Transformer for Partial Differential Equations’ Operator Learning

Zijie Li  
Department of Mechanical Engineering  
Carnegie Mellon University  
zijieli@andrew.cmu.edu

Kazem Meidani  
Department of Mechanical Engineering  
Carnegie Mellon University  
mmeidani@andrew.cmu.edu

Amir Barati Farimani  
Department of Mechanical Engineering  
Carnegie Mellon University  
barati@cmu.edu

Reviewed on OpenReview: [https://openreview.net/forum?id=EPPqt3uERT](https://openreview.net/forum?id=EPPqt3uERT)

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 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.

1 Code is available at: [https://github.com/BaratiLab/OFormer](https://github.com/BaratiLab/OFormer)
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 et al., 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 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; Cao et al., 2021). As stated in Cao (2021), attention (Vaswani et al., 2017) can be viewed as certain types of learnable integral operators and its inference is equivalent to calculating the integral using input points as the quadrature points, which poses few constraints on the underlying discretization grid. Leveraging this property, we propose a flexible attention-based operator learning framework. Concretely, our main contribution includes: (i) A cross-attention module that enables discretization-invariant query of output function; (ii) A latent time-marching scheme that reduces time-dependent PDE into latent ODE (ordinary differential equation) with a fixed time interval; (iii) A flexible Transformer architecture that can take a varying number of input points and can handle both uniform and non-uniform discretization grids.

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 require 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** The pioneering work DeepONet (Lu et al., 2021a) introduces the first practical realization of universal operator approximation theorem (Chen & Chen, 1995), which can further 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., 2013; Luong et al., 2015) is an operation defined upon three sets of vectors, namely the query vectors \( \{q_i\} \), key vectors \( \{k_j\} \), and value vectors \( \{v_j\} \). Concretely, for each query \( q_i \), the corresponding output \( z_i \) is calculated as a weighted sum (assuming number of vectors in each set is \( n \)):

\[
z_i = \sum_{j=1}^{n} \alpha_{ij} v_j, \quad \alpha_{ij} = \frac{\exp(h(q_i, k_j))}{\sum_{s=1}^{n} \exp(h(q_i, k_s))}.
\]

The most common choice for the weight function \( h(\cdot) \) is the scaled dot-product (Vaswani et al., 2017):

\[
h(q_i, k_j) = (q_i \cdot k_j) / \sqrt{d}, \quad \text{with } d \text{ being the channel of vectors } q_i \text{ and } k_j.
\]

The scaled dot-product can be written in matrix representation as:

\[
Z = \text{softmax}(QK^T / \sqrt{d}) V,
\]

where \( z_i, q_i, k_i, v_i \) are the \( i \)-th row vectors of matrices \( Z, Q, K, V \) respectively. Due to the presence of the softmax, the multiplication \( QK^T \) must be evaluated explicitly, resulting in overall complexity of \( O(n^2d) \), 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, \( q_i/k_i/v_i \), 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, \( V_{ij} \) (also the \( i \)-th element of the \( j \)-th column vector: \( \{v^j\}_i \)) can be viewed as the evaluation of the \( j \)-th basis function on the \( i \)-th grid point \( x_i \), i.e. \( V_{ij} = v_j(x_i) \), 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:

Fourier type: $$(z_i)_j = \frac{1}{n} \sum_{s=1}^{n} (\mathbf{q}_s \cdot \mathbf{k}_s)(\mathbf{v}_j)_s \approx \int_{\Omega} \kappa(x_i, \xi) v_j(\xi) d\xi,$$ (2)

Galerkin type: $$\langle z_i \rangle_i = \sum_{l=1}^{d} \frac{(k_i \cdot v^j)}{n} \langle q^i \rangle_i \approx \sum_{l=1}^{d} \left( \int_{\Omega} (k_l(\xi)v_j(\xi)) d\xi \right) q_l(x_i),$$ (3)

where $\Omega$ denotes the spatial domain that is discretized by $n$ points $\{x_i\}_{i=1}^{n}$, and the dot product $\mathbf{q}_s \cdot \mathbf{k}_s$ in equation 2 approximates the learnable kernel function $\kappa(\cdot, \cdot)$ at $(x_i, x_j)$. The approximation capability of Fourier type attention and Galerkin type attention are studied theoretically at Kovachki et al. (2021), 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 $\mathbf{q}_s, \mathbf{k}_s$ in equation 2 and $\mathbf{k}_i, \mathbf{v}^j$ in equation 3. But if we further exploit the basis function property of these column vectors, it is also reasonable to non-learnable instance normalization (Ulyanov et al., 2016) on each column vector such that their $L_2$ norm equals to one, i.e. $\|\mathbf{v}^j\|_2 = 1$. 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:

