Computer Methods in Applied Mechanics and Engineering 426 (2024) 116983

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Multi-scale time-stepping of Partial Differential Equations with transformers

AmirPouya Hemmasian [a], Amir Barati Farimani [a,b,*]

[a] Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, 15213, PA, USA
[b] Machine Learning Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, 15213, PA, USA

ARTICLE INFO

Keywords:
Machine learning
Transformers
Numerical time-stepping
Partial differential equations

ABSTRACT

Developing fast surrogates for Partial Differential Equations (PDEs) will accelerate design and optimization in almost all scientific and engineering applications. Neural networks have been receiving ever-increasing attention and demonstrated remarkable success in computational modeling of PDEs, however; their prediction accuracy is not at the level of full deployment. In this work, we utilize the transformer architecture, the backbone of numerous state-of-the-art AI models, to learn the dynamics of physical systems as the mixing of spatial patterns learned by a convolutional autoencoder. Moreover, we incorporate the idea of multi-scale hierarchical time-stepping to increase the prediction speed and decrease accumulated error over time. Our model achieves similar or better results in predicting the time-evolution of Navier–Stokes equations compared to the powerful Fourier Neural Operator (FNO) and two transformer-based neural operators OFormer and Galerkin Transformer. The code and data are available on https://github.com/BaratiLab/MST_PDE.

1. Introduction

Partial Differential Equations (PDEs) govern the dynamics of continuous physical systems in science and engineering. To solve them numerically, functions are represented as finite sets of numbers, typically through methods like finite difference/element/volume or spectral methods. While these methods offer stability and accuracy, they can be computationally intensive, especially for complex systems. In recent years, Machine Learning and Deep Learning have emerged as alternative approaches for scientific computation and PDE modeling due to their remarkable achievements in various fields [1–3].

The art of deep learning is to design or utilize the network architecture best suited for the data or the task at hand. Convolutional Neural Networks (CNNs) have been utilized for modal analysis and model reduction of physical systems, as well as PDE solving [4–7]. While CNNs have been shown to have adequate sample efficiency and accuracy, they are limited to data represented on an equispaced mesh. Recurrent Neural Networks (RNNs) are another class of networks that are usually used to model dynamic systems and time-dependent differential equations [8–10]. Despite the success of RNNs in many applications, there are challenges with training that limit their utilization in some problems.

The key component of this work is a novel architecture that has been achieving remarkable success in numerous applications, and that is the transformer architecture [11–14]. The transformer was first introduced as a mechanism for learning from a global context without suffering from the shortcomings of sequential models like RNNs. We will briefly review the common data-driven frameworks in PDE modeling, followed by a more focused elaboration on the applications of the transformer architecture in the context of PDE modeling.

* Corresponding author.
E-mail address: barati@cmu.edu (A. Barati Farimani).

https://doi.org/10.1016/j.cma.2024.116983
Received 4 December 2023; Received in revised form 3 April 2024; Accepted 4 April 2024
Available online 13 April 2024

For the sake of generality and consistency throughout the paper, we assume a PDE of the form shown in Eq. (1) where $\mathbf{x} = [x_1, x_2, \dots, x_d]^T \in \mathbb{R}^d$ represents a d-dimensional spatial domain and $t \in \mathbb{R}$ represents time.

$$u_t = N(\mathbf{x}, u, u_{x_1}, u_{x_2}, \dots, u_{x_1 x_1}, u_{x_1 x_2}, u_{x_2 x_2}, \dots) \tag{1}$$

Reduced-order models. The Reduced-Order Modeling (ROM) framework is based on the separation of variables space and time [1] and is a classic method to obtain low-cost surrogate models of PDEs, especially fluid flows. A ROM assumes a solution form shown in Eq. (2) where $\cdot$ is the inner product and $a, \psi \in \mathbb{R}^K$ are the temporal and spatial components respectively. The spatial component ψ consists of basis functions ψ_k, also known as spatial modes, that are chosen to preserve the most spatial information and variance of the system and manifest as dominant patterns. Eventually, the temporal component $a(t)$ is obtained using analytical, numerical, or data-driven algorithms. In data-driven ROMs, sample solution functions are represented in a discretized temporal and spatial domain. Proper Orthogonal Decomposition is a classic algorithm to obtain the spatial component from the eigenvectors of the data matrix [15,16]. Different machine learning algorithms and deep learning architectures can be utilized to model either the spatial or the temporal component of the ROM [5,8–10,17–19].

$$u(\mathbf{x}, t) = a(t) \cdot \psi(\mathbf{x}) = \sum_{k=1}^{K} a_k(t)\psi_k(\mathbf{x}) \tag{2}$$

Latent space time-stepping. This framework is a generalized version of ROM where the spatial information is not simply compressed into a few modes or bases, but possibly a richer representation learned by a neural network [20–23]. It usually consists of three main components namely the encoder, decoder, and dynamical model, denoted as $P, Q, D^{\Delta t}$ respectively such that $z_t = P(s_t)$, $Q(z_t) = \tilde{s}_t$ and $z_{t+\Delta t} = D^{\Delta t}(z_t)$. Here, s_t, $\hat{s}_t$, and z_t are the state, reconstructed state, and latent/encoded state of the system at time t respectively, and z_t is a generalization of $a(t)$ in ROMs.

Direct time-stepping. If P and Q from the previous framework are identity functions, only a dynamical model is to be learned to simulate the time evolution of the system in its original representation. Since the state representation is maintained, the choice of architecture suited for the problem at hand may be more transparent, i.e. using CNNs for regular meshes and GNNs for irregular meshes [24–30]. These methods usually can achieve high accuracy and learn the fine changes and patterns as well due to their working with the original representation rather than a latent or reduced one, but they come with a high computational cost for both training and testing compared to the previous frameworks.

Solution function approximation. Another recently popular approach is to approximate the solution function by a parameterized function of a certain form like a neural network such that $u(\mathbf{x}, t) \simeq f_\phi(\mathbf{x}, t)$. The model parameters can then be obtained by minimizing the approximation error consisting of data-driven loss and physics-informed loss, giving them the name physics-informed neural networks (PINN) [31–34]. These models are completely mesh-agnostic and can even approximate the solution function solely from the physics-informed loss without any sampled simulation data [35]. However, a new model has to be trained for each instance of the PDE with new parameters and initial and boundary conditions, and the calculation of high-order derivatives in high-dimensional domains can be prohibitive.

