takes the form:

$$\begin{aligned} \partial_t \rho + \nabla \cdot (\rho \mathbf{u}) &= f_1, \\ \partial_t (\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + p \mathbb{I}) &= \mathbf{f}_2, \\ \partial_t (\rho E) + \nabla \cdot (\rho E \mathbf{u} + p \mathbf{u}) &= f_3, \\ \rho := \rho(x,t), \mathbf{u} := \mathbf{u}(x,t), p &:= p(x,t), \\ x \in \Omega, t &\in [0,T], \end{aligned} \tag{22}$$

where ρ is the density, $\mathbf{u}$ is the velocity field, p is the pressure, E is the total energy per unit mass (which can be closed via $p = (\gamma - 1)\rho\left[E - 0.5(\mathbf{u} \cdot \mathbf{u})\right]$), and $f_1, \mathbf{f}_2, f_3$ are generic source terms. No-penetration condition is imposed at the airfoil. The solution operator of interest is: $\mathcal{G} : \{\mathbf{u}, \rho, p\}|_{x \in \Omega, t \in [0, 0.576]} \mapsto \{\mathbf{u}, \rho, p\}|_{x \in \Omega, t \in (0.576, 4.800]}$.

The dataset is generated by solving the above equation using finite volume method built in the SU2 library (Palacios et al., 2013). We use the pre-generated dataset[8] from Pfaff et al. (2020) to carry out the experiment. Note that different from Pfaff et al. (2020), we formulate this problem as a non-Markovian initial value problem and use a much large time step size ($\Delta t = 0.192$). In Pfaff et al. (2020), the learned solver is designed to learn the mapping: $u_t \mapsto u_{t+\Delta t}$ with $\Delta t = 0.008$ and then rollout the simulation with a Markovian setting.

E Further results and visualization

In this section we present the visualization of model's prediction on different problems. All error plots are based on point-wise absolute error.