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

where $\hat{\cdot}$ 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): $\hat{\mathbf{Z}} = \mathbf{Q}(\mathbf{K}^T \mathbf{V})/n$, 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:

$$z_s(y_j) = \sum_{l=1}^{d} \frac{\sum_{i=1}^{n} k_l(x_i) v_l(x_i)}{n} q_l(y_j).$$ (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): $x \leftarrow \sigma(x \mathbf{W}^{(1)}) \mathbf{W}^{(2)}$, 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 $z_s(y_j)$ 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 $\sum_{i=1}^{n} k_l(x_i) v_l(x_i) / n$ as the output of branch net and $q_l(y_j)$ as the output of 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). Another notable difference is that the number of parameters in the attention layer does not depend on the input’s discretization and it is applicable to samples with varying grids without re-training. 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).
Observations of input function

Input encoding

Coordinates of query locations

Query encoding

Latent encoding on query locations

Figure 1: Attention-based encoder architecture. **Top row:** Input encoder that encodes input function information based on locations \(\{x_i\}_{i=1}^n\) where input function is sampled and the sampled function value. **Bottom row:** Query encoder that encodes the coordinates of query locations \(\{y_i\}_{i=1}^m\) and uses encoded coordinates to aggregate information from input encoding via cross-attention. Rotary positional encodings \(\Theta(\cdot)\) (equation 9) are used in both self-attention and cross-attention.

### 3.2 Model architecture

The proposed model has an Encoder-Decoder architecture that is similar to the original Transformer (Vaswani et al., 2017), where the input is first processed by several self-attention blocks and then attended to the output. However, we only use cross-attention to derive the latent embeddings sampled on query locations and then use recurrent MLPs to propagate dynamics instead of masked attention as in typical Transformer decoder.

**Encoder**

The encoder comprises three components (Figure 1), an input encoder \(\phi^X(\cdot)\) 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 \(\phi^Y(\cdot)\) 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 \(f(0)\) 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):

\[
f(l') = \text{LayerNorm} \left( f(l) + \text{Attn}(f(l)) \right), \quad f(l+1) = \text{LayerNorm} \left( f(l') + \text{FFN}(f(l')) \right),
\]

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; Rahimi & Recht, 2007). The random Fourier projection \(\gamma(\cdot)\) using Gaussian mapping is defined as:

\[
\gamma(Y) = [\cos(2\pi Y B), \sin(2\pi Y B)],
\]

where \(Y = [y_1, y_2, \ldots, y_n]^T, y_i\) is the Cartesian coordinates of the \(i\)-th query point, \(B \in \mathbb{R}^{d_1 \times d_2}\) (with \(d_1\) the dimension of input coordinates and \(d_2\) the output dimension) is a matrix with elements sampled from Gaussian distribution \(N(0, \sigma^2)\) with predefined \(\sigma\). The random Fourier projection \(\gamma(\cdot)\) alleviates the spectral bias of coordinate-based neural networks (Tancik et al., 2020; Mildenhall et al., 2020), which has also been introduced into physics-informed machine learning in Wang et al. (2021c).
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 $z$ 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.

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

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

(8)

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

**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 grid 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 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 $f_{NN}(:, )$ to predict the full trajectory ($\hat{u}_1 = f_{NN}(u_0), \hat{u}_2 = f_{NN}(\hat{u}_1), \ldots$), 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 $z$ from equation [8], we set it as the initial condition $z^0$ and recurrently forward the latent state with a propagator $N(\cdot)$ that predicts the residuals [He et al. 2015] between each step: $z^{t+1} = N(z^t) + z^t$. 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 fixed-interval ODE in the latent space.

After reaching target time step in the latent space, we use another point-wise MLP to decode the latent encodings $z^t(x)$ back to the output function values $u(x,t)$.