Neural operators. While previous frameworks treat the discretized representation of functions as members of Euclidean spaces, neural operators conserve their identity and properties as functions, enabling them to work with arbitrary discretizations. Based on the universal operator approximation theorem for neural networks [36], Deep Operator Networks (DeepONets) realized the practical application of neural networks in operator learning [37]. Another class of neural operators uses an analogous architecture to fully connected networks by approximating nonlinear operators with a composite of several layers of kernel integral operators and nonlinear activation functions. The kernel can be approximated with a neural network [38], the column space of attention weights in a transformer [39,40], wavelet bases [41], or Fourier bases [42–44].

Transformer and PDE simulation. Following the groundbreaking success of the transformer model since its introduction [11–14,45], researchers have utilized its architecture and the attention mechanism to model and predict physical systems [46] in frameworks such as ROM [47,48] and neural operators [39,40,49–56]. As pointed out by the aforementioned works, the attention mechanism can be viewed as a discretized approximation of a kernel integral operator, making it a suitable tool in operator learning. However, the computational cost can be hindering for high-resolution meshes. The problem of long input sequences and the high computational cost of the attention mechanism has been the focus of many works and is usually mitigated by modifying the attention mechanism to have linear or quasi-linear cost in terms of the input length [57–61].

In the context of solving PDEs, Cao [39] introduced the Galerkin Transformer which omits the softmax function in the attention and changes the order of calculations to have a linear cost in the input length, but the cost is still prohibitive in high-dimensional and high-resolution problems. Li et al. [55] applied the attention separately across different axes, reducing the total cost. Unlike such works that focus on modifying the algorithm and mechanism of the attention, ROMER [48] approaches this issue from a different perspective, investigating how compressing the data representation itself can be a path to an efficient and accurate model based on attention. This work provides this framework with essential enhancements that enable it to achieve remarkable performance, similar or superior to state-of-the-art models like neural operators. The introduced improvements consist of finite backpropagation in time, multi-scale time-stepping, and a more intuitive positional encoding for the transformer architecture, which will be explained in detail.

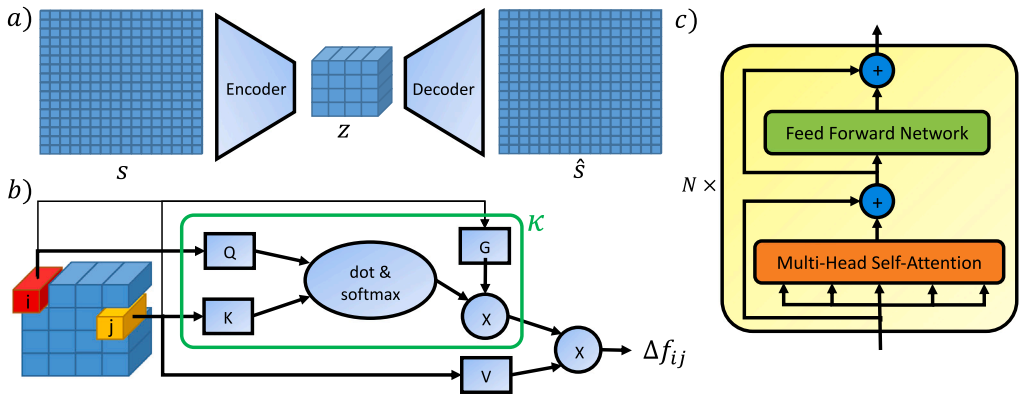

Fig. 1. (a) The convolutional autoencoder at the stem of our model. (b) The attention mechanism of our model with disentangled incorporation of positional information and feature values. (c) The transformer model consisting of N transformer layers. The feed-forward network is implemented using 1×1 convolution layers to apply an identical fully connected network to all elements.

2. Methodology

This section explains the architecture of our model step by step. First, the convolutional autoencoder (CAE) learns a coarse-meshed feature representation of the system called the encoding. Next, the attention mechanism which is the backbone of the transformer architecture is introduced and its connection to the kernel integral operator is elucidated. After that, the multi-head self-attention layer and the transformer architecture are presented which are used to learn the dynamics in the encoding space. Finally, details and strategies for the training process are provided.

2.1. Convolutional autoencoder

We leverage the locality and multiscale nature of the spatial patterns commonly observed in physical systems like fluid flows and reduce the spatial dimension of the state representation using a CAE. A fully convolutional network learns a feature representation that conserves the order of patterns while compressing the spatial dimensions. Since the attention mechanism has a quadratic cost in input length, this greatly reduces the cost for the following components of the model based on attention.

Another important motivation to put the CAE at the stem of our model is to process information and learn features in the spatial dimensions first, rather than doing so in the temporal dimension. We believe this to be more intuitive considering the nature of a typical time-dependent PDE of the form in Eq. (1). This makes it more similar to classic numerical algorithms since they usually estimate spatial gradients and the right hand of this equation using discretized approximations first, then use them to predict the values at the next time-step. Likewise, our model first learns spatial features and patterns from a single time frame and models the dynamics as exchanges among them.

The CAE consists of the encoder and the decoder, which can be viewed analogously to the lifting and projecting layer (P and Q) at the beginning and end of neural operators like FNO [42]. While in neural operators these layers usually operate on a concatenation of consecutive snapshots and learn temporal features only, our model works with a single snapshot in time and learns spatial features. Without the loss of generality, we present the data pipeline for a 2D spatial domain and denote the state and the encoded state as $s \in \mathbb{R}^{N_x \times N_y}$ and $z \in \mathbb{R}^{n_x \times n_y \times d_f}$ respectively. The dynamics are then to be learned in the z-space by the attention mechanism and the transformer model, which are explained next.

A simple visualization of the CAE and its functionality is provided in Fig. 1a. The detailed architecture is similar to [48], each consisting of 4 encoder or decoder blocks respectively. An encoder block is composed of a convolution layer, an average pooling layer, and a nonlinear activation function. A decoder block is composed of a linear upsampling layer, a convolution layer, and a nonlinear activation function. Leaky ReLU is the choice for the activation function since it does not have a saturation region that causes gradient vanishing, excluding the final block of both models which do not have an activation function. All convolution layers use kernel size 3, and both pooling and upsampling layers use a scale of 2.