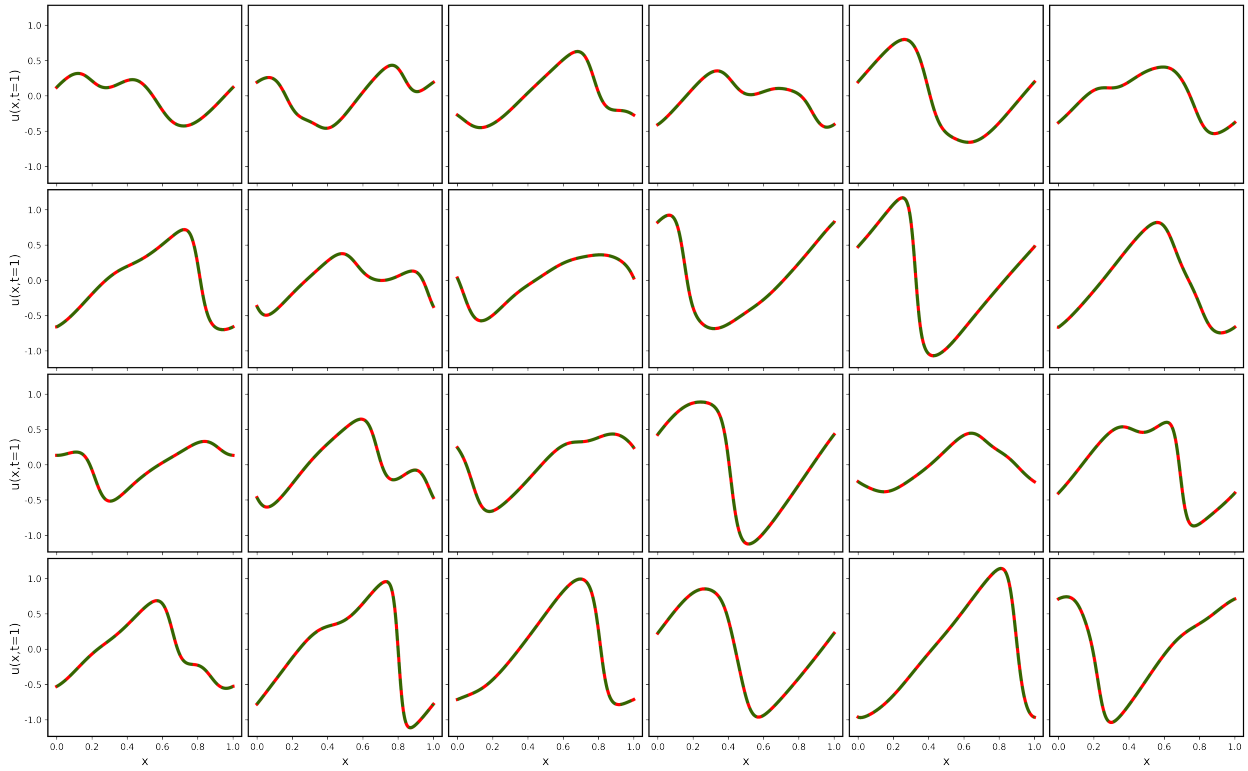


Figure 11: Model's prediction on 1D Burgers' equation, samples are randomly selected from test set. Green dotted lines denote the ground truth, and red lines denote model's prediction.

[8]Dataset courtesy: `https://github.com/deepmind/deepmind-research/tree/master/meshgraphnets`

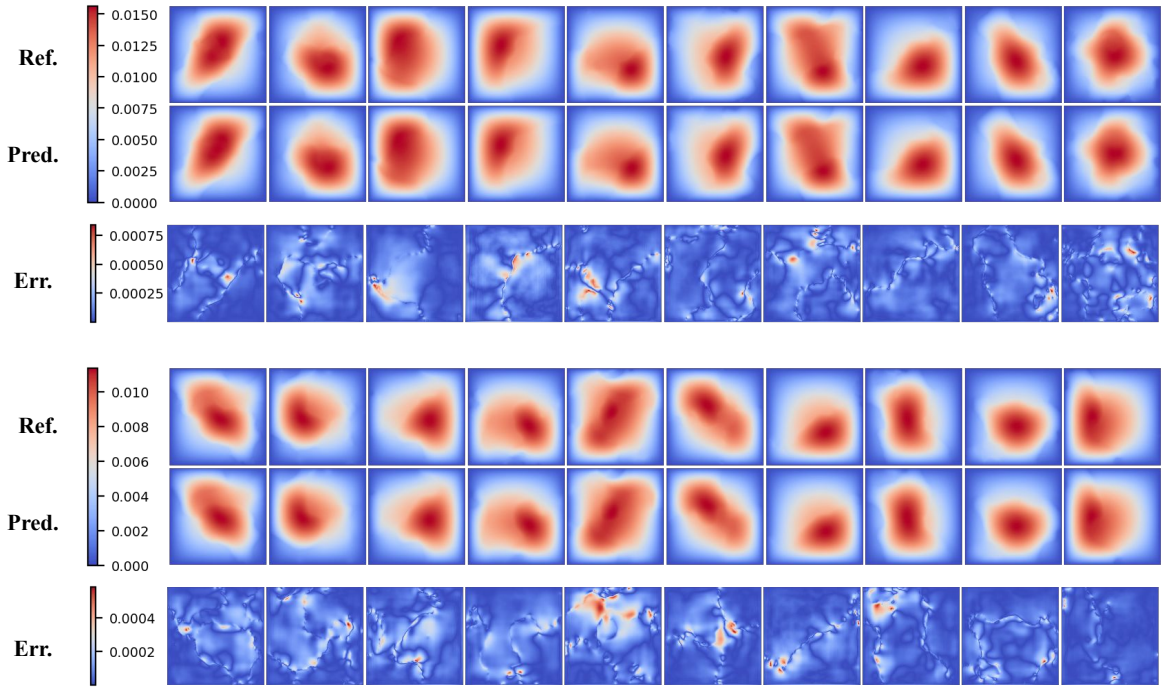


Figure 12: Model's prediction on 2D Darcy flow, samples are randomly selected from test set.