**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 encode 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 $q_i \in \mathbb{R}^{d\times1}$, the RoPE $\psi(\cdot)$ is defined as:

$$
\psi(q_i, x_i) = \Theta(x_i) q_i,
$$

where: $\Theta(x_i) = \text{Diag}(R_1, R_2, \ldots, R_{d/2})$, \( \text{Diag}(\cdot) \) denotes stacking sub-matrices along the diagonal, $\lambda$ is wavelength of the spatial domain (e.g. $\lambda = 2048$ for a spatial domain discretized by 2048 equi-distant points), $\theta_l$ 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 $q_i$ with a key vector $k_j$: $\psi(q_i, x_i) \cdot \psi(k_j, x_j) = (\Theta(x_i) q_i) \cdot (\Theta(x_j) k_j) = q_i^T \Theta(x_i - x_j) k_j$. 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 $q_i$ with coordinate $x_i = (\alpha_i, \beta_i)$ and key $k_j$ with $x_j = (\alpha_j, \beta_j)$ write as:

$$
\psi(q_i, x_i) \cdot \psi(k_j, x_j) = \\
\psi_1 \left( (q_i)_{d/2}, \alpha_i \right) \cdot \psi_1 \left( (k_j)_{d/2}, \alpha_j \right) + \psi_2 \left( (q_i)_{d/2}, \beta_i \right) \cdot \psi_2 \left( (k_j)_{d/2}, \beta_j \right),
$$

where $(q_i)_{d/2}$ denotes the first $d/2$ dimensions of vector $q_i$ and $(q_i)_{d/2}$ denotes the rest $d/2$ dimensions, similarly for $k_j$, 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 $G_\theta : \mathcal{A} \rightarrow \mathcal{U}$. The model parameter $\theta$ is optimized by minimizing the empirical loss $\mathbb{E}_{a \sim \mu} [\mathcal{L}(G_\theta(a), u)]$, where $\mu$ 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 $(a^{(j)}, u_D^{(j)}(a))_{j=1}^N$ on discrete grid points $D(x) : \{x_i\}_{i=1}^m, D(y) : \{y_i\}_{i=1}^n$, and choose $\mathcal{L}(\cdot)$ as relative $L_2$ norm: $\mathbb{E}_{a \sim \mu} \sum_{j=1}^B \sum_{i=1}^m ||a^{(j)} - u_D^{(j)}(a)||_2^2$, given $B$ samples with $a^{(j)}$ 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 $\Omega$ and can vary across different training samples $(u_D^{(j)})_a, u_D^{(j)}(a))$. 

---

7
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 studies of hyperparameter choices are provided 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 incompressible 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):

\begin{align}
\text{Burgers':} & \quad \mathcal{G} : u(\cdot, t)|_{t=0} \mapsto u(\cdot, t)|_{t=1}, \\
\text{Darcy flow:} & \quad \mathcal{G} : a \mapsto u, \\
\text{Navier-Stokes:} & \quad \mathcal{G} : u(\cdot, t)|_{t \in [0, 10]} \mapsto u(\cdot, t)|_{t \in (10, T]}.
\end{align}

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 $\nu$ 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 and source-terms. 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:

\begin{align}
\text{Poisson:} & \quad \mathcal{G} : f \mapsto u, \quad u = 0 \text{ on } \partial \Omega, \\
\text{Airfoil:} & \quad \mathcal{G} : u(\cdot, t)|_{t \in [0, 0.576]} \mapsto u(\cdot, t)|_{t \in [0.576, 4.800]}.
\end{align}
where $\Omega$ is the solution spatial domain and $f$ is the source term of the Poisson equation. For Poisson equation we predict the field vector in addition to the scalar potential. 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 for each problem. Details of baseline models' implementation and comparison of calculation time can be found in the Appendix B.