2.2. The attention mechanism

In order to pass the encoded state z to the attention mechanism, it is reshaped as a set of $n = n_x n_y$ feature vectors denoted as $f_i \in \mathbb{R}^{d_f}$ where $i = 0, 1, 2, \ldots, n-1$. We also define a corresponding set of vectors containing positional information about the location of f_i vectors as $p_i \in \mathbb{R}^{d_p}$. A simple choice for p would be $p = [x, y] \in \mathbb{R}^2$ containing the index or value in x and y axis. A general formulation of the attention mechanism is presented in Eq. (3).

$$\Delta f_i := \sum_{j=0}^{n-1} \kappa(p_i, p_j, f_i, f_j) V(f_j), \qquad \forall i \in \{0, 1, \ldots, n-1\} \tag{3}$$

Here, $\kappa : \mathbb{R}^{2(d_p+d_f)} \to \mathbb{R}$ is the attention weight function, $V : \mathbb{R}^{d_f} \to \mathbb{R}^{d_f}$ is called the value function, and Δf_i is the incremental change of f_i. This formula can be viewed as a discretized version of a kernel integral operator K_ϕ used in neural operators like FNO [42], as shown in Eq. (4).

$$(K_\phi(a)v)(x) := \int_D \kappa_\phi(x, \tilde{x}, a(x), a(\tilde{x}))v(\tilde{x})d\tilde{x}, \qquad \forall x \in D \tag{4}$$

Here, $\kappa_\phi : \mathbb{R}^{2(d_x+d_a)} \to \mathbb{R}^{d_v \times d_v}$ is a parameterized kernel function and v is the input to the operator. There are interesting correspondences between the two formulations; $x, \tilde{x}, v(\tilde{x})$ are analogous to p_i, p_j, f_j in Eq. (3) respectively. The spatial variable is denoted as x in Eq. (4) which can be of any number of dimensions, while we denote it as p for position to avoid confusion with the first spatial dimension. In Eq. (4), a is the model's input which can be initial or boundary conditions or the parameters of the PDE, and it does not have an exact counterpart in Eq. (3). The initial condition is, however, of the same shape and nature as the output of each layer and can be treated as the input to the operator itself.

The original attention [11] and ROMER [48] assume κ of the form shown in Eq. (5), where Q, K are linear functions named query and key, and $\cdot$ is the dot product. The value function V is also linear and takes $f + p$ as input instead of f. Basically, the positional information is incorporated into the model by being added to the input vectors at the beginning. This can be problematic since it limits the shape of the positional feature to be the same as the main features and also deviates the values of the feature vectors for the upcoming calculations later in the model.

$$\kappa(p_i, p_j, f_i, f_j) = Q(f_i + p_i) \cdot K(f_j + p_j) \tag{5}$$

our model assumes a different form shown in Eq. (6) where $G : \mathbb{R}^{2d_p} \to \mathbb{R}$ is the positional encoder modeled by a fully connected network. This formulation disentangles the effect of positional and feature information and conserves the actual value of the input vectors for further calculations. A graphic representation of this attention mechanism is illustrated in Fig. 1b.

$$\kappa(p_i, p_j, f_i, f_j) = G(p_i, p_j)(Q(f_i) \cdot K(f_j)) \tag{6}$$

A desirable and intuitive property for the positional encoder is to be a function of the relative location of p_j to p_i, leading us to choose the formulation shown in Eq. (7). This also makes the overall model conserve the property of translation invariance, an important and helpful inductive bias for physical systems and PDE solution operators. More arguments can also be included that conserve this property based on the design of p.

$$G(p_i, p_j) = G(p_j - p_i, |p_i - p_j|) \tag{7}$$

2.3. Transformer

The architecture chosen to model the dynamics in the encoding space is based on the classic transformer architecture [11] excluding normalization layers. The architecture of the transformer is shown in Fig. 1c. The multi-head attention is simply a set of independent attention mechanisms that have their output concatenated together, each called an attention head. Each head learns distinct Q, K, V, G which helps the expressive ability of the model as opposed to learning one positional encoder G which may limit the ability to capture a diverse range of dynamics in the spatial domain. For example, one head can focus on close-range dynamics and one can learn longer-range dynamics, where G assigns bigger weights to closer and further values respectively. The attention mechanism models the interaction of the feature vectors in space, the fully connected layers model the dynamics of individual elements over time, and the residual connection is meant to model time-stepping and incremental change of the system in each time-step.

A detailed breakdown of the procedure happening in a transformer layer is provided in algorithm 1. Note that the for loops are implemented in a parallelized manner using tensor-based operations. We chose to explain the procedure using for loops to make it easier and clearer to understand what happens between each pair of vectors in each head, and how they come together.

2.4. Assembly and training

The whole model consists of an autoencoder and several transformers, each meant to model the dynamics at a specific time scale. This was inspired by the work of Liu et al. [62] which took a similar approach but with low-order non-linear models and proved the concept using fully connected neural networks. We denote the encoder and decoder as P, Q and the model that propagates the encoded state to the future for Δt time-steps as $D^{\Delta t}$. This work uses up to four different dynamical models with $\Delta t = \{1, 2, 4, 8\}$.

The loss function used to train all models is the normalized Root Mean Squared Error (nRMSE) defined in Eq. (8), where $\tilde{z}$ and z represent the model's output and the ground truth for a certain sample respectively. This loss provides a measure of error not affected by the magnitude or resolution of data, making it a popular choice to evaluate PDE solvers [42]. The total loss is obtained by averaging the loss over all of the samples (trajectories) in each dataset. We use the Adam optimization algorithm [63] to minimize

Algorithm 1 The forward pass of a transformer layer in Fig. 1c.

Inputs: Spatially flattened encoding $z_t = \{f_i | i \in \{0, 1, ..., n-1\}\}$ and the corresponding positional features $\{p_i | i \in 0, 1, ..., n-1\}$

for $h = 1, 2, ..., H$ (heads) **do**
  for $i = 0, 1, ..., n-1$ **do**
    for $j = 0, 1, ..., n-1$ **do**
      $\alpha_{ij} = Q^h(f_i) \cdot K^h(f_j)$
    end for
    $\kappa_{ij} = \frac{e^{\alpha_{ij}}}{\sum_{j=0}^{n-1} e^{\alpha_{ij}}}$ (softmax)
    $\kappa_{ij} \leftarrow G^h(p_i, p_j)\kappa_{ij}$ (multiply by positional encoding)
    $df_i^h = \sum_{j=0}^{n-1} \kappa_{ij} V^h(f_j)$
  end for
end for
for $i = 0, 1, ..., n-1$ **do**
  $df_i = Concat(df_i^1, df_i^2, ...df_i^H)W_{proj}$ (linear projection)
  $f_i \leftarrow f_i + df_i$
  $f_i \leftarrow f_i + FFN(f_i)$
end for

the loss with the initial learning rate 0.001 and a learning rate scheduler that reduces the learning rate by a factor of 0.2 where the learning curve reaches a plateau, with a patience factor of 5.

$$nRMSE(\tilde{z}, z) = \frac{\left\| \tilde{z} - z \right\|_2}{\left\| z \right\|_2} \tag{8}$$

First, the autoencoder is trained to extract spatial features and dominant patterns. The loss function for the autoencoder is defined by setting $z = s$ and $\tilde{z} = Q(P(s))$ in Eq. (8). After training, the encoder and decoder are used to move to and from the encoding space, in which the dynamics are to be learned by the transformer models.