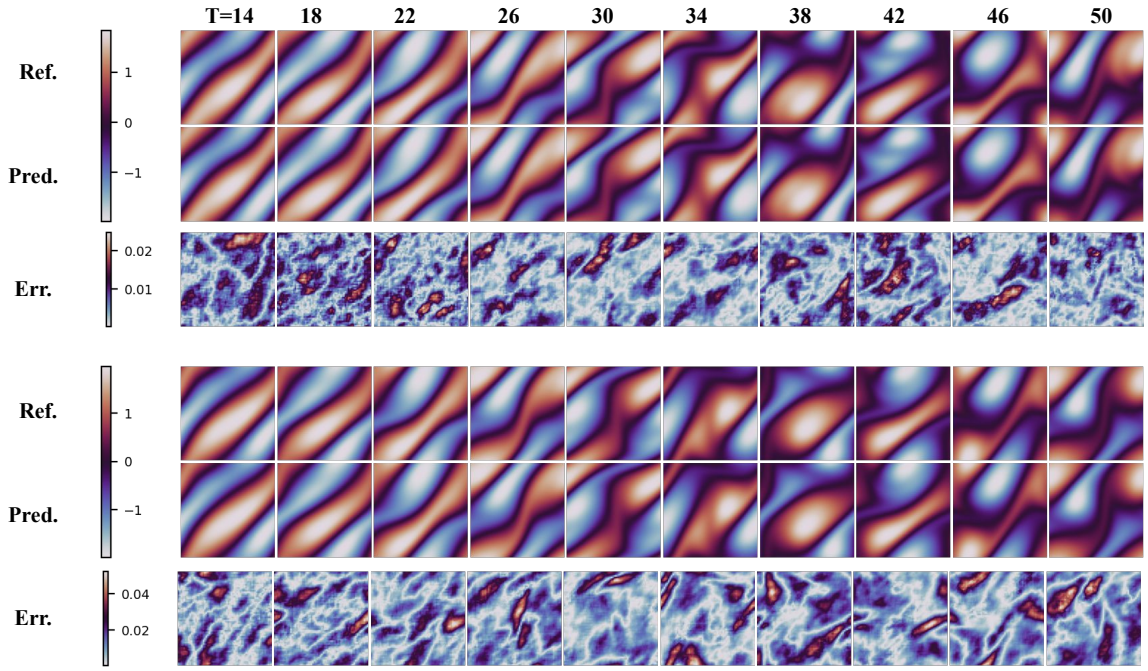


Figure 13: Model's prediction on NS1's test set ($\nu = 1e-3$), with a Reynolds number around 20.