#### Data settings

<table>
<thead>
<tr>
<th>Case</th>
<th>$\nu, T$</th>
<th>FNO-2D</th>
<th>FNO-3D</th>
<th>MWT</th>
<th>G.T.*</th>
<th>OFormer</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS1</td>
<td>$1 \times 10^{-3}, 50$</td>
<td>0.0128</td>
<td>0.0086</td>
<td><strong>0.0062</strong></td>
<td>0.0098</td>
<td>0.0104 ± 0.0003</td>
</tr>
<tr>
<td>NS2-part</td>
<td>$1 \times 10^{-4}, 30$</td>
<td>0.1559</td>
<td>0.1918</td>
<td>0.1518</td>
<td><strong>0.1399</strong></td>
<td>0.1755 ± 0.0056</td>
</tr>
<tr>
<td>NS2-full</td>
<td>$1 \times 10^{-4}, 30$</td>
<td>0.0834</td>
<td>0.0820</td>
<td>0.0667</td>
<td>0.0792</td>
<td><strong>0.0645 ± 0.0011</strong></td>
</tr>
<tr>
<td>NS3</td>
<td>$1 \times 10^{-5}, 20$</td>
<td>0.1556</td>
<td>0.1893</td>
<td>0.1541</td>
<td><strong>0.1340</strong></td>
<td>0.1705 ± 0.0007</td>
</tr>
<tr>
<td>NS-mix*</td>
<td>[0.4,1] $\times 10^{-4}$, 30</td>
<td>0.1650</td>
<td>0.1654</td>
<td><strong>0.1267</strong></td>
<td>0.1462</td>
<td>0.1402 ± 0.0016</td>
</tr>
</tbody>
</table>

| # 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 $64 \times 64$ 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.

#### Table 2: Benchmark on 1D Burgers’ equation with different resolution. FNO uses ReLU whereas FNO+ uses GELU. Both FNO variants remove normalization compared to the original paper.

<table>
<thead>
<tr>
<th>Res.</th>
<th>FNO*</th>
<th>FNO+</th>
<th>MWT</th>
<th>G.T.</th>
<th>OFormer</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>3.15</td>
<td>0.72</td>
<td>1.85</td>
<td>1.14</td>
<td>1.42 ± 0.03</td>
</tr>
<tr>
<td>2048</td>
<td>3.07</td>
<td>0.72</td>
<td>1.86</td>
<td>1.12</td>
<td>1.30 ± 0.07</td>
</tr>
<tr>
<td>8192</td>
<td>2.76</td>
<td>0.70</td>
<td>1.78</td>
<td>1.07</td>
<td>1.54 ± 0.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Res.</th>
<th>FNO*</th>
<th>FNO+</th>
<th>G.T.</th>
<th>OFormer</th>
</tr>
</thead>
<tbody>
<tr>
<td>141</td>
<td>1.09</td>
<td><strong>0.84</strong></td>
<td>1.26 ± 0.03</td>
<td></td>
</tr>
<tr>
<td>211</td>
<td>1.09</td>
<td><strong>0.84</strong></td>
<td>1.28 ± 0.02</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Benchmark on 1D Burgers’ equation 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.

#### 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. (The temporal error trend can be found in Appendix C. Contour plot of the prediction and corresponding error can be found in Appendix E.)

---

2The electric field $\mathbf{E}$ is defined as the negative gradient of electric potential ($\phi$): $\mathbf{E} = -\nabla \phi$. For magnetic field $\mathbf{B}$, it is defined as the curl of magnetic potential $\mathbf{A}$: $\mathbf{B} = \text{rot} \mathbf{A}$. In this problem $\mathbf{A}_x = \mathbf{A}_y = 0$, and therefore magnetic potential reduces to: $B_x = \partial A_z / \partial y, B_y = -\partial A_z / \partial x$. In the model we use a separate head to predict these fields.
Figure 3: Visualization of training/testing samples in 2D Poisson problem. Testing set contains new geometries that never appeared in the training set.

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.

<table>
<thead>
<tr>
<th>Mesh aug.</th>
<th>Model</th>
<th>Mean squared error ($\times 10^{-3}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Electric</td>
<td>Magnetic</td>
</tr>
<tr>
<td>No</td>
<td>OFormer</td>
<td>0.174 ± 0.083</td>
</tr>
<tr>
<td></td>
<td>GNN</td>
<td>0.327</td>
</tr>
<tr>
<td>Yes</td>
<td>OFormer</td>
<td>0.019 ± 0.004</td>
</tr>
<tr>
<td></td>
<td>GNN</td>
<td>0.006</td>
</tr>
</tbody>
</table>

Application to irregular grids The comparison between OFormer and GNN baseline on the 2D Poisson problem is 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 in 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 in 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 $15.469 \pm 0.074$ (relative error: $8.17 \pm 0.06\%$).