The calculation of the loss function for the dynamical models is shown in algorithm 2. An important training strategy of this work is using rollout in training, also known as backpropagation through time. Models in similar applications usually use a full rollout ($R = T - 1$) which prevents parallelized training across time hence increasing training time, and introduces a risk of gradient explosion or vanishing as well. Unlike the original applications of many recurrent models like natural language processing (NLP), a physical system has the Markov property, meaning that its future is dependent only on the present and not the past provided a sufficient state representation. Therefore, a full rollout may cost too much compared to its advantage. By using a finite-time rollout, our model can benefit from the advantages while still being able to parallelize the training over time with much less risk of gradient explosion or vanishing.

Algorithm 2 Calculation of the loss function of $D^{\Delta t}$ for a sample trajectory of length T, using a trained encoder P and training rollout R

for $t = 0, 1, \dots, T - R\Delta t$ **do**
  $Loss = 0$
  $z_t = P(s_t)$
  $\tilde{z}_t = z_t$
  for $r = 1, 2, \dots, R$ **do**
    $z_{t+r\Delta t} = P\left(s_{t+r\Delta t}\right)$
    $\tilde{z}_{t+r\Delta t} = D^{\Delta t}\left(\tilde{z}_{t+(r-1)\Delta t}\right)$
    $Loss \leftarrow Loss + nRMSE\left(\tilde{z}_{t+r\Delta t}, z_{t+r\Delta t}\right)$
  end for
  $Loss \leftarrow Loss \div R$
end for

We also use transfer learning to ease the training for the dynamical models. The dynamics for a large time-step are expected to be more complex and difficult than for small time-steps. Therefore, we use the trained dynamical models for better initialization of the next models to train for larger time-steps. For example, D^2 is initialized by the trained weights of D^1, D^4 by the trained D^2, and D^8 by D^4. The overall framework is summarized in Fig. 2.

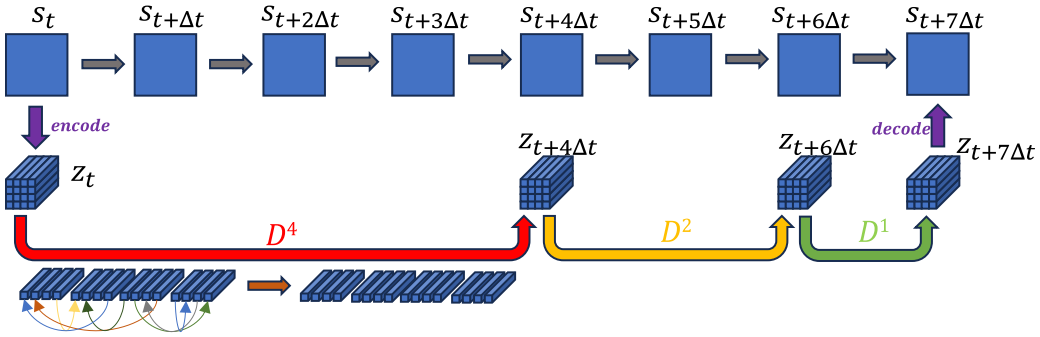

Fig. 2. Our strategy of using the transformer for time-stepping (bottom row) compared to the default strategy (top row). By reducing the spatial dimension and moving to the latent space, the cost of a forward pass is greatly decreased. Then by using multi-scale time-stepping, we also decrease the number of required forward passes to reach a certain time. Each transformer models the mixing of coarsened features in the space (shown below D^4) for a specific time step.

3. Experiments and discussion

In order to assess the capability of our model, we train it on two challenging tasks used to evaluate powerful and novel PDE solvers like neural operators. We choose 2D incompressible Navier–Stokes (NS) with three different Reynolds numbers, and a low-Reynolds 2D Kolmogorov Flow (KF), both provided publicly by Li et al. in [42]. The performance of the model on these datasets is considered a reliable metric for its general capabilities because of their multi-dimensional spatial domain and nonlinearity, and they are usually presented as the last challenge to such models in the literature after 1D and/or linear datasets. Since the computational cost is more of a concern in multi-dimensional domains, we evaluate our model on these datasets to highlight the benefits of our framework.

According to [42], the NS datasets were generated by solving the 2D Navier–Stokes equations for a viscous incompressible flow in vorticity form on the unit torus with periodic boundary conditions:

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

Here u is the velocity field, $\omega = \nabla\times u$ is the vorticity, ω_0 is the initial vorticity, ν is the viscosity coefficient, and f is the forcing function. The equation was solved using the stream-function formulation with a pseudospectral method, the time-stepping was done using the Crank–Nicolson scheme with a time-step of $1e{-}4$, and the solution was recorded every $t = 1$ time units. The simulations used a 256×256 spatial grid and the results were recorded in a 64×64 grid.

Each sample in the dataset was obtained by solving the equation over the spatiotemporal domain for a random initial condition taken from a distribution denoted as μ. The distribution of the initial conditions and the forcing term used are defined below. The distribution μ is of a Gaussian Random Field, where the covariance is defined using the Matérn-like covariance operator based on the notation used in [64].

$$\mu = N(0, 7^{3/2}(-\Delta + 49I)^{-2.5})$$
$$f(x) = 0.1(sin(2\pi(x_1 + x_2)) + cos(2\pi(x_1 + x_2)))$$

Each of the NS datasets was obtained by solving the equations with a different viscosity among $\nu\in\{1e{-}3, 1e{-}4, 1e{-}5\}$ and contains 1000 training samples and 200 test samples. The KF dataset consists of 160 training samples and 40 test samples.

We start with the NS datasets and compare the performance of our model with TF-Net, the powerful FNO, and two transformer-based neural operators. The transformer we use for all dynamical models on NS datasets has 4 layers and 8 heads, and each component of the models is trained for 100 epochs with a batch size of 64. The results are shown in Table 1. Since the competing models take 10 time-steps as their input, the reported results in their works are measured on the predicted remaining time-steps (excluding the first 10 time-steps). Therefore, we start the evaluation of the model from the 10th time-step as well, although our model only takes a single time-step as input. Symbols T and T_{pred} represent the total time horizon of each dataset and the time window predicted by the models respectively. The reported error is normalized RMSE on the final 3D output containing the 2D state over the predicted time window, which is of shape $N_x\times N_y\times T_{pred}$. It is observed that our model performs almost as well as FNO on NS1, and outperforms all models on NS2 and NS3 with an impressive margin considering the turbulence and relatively small dataset size. This is despite the fact that our model uses finite training rollout and a single time-step as input while the competing models used 10 time-steps and full training rollout. A visual comparison for each dataset is shown in Fig. 4.