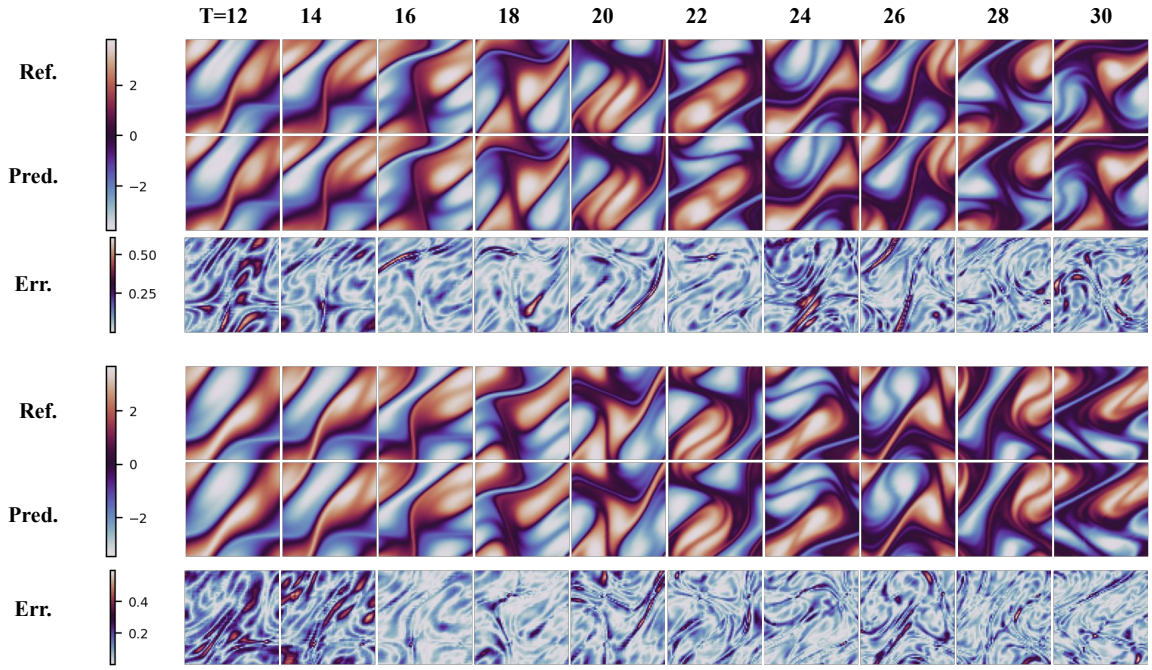


Figure 14: Model's prediction on NS2-full ($\nu = 1e-4$), with a Reynolds number around 200.