Another important property of OFormer is that it can handle variable number of input/query points with the same set of weights. Here we showcase an application of such property. As shown in Figure 5, the model can still predict reasonably accurate results on a full-resolution output grid based on sparse input. The input is generated by randomly sampling 25% of the grid points in the spatial domain, which is 150% of the maximum dropping ratio during training.3 (More quantitative results on varying sampling grids can be found in Appendix C.)

3During training, we randomly drop some of the input points with a ratio of $r \sim U([0,0.5])$, but still query at full resolution and train on full resolution.
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 \( \nu \) 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 \( f^{(k)}_{\text{avg}} \) of \( k \)-th sample is obtained by applying average pooling on the encodings \( f^{(k)}_{\text{avg}} = \text{Avg}(\{f^{(k)}_{i}\}_{i=1}^{n}) \) 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. \( \nu \) 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 the \( \nu \) on testing samples. Figure 6(b) shows 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 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 \( \nu \) of testing samples.
4.3 Ablation study

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 visualized sequence is randomly chosen. Error is measured by absolute error.

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 $\|v^i\|_2 = 1$, performs better than LayerNorm, further facilitating the bases interpretation of two linear attention mechanisms. Scaling up the model’s size (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 computationally 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.

6 Acknowledgments

This work is supported by the start-up fund from the Department of Mechanical Engineering, Carnegie Mellon University, United States.

References

Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. Archive of Numerical Software, 3(100), 2015.


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($d_1$, $d_2$) 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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Input encoder top</th>
<th>SA hidden dim</th>
<th>SA FFN</th>
<th># SA Block</th>
<th>Input encoder bottom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D Burgers</td>
<td>[2, 96]</td>
<td>96×1</td>
<td>[96, 192, 96]</td>
<td>4</td>
<td>[96, 96]</td>
</tr>
<tr>
<td>2D Darcy flow</td>
<td>[3, 96]</td>
<td>96×4</td>
<td>[96, 192, 96]</td>
<td>4</td>
<td>[96, 256]</td>
</tr>
<tr>
<td>2D Navier-Stokes</td>
<td>[12, 128]</td>
<td>128×1</td>
<td>[128, 256, 128]</td>
<td>5</td>
<td>[128, 192]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Query encoder top</th>
<th>CA hidden dim</th>
<th>CA FFN</th>
<th>Query encoder bottom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D Burgers</td>
<td>[1, 96, 96]</td>
<td>96×8</td>
<td>[96, 192, 96]</td>
<td>-</td>
</tr>
<tr>
<td>2D Darcy flow</td>
<td>[2, 256, 256]</td>
<td>256×4</td>
<td>[256, 512, 256]</td>
<td>-</td>
</tr>
<tr>
<td>2D Navier-Stokes</td>
<td>[2, 192, 192]</td>
<td>192×4</td>
<td>[192, 384, 196]</td>
<td>[192, 384]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Propagator</th>
<th>Decoder</th>
<th>Total # of params (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D Burgers</td>
<td>[96, 96, 96, 96]×3</td>
<td>[96, 48, 1]</td>
<td>0.67</td>
</tr>
<tr>
<td>2D Darcy flow</td>
<td>-</td>
<td>[256, 128, 64, 1]</td>
<td>2.51</td>
</tr>
<tr>
<td>2D Navier-Stokes</td>
<td>[384, 384, 384, 384]</td>
<td>[384, 192, 96, 1]</td>
<td>1.85</td>
</tr>
</tbody>
</table>

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 we train model for 50k iterations using a batch size of 10.
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 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) [Li et al., 2021a], Multiwavelet Operator (MWT) [Gupta et al., 2021] and Galerkin/Fourier Transformer (G.T./F.T.) [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.

<table>
<thead>
<tr>
<th>Model</th>
<th>Iters / sec</th>
<th>Memory (GB)</th>
<th># of params (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNO-2D</td>
<td>9.71</td>
<td>0.33</td>
<td>2.37</td>
</tr>
<tr>
<td>FNO-3D</td>
<td>10.74</td>
<td>0.42</td>
<td>6.56</td>
</tr>
<tr>
<td>MWT</td>
<td>1.67</td>
<td>10.95</td>
<td>179.18</td>
</tr>
<tr>
<td>G.T.</td>
<td>1.79</td>
<td>16.65</td>
<td>1.56</td>
</tr>
<tr>
<td>OFomer</td>
<td>1.89</td>
<td>15.93</td>
<td>1.85</td>
</tr>
</tbody>
</table>

Table 11: Training computational cost of different models.