The NS1 dataset has a low Reynolds number and a long time horizon, making the primary challenge the error accumulation over time rather than complex and chaotic dynamics in the immediate future. Our model achieves an average test error of 0.88% using multi-scale time-stepping with 4 dynamical models and a training rollout of 2 for each. Compared to a full rollout, the parallelization

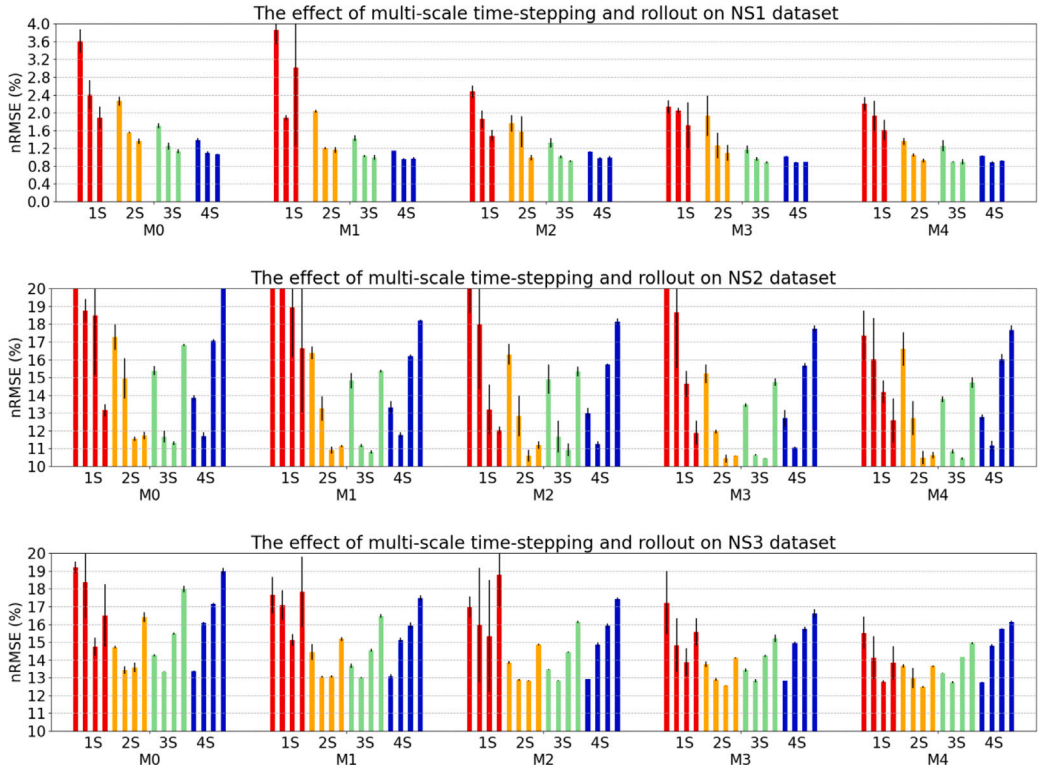

Fig. 3. Multi-scale time-stepping and rollout effect on NS datasets. Each bar group represents a different type of attention mechanism from Table 2 denoted by M. Models of the same color use the same number of time scales denoted by their label (1S, 2S, 3S, 4S), in which different rollouts of $R = 1, 2, 4, 8$ are shown from left to right (except for $R = 8$ excluded from NS1). The black error bars represent the variation across three random seeds that each model was trained with. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Benchmarks on 2D Navier–Stokes data provided by Li et al. [42]. Results reported by the papers FNO [42] and OFormer [40].

Data	NS1	NS2	NS3	#param(M)
ν	$1e{-}3$	$1e{-}4$	$1e{-}5$	–
$T(T_{pred})$	50 (40)	30 (20)	20 (10)	–
TF-Net [65]	2.25%	22.53%	22.68%	7.45
FNO-3D [42]	**0.86%**	19.18%	18.93%	6.56
FNO-2D [42]	1.28%	15.59%	15.56%	0.41
GT [39]	0.94%	**13.99%**	**13.40%**	1.56
OFormer [40]	1.04%	17.55%	17.05%	1.85
Ours	**0.88%**	**10.43%**	**12.48%**	1.02

of training across time can be almost fully maintained, making it possible to speed up the training up to 10 times faster, since we also train four models. Moreover, the cost of the forward pass to predict the state at $t = 49$ for example can be reduced up to 8 times since only 5 forward passes of D^8 are needed instead of 40 forward passes. The lesson here is that when the dynamics are simple enough, modeling large time-steps and a smaller training rollout can be a suitable strategy to achieve high accuracy with shorter and easier training.

The situation is different for NS2 and NS3 since they have more complex dynamics and shorter time horizons. The primary contributor to the prediction error here is the complex dynamics even in the immediate future. This is the reason FNO-3d, which outputs all future time-steps in a single forward pass, does not outperform FNO-2d as it does on NS1. The great performance of our model on these datasets implicates the remarkable potential of the architecture of its architecture and the enhancements introduced in this work to model complex spatiotemporal dynamics while having the Markov property.

To validate the advantage of our model in terms of computational efficiency, we compare the training and testing time of our model compared to the competing autoregressive models FNO-2d [42], Galerkin Transformer (GT) [39] and OFormer [40] as shown in Table 3. The training time is spent on forward and backward passes, while the testing time is spent only on the forward pass. All models are set to take one time-step as input and give one-step prediction as output. The training is done using a full rollout to leverage the benefits of time-stepping in the latent space. The model hyperparameters for the competing models are taken

Table 2
Explored options for attention and positional encoder.