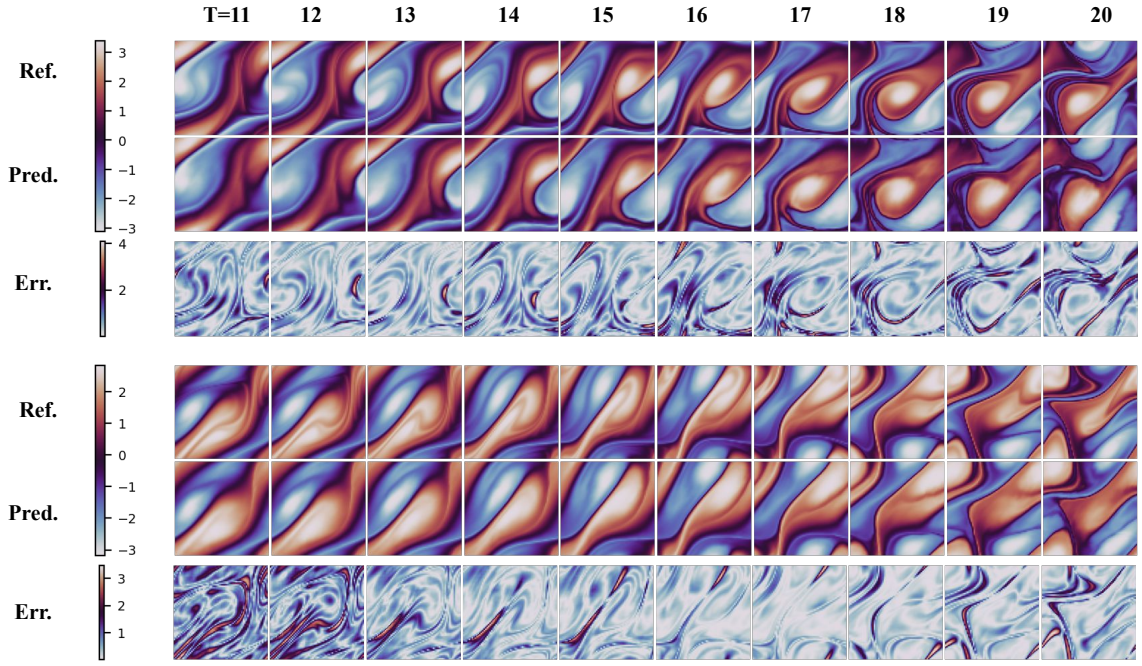
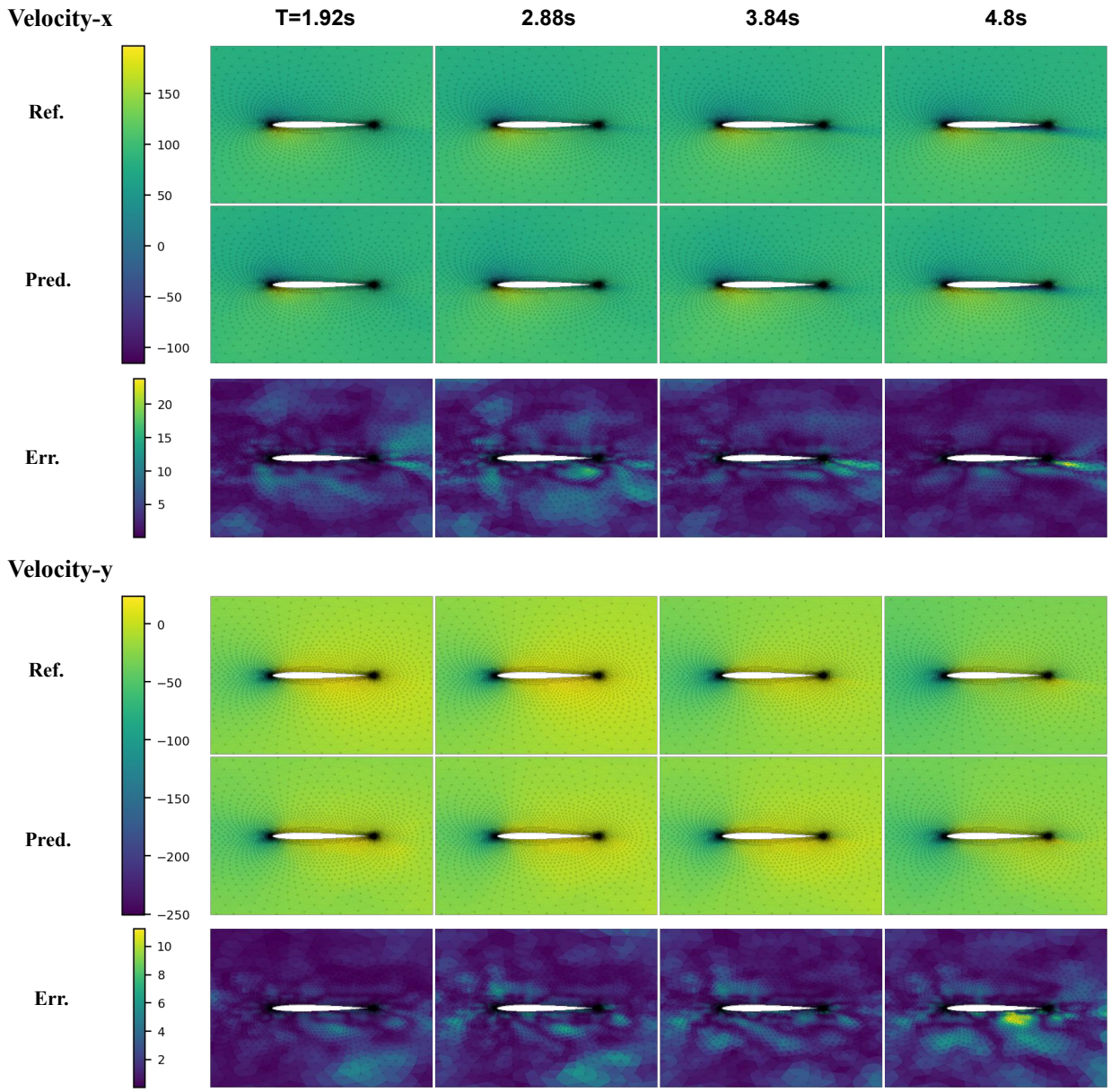