<table>
<thead>
<tr>
<th>Model</th>
<th>OFomer</th>
<th>G.T.</th>
<th>FNO-2D</th>
<th>FNO-3D</th>
<th>MWT</th>
<th>Pseudo-spectral</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fine</td>
</tr>
<tr>
<td>Time (sec)</td>
<td>2.32</td>
<td>3.22</td>
<td>0.58</td>
<td>0.50</td>
<td>2.21</td>
<td>5735.74</td>
</tr>
</tbody>
</table>

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 $256 \times 256$ while coarse grid has a resolution of $64 \times 64$.

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 $r \sim U([0,0.5])$) 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.
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.

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

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 $64 \times 64$; random drop indicates that some are the input grid points are randomly dropped during training, while output grid is still fixed at $64 \times 64$. Error is measured by relative $L_2$ norm. The resolution of Burgers’ equation is 2048.

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 $L_2$ norm on comparing full-resolution ($64 \times 64$) prediction with ground truth.

<table>
<thead>
<tr>
<th>Training strategy</th>
<th>All</th>
<th>95%</th>
<th>75%</th>
<th>50%</th>
<th>25%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random drop</td>
<td>0.0672</td>
<td>0.0687</td>
<td>0.0772</td>
<td>0.0939</td>
<td>0.1294</td>
</tr>
<tr>
<td>No random drop</td>
<td>0.0646</td>
<td>0.0795</td>
<td>0.1293</td>
<td>0.2065</td>
<td>0.3774</td>
</tr>
</tbody>
</table>

Influence of initialization  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).

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

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

$$W(q) = \sigma B + \delta I,$$

(17)
where $\sigma$ 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: $\mathbf{z}' = \mathbf{z} + \text{Attn}(\mathbf{z})$ 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 10: Performance of OFormer on NS2 when trained with different amount of data.](image)

(a) Basis learned with a MLP that does not contain any transformation on the top layer.

![Figure 11: Visualization of bases learned (the output of coordinate-based network in the decoder part) on 2D Darcy flow. The visualized bases are randomly chosen.](image)

(b) Basis learned with a MLP that has RFF on the top layer.

**Influence of Random Fourier Feature** We study the influence of Random Fourier Features (RFF) [Tancik et al., 2020; Rahimi & Recht, 2007] on 3 different problems - 1D Burgers, 2D Darcy flow and Navier-Stokes. As shown in Figure 12 applying RFF to the input coordinates boosts the performance on all
three problems. We hypothesize the main reason is the original MLP tends to learn a set of bases with lower frequency while RFF broadens the spectrum of these earned bases (Tancik et al., 2020). As an example, Figure 11 reveals that without RFF, learned bases have a relatively smoother pattern than those with RFF.

(a) Performance on Burgers’ equation
(b) Performance on Darcy flow with 141 × 141 resolution.
(c) Performance on NS2-full.

Figure 12: Ablation on the influence of RFF.

Temporal trend of error  We investigate the temporal trend of OFormer’s prediction error on time-dependent system (Figure 13) by computing the mean error of model’s prediction across all test sequences at a particular time step. In general, we observe that the error will accumulate exponentially as time grew. The compressible flow’s error (Figure 13f) exhibits some levels of periodicity due to the intrinsic periodic property of the flow. Increasing the number of training samples alleviate the error, but does not change the
exponential trend. How to alleviate the compounding prediction error for neural operator/PDE solver can be an interesting and important future venue.

D Dataset details

D.1 Data generated on equi-spaced mesh

Following datasets 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:

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

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

\[
\begin{align*}
-\nabla \cdot (a(x) \nabla u(x)) &= f(x), \quad x \in (0, 1)^2, \\
u_0(x) &= 0, \quad x \in \partial(0, 1)^2,
\end{align*}
\] (19)

where \(f(x)\) is the forcing function (set to constant 1) and we aim to learn the operator mapping diffusion coefficient \(a(\cdot)\) to the solution \(u(\cdot)\), i.e., \(\mathcal{G} : a \mapsto u\). 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 \times 421 resolution grid.