Name	Positional encoding	Insertion method
M0	None	None
M1	FFN(cartesian grid)	Additive to input
M2	FFN(periodic features)	Additive to input
M3	FFN(periodic features)	Additive before softmax
M4	FFN(periodic features)	Multiplicative after softmax

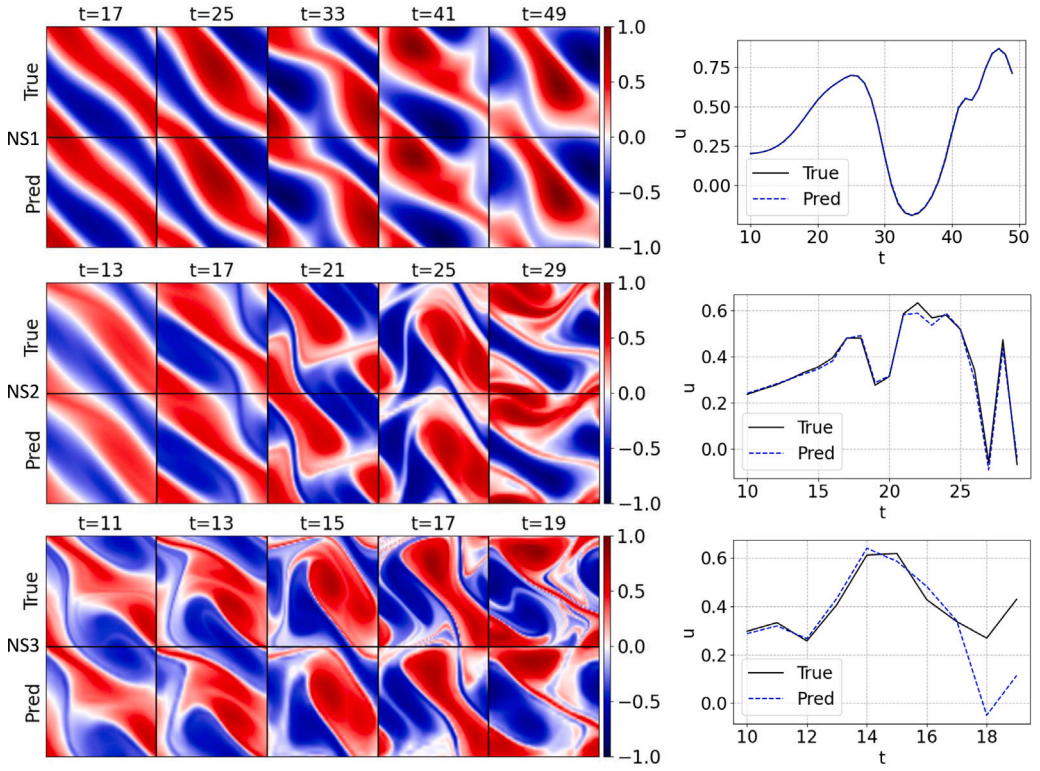

Fig. 4. Comparison of the model's predictions with the ground truth for a sample test trajectory, starting from $t = 9$ (10th time-step) going forward. The value at the center is plotted on the right column for each dataset. The error increases with the more turbulent datasets and over time.

Table 3
Time spent to process a batch of 8 samples, each being a trajectory of 11 time-steps and 64 × 64 snapshots. A full training rollout of 10 is used.

Model	Train (s)	Test (s)
FNO-2d	0.0821 ± 0.0013	0.0289 ± 0.0008
GT	0.2518 ± 0.0022	0.1027 ± 0.0012
OFormer	0.4319 ± 0.0058	0.1084 ± 0.0004
Our autoencoder	0.0045 ± 0.0003	0.0011 ± 0.0001
Our model	0.0707 ± 0.0017	0.0351 ± 0.0010

from [40]. The advantage is mainly because the encoding in our model is of spatial resolution 4 × 4 while others maintain the original resolution of 64 × 64. All runs were conducted on one NVIDIA GeForce RTX 2080 Ti GPU. The cost is separately reported for our autoencoder and dynamical model. One can estimate the total time by summing the two, although the contribution of the autoencoder is relatively small.

In order to find the best choice for the number of time scales and the training rollout, as well as to verify the improvement of the model due to our proposed modifications, we experiment with 5 different types of attention mechanisms (Table 2) with different settings for multi-scale time-stepping and training rollout. Each model is trained with three different random seeds, and the average and standard deviation of the test error for each configuration are shown in Fig. 3. It is observed that M3 and M4 outperform the other options as expected because of the way they incorporate positional information without disturbing the actual input values for

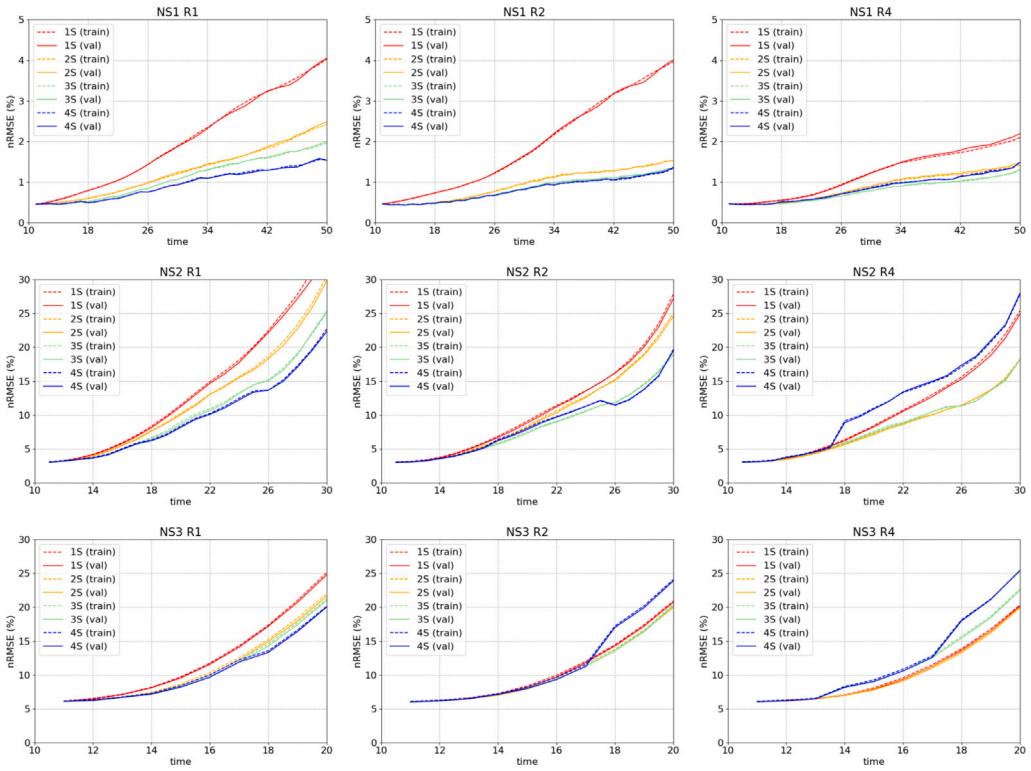

Fig. 5. The evolution of error over time for different datasets and training settings. For more turbulent datasets, the transformer fails to capture the large time-step dynamics while the iterative use of smaller time-steps is successful.

downstream calculations. The importance of including positional information and periodic positional features is also noticeable by looking at M0, M1, and M2.