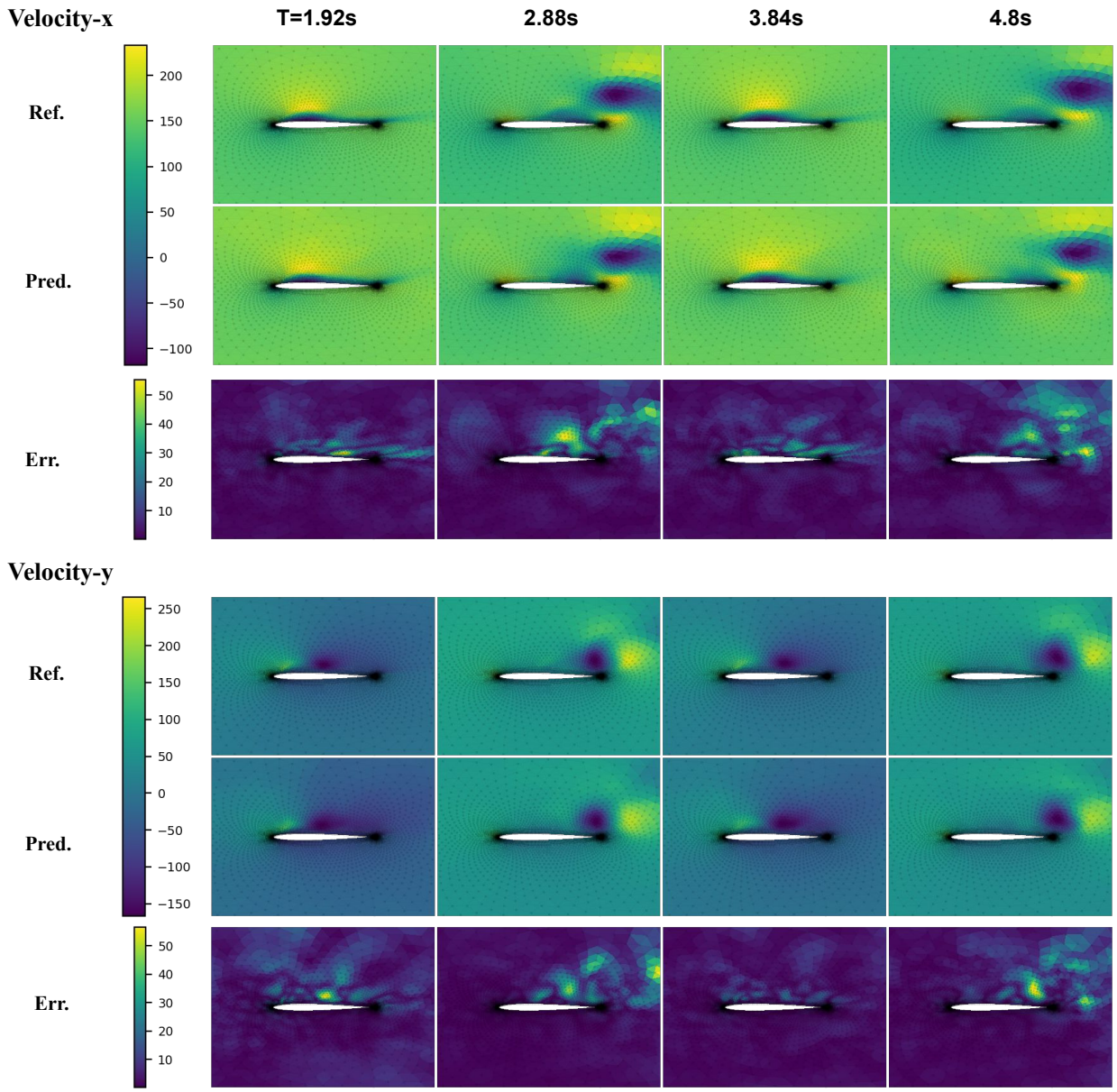