\footnote{Dataset and numerical solver courtesy: \url{https://github.com/zongyi-li/fourier_neural_operator}}
Navier-Stokes Equation  The vorticity form of 2D Navier Stokes equation for the viscous, incompressible fluid reads as:
\[
\begin{aligned}
\partial_t \omega(x,t) + \mathbf{u}(x,t) \cdot \nabla \omega(x,t) &= \nu \Delta \omega(x,t) + f(x), \quad x \in (0,1)^2, t \in (0,T], \\
\nabla \cdot \mathbf{u}(x,t) &= 0, \quad x \in (0,1)^2, t \in (0,T], \\
\omega(x,0) &= \omega_0(x), \quad x \in (0,1)^2, 
\end{aligned}
\]  
(20)

where \( \omega \) is vorticity, \( \mathbf{u} \) represents velocity field, and \( \nu \) is the viscosity, source term is set as: \( f(x) = 0.1 \left( \sin(2\pi(x_1+x_2)) + \cos(2\pi(x_1+x_2)) \right) \), 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., \( \mathcal{G} : \omega(\cdot,t)_{|t \in [0,10]} \mapsto \omega(\cdot,t)_{|t \in [10,T]} \). Generally, lower viscosity coefficients \( \nu \) 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 \times 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 vorticity in the Fourier space, dealised 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:
\[
\begin{aligned}
\text{Electric:} \quad & -\nabla^2 U(x) = \frac{\rho(x)}{\epsilon(x)}, \quad x \in \Omega \\
\text{Magnetic:} \quad & -\nabla^2 A_z(x) = \mu(x) I_z(x), \quad x \in \Omega
\end{aligned}
\]  
(21)

where \( \rho \) is the charge density, \( \epsilon \) is the permittivity of the material, \( U \) is the electric potential, \( \mu \) 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:
\[ \mathcal{G} : \{ \rho, \epsilon \} \mapsto U \ (U(x) = c \text{ for } x \in \partial \Omega) \text{ for electrostatics and } \mathcal{G} : \{ I_z, \mu \} \mapsto A_z \ (A_z(x) = c \text{ for } x \in \partial \Omega) \text{ for magnetostatics}. \]

The data is generated by solving the above equation using finite element method implemented with FEniCS library [Alnæs et al., 2015]. The element used is a quadratic triangle element. We use the pre-generated data from Lötzsch et al. (2022) for experiment.

Euler equation of compressible flow  The Euler equation of compressible flow takes the form:
\[
\begin{aligned}
\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) &= f_1, \\
\partial_t (\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + \rho \mathbb{I}) &= f_2, \\
\partial_t (\rho E) + \nabla \cdot (\rho E \mathbf{u} + \rho \mathbf{u} \otimes \mathbf{u}) &= f_3, \\
\rho &= \rho(x,t), \mathbf{u} := \mathbf{u}(x,t), p := p(x,t), \\
&\quad x \in \Omega, t \in [0,T],
\end{aligned}
\]  
(22)

where \( \rho \) 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 \( p = (\gamma - 1)\rho [E - 0.5(\mathbf{u} \cdot \mathbf{u})] \)), and \( f_1, f_2, f_3 \) are generic source terms. No-penetration condition is imposed at the airfoil. The solution operator of interest is: \( \mathcal{G} : \{ \mathbf{u}, \rho, p \}_{|x \in \Omega, t \in \{0,0.576\}} \mapsto \{ \mathbf{u}, \rho, p \}_{|x \in \Omega, t \in \{0,576,1.800\}} \).

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\footnote{Dataset and numerical solver courtesy: https://github.com/nerantix-momentum/gnn-bvp-solver} 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 14: 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.
Figure 15: Model’s prediction on 2D Darcy flow, samples are randomly selected from test set.

Figure 16: Model’s prediction on NS1’s test set ($\nu = 1e^{-3}$), with a Reynolds number around 20.
Figure 17: Model’s prediction on NS2-full ($\nu = 1e^{-4}$), with a Reynolds number around 200.

Figure 18: Model’s prediction on NS3 ($\nu = 1e^{-5}$), which has a Reynolds number around 2000.
Velocity-x

Ref.

Pred.

Err.

Velocity-y

Ref.

Pred.

Err.

T=1.92s 2.88s 3.84s 4.8s
Figure 19: 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 20: Model's prediction (interpolated from non-uniform grid points): (a) potential; (b) vector field’s x component; (c) vector field’s y component.