Looking at the effect of the training rollout, it can be observed that low R causes high uncertainty and variation based on the random seed. This means that even a finite rollout can be of great help to guide the training towards the global optima rather than a random local optima with poor performance when used autoregressively. However, if R is too high considering the complexity of the system, the model may not succeed in simultaneously modeling the immediate future and a more distant future well, leading to an increase in the error for larger rollouts in NS2 and NS3. By finding the right balance, the responsibility of learning the dynamics of different time scales is divided across different models and lets them focus on different tasks.

Fig. 5 illustrates the error of M4 over time with different settings and provides a better insight into the effect of training rollout and multi-scale time-stepping and their balance. The NS1 is a laminar flow, therefore predicting the future even for 8 time-steps with a single model is possible. However, NS2 and NS3 show a different behavior as using a smaller time-step and a finite training rollout shows better performance over directly predicting larger time-steps into the future. The increase of error for the models using 4 time-scales (including D^8) is due to the failure of D^8 to do as well as other models taking smaller but more steps.

Moving on from the NS dataset, the results for the Kolmogorov Flow were not as successful but led to interesting observations. Competing models like FNO and GT also face similar limitations, so our model does not seem to have an advantage in this case. Although the final loss for both the autoencoder and the dynamical models were similar to NS3, our model failed to learn rapid changes accurately. However, it seems to perform well in the slow-changing regimes as shown in Fig. 6. Upon closer investigation, this turned out to be because of the data distribution. Since the training is done across time, starting from arbitrary time-steps and predicting a finite number of future steps, it overfits the majority of the data which consists of a slow-changing and almost steady flow. Fig. 8 provides some insight into the distribution of the data. The great imbalance of the distribution of the KF dataset is a suspected factor that makes the model overfit to the slow-changing regime which makes up most of the data. This is not the case for NS1 and NS2 (top row) which seem to have a more symmetric distribution. Although the NS3 dataset is also asymmetric, the KF dataset has a longer tail that consists of data until 1.0 on the horizontal axis and even a few points above 1.0, which are greatly underrepresented in the dataset. In the case of predicting time-series, such an error is enough to sabotage the prediction even if the model has learned most of the data since a large error for a short time is enough to diverge the remainder of the predicted trajectory from the ground truth. Still, the energy spectrum of the model's predictions mostly matches the ground truth except for the high

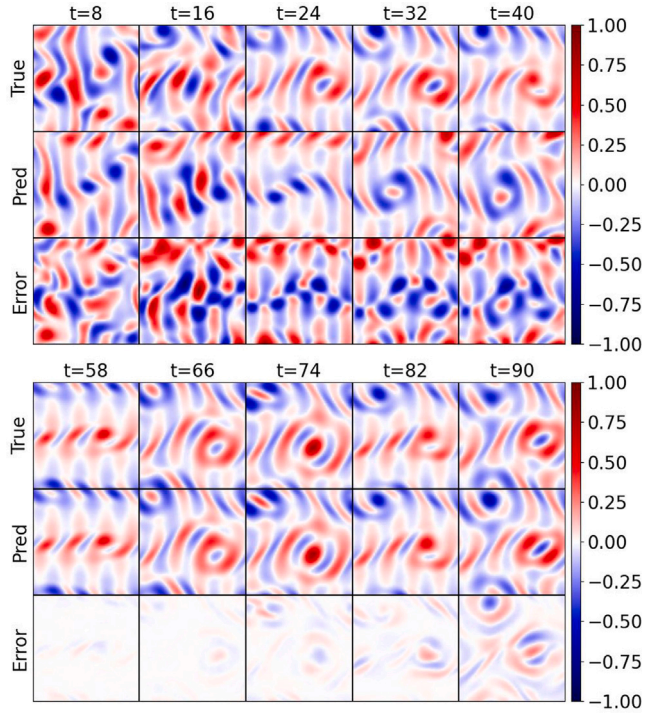

Fig. 6. The model overfitting to the slow-changing regime in the Kolmogorov Flow and not learning the fast transient dynamics. Top: A rollout of D^1 for 40 time-steps, starting from $t_0 = 0$ (fast change). Bottom: A rollout of D^1 for 40 time-steps, starting from $t_0 = 50$ (slow change).

frequencies as shown in Fig. 7. Since our model uses a coarsened spatial encoding, it fails to model the high frequencies accurately as expected.

Finally, we would like to point out an interesting observation about the performance of the model over an extensive amount of time. As shown in Fig. 9, the model's output seems to stay stable for a very long time. Although the predictions are probably not physically accurate, stability of the output is a desirable property that models like Galerkin Transformer lack since they omitted the softmax operation [39], which may cause problems even in the available time-window in the data. The time horizon of the NS datasets is not enough to compare the long-term performance of the model to the ground truth. However, in the KF dataset as well as the NS datasets, the predictions seem to contain the patterns present in the data and not show unstable behavior. This can be further leveraged and investigated in future work both theoretically and empirically.

4. Conclusion and future work

This work illustrated the potential of the transformer architecture to develop models to solve time-dependent PDEs that are efficient, accurate, and have the Markov property. By introducing effective enhancements to the transformer architecture and the training process, similar or better performance was observed compared to powerful models such as the Fourier Neural Operator or other transformer-based neural operators. By using a finite-horizon rollout in training, multi-scale time-stepping, and a new way to include positional information into the attention mechanism, our model was able to model the solution of 2D Navier–Stokes equations in turbulent regimes with a small training dataset, and fast training and testing. However, the model failed to learn the rapid dynamics in the Kolmogorov dataset and overfitted to the slow-changing dynamics. However, the model exhibits an interesting behavior when used for a long time and seems to converge to a certain regime of solutions instead of getting unstable like some traditional numerical methods. A possible line of future research is to investigate how we can improve the accuracy and reliability of these predictions using corrective models or iterative methods, which would not be possible with an unbounded and unstable prediction to begin with.

Despite the accuracy and efficiency of this framework, it comes with its limitations which are to be addressed in future work. For example, if the resolution of the state representation is too high, the convolutional autoencoder needs many layers to reduce the dimensions to be computationally tractable for the transformer in the dynamical model. This can cause difficulties in training and decrease training and inference speed. If the data is not represented on an equispaced grid, the convolutional autoencoder cannot