Figure 15: Model's prediction on NS3 ($\nu = 1e-5$), which has a Reynolds number around 2000.


Velocity-x
T=1.92s
2.88s
3.84s
4.8s
Ref.
Pred.
Err.
150
100
50
0
−50
−100
20
15
10
5
Velocity-y
Ref.
Pred.
Err.
0
−50
−100
−150
−200
−250
10
8
6
4
2


Velocity-x
T=1.92s
2.88s
3.84s
4.8s
Ref.
Pred.
Err.
200
150
100
50
0
−50
−100
50
40
30
20
10
Velocity-y
Ref.
Pred.
Err.
250
200
150
100
50
0
−50
−100
−150
50
40
30
20
10

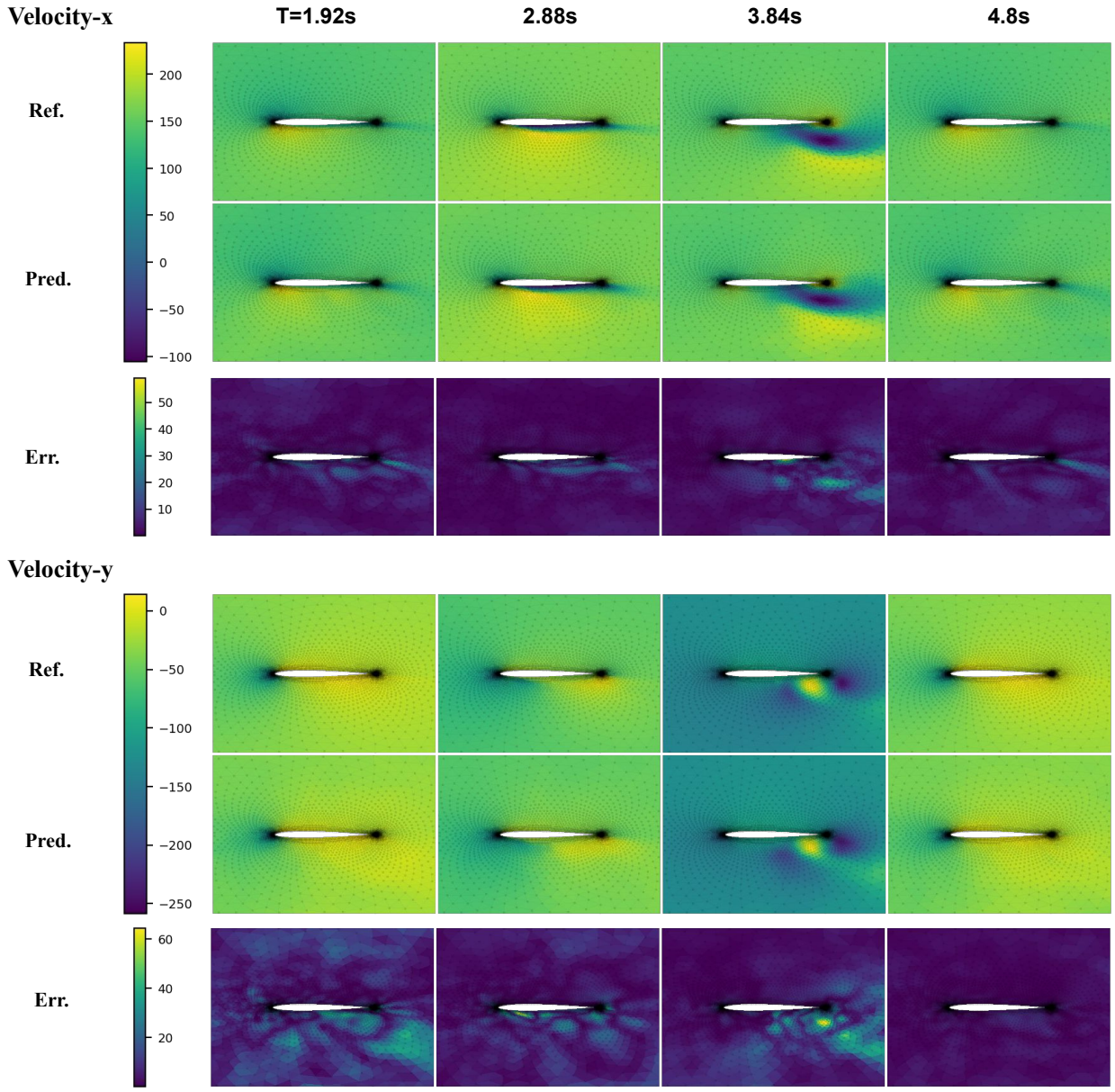


Figure 16: Model's prediction of the compressible flow around airfoil. The test root mean squared error of predicted momentum is **15.477** at plotted region (multiplying velocities with density on each node).

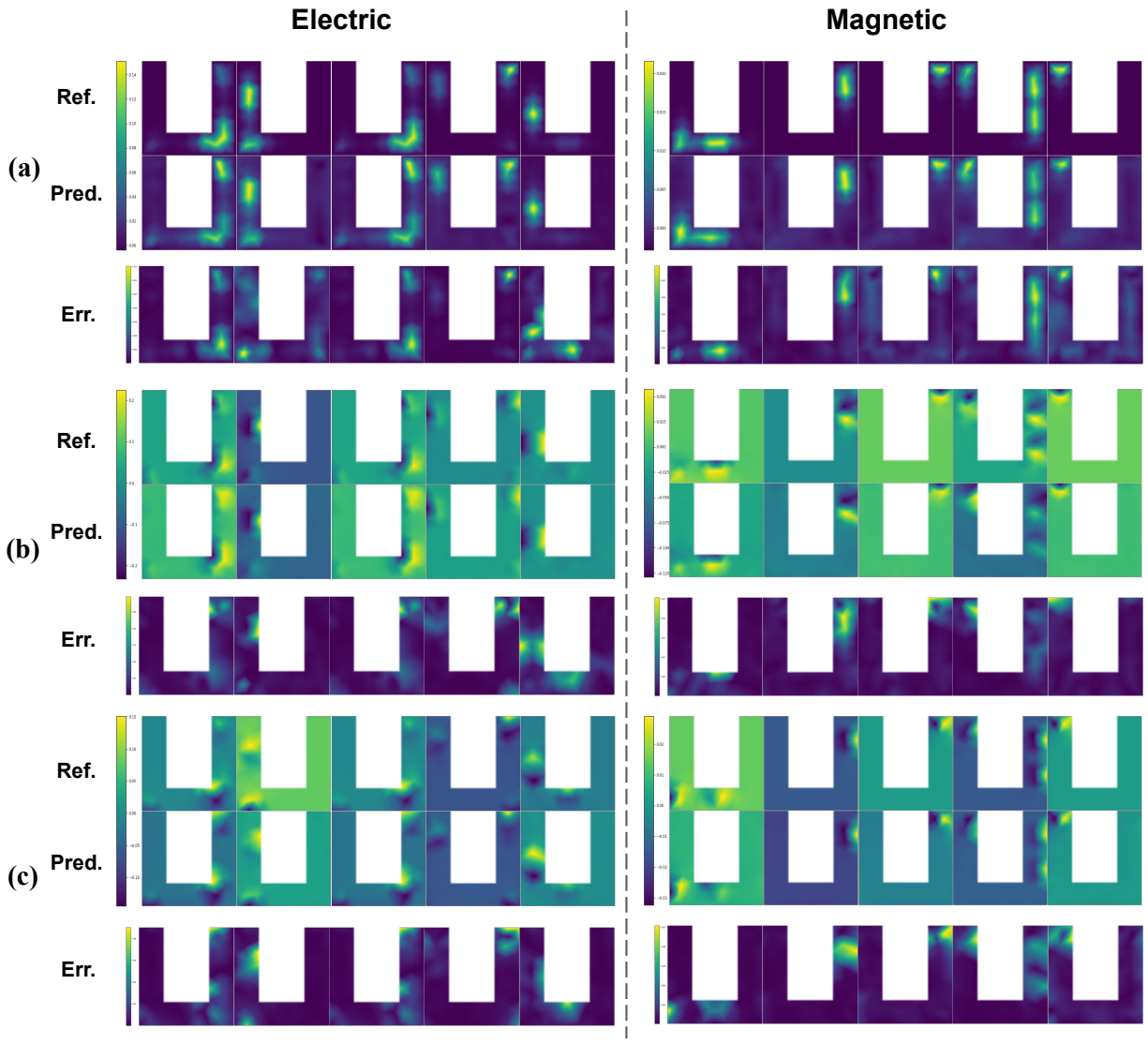


Figure 17: Model's prediction (interpolated from non-uniform grid points): **(a)** potential; **(b)** vector field's x component; **(c)**vector field's y component.