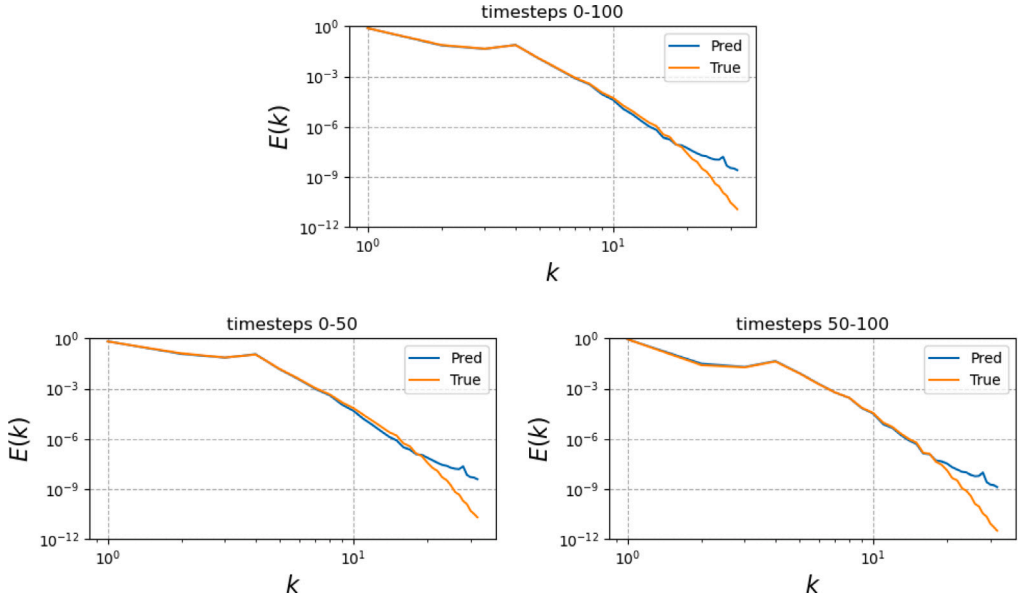

Fig. 7. Comparison of the average energy spectrum of the KF dataset on the validation dataset. We can see a noticeable mismatch for the fast-changing regime (timesteps 0–50) that is not present in the slow-changing regime (50–100) where we use the model after the flow has already entered the slow regime. The mismatch is still visible when we run the model over the combined time window (0–100).

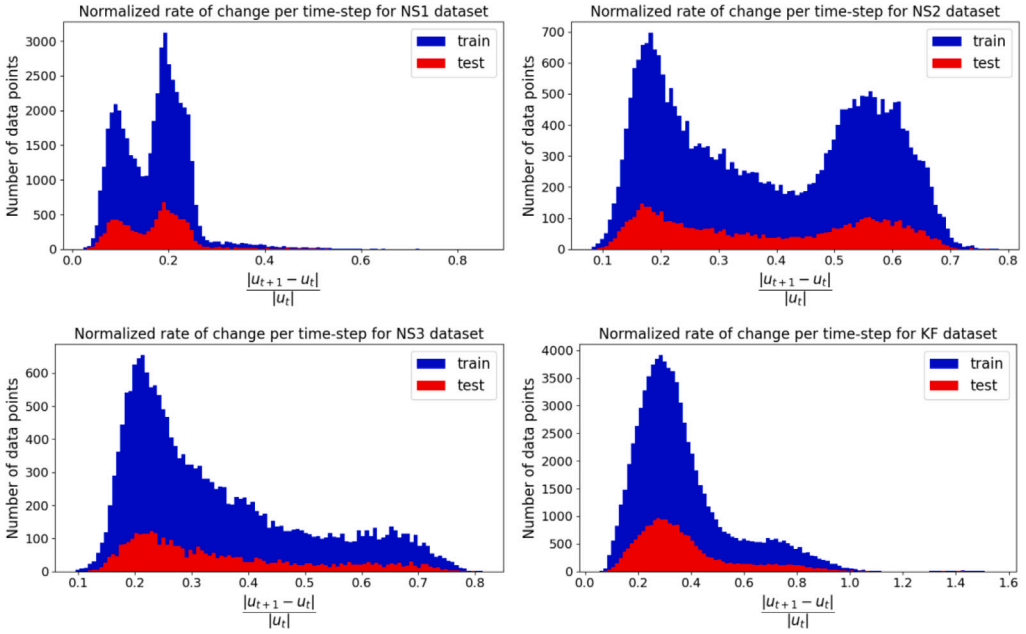

Fig. 8. A brief quantitative analysis of the distribution of the datasets. One suspected reason behind the poor performance of the model on the KF dataset other than its inherent complexity is the skewed distribution of the data and the probable overfitting of the model to the dominant part of the distribution. On the other hand, NS1 and NS2 seem to have a relatively symmetric distribution. NS3 has a skewed distribution but is still closer to a uniform distribution than KF.

be applied anymore. Another potent line of future research is making this scheme applicable to different geometries and boundary conditions. Since other forms of boundary conditions can be considered external influences, a new component is needed to encode them. Regarding general geometries, both reducing the spatial representation and handling of the boundary conditions need new components since the convolutional layers cannot process them. One can also achieve such goals by incorporating these ML-based

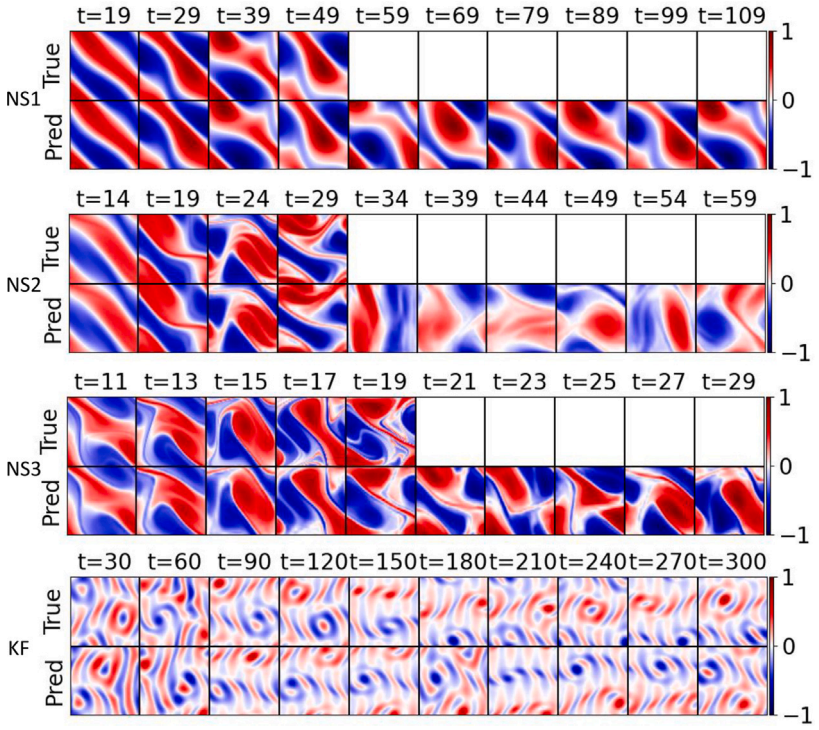

Fig. 9. The performance of the model over an extended time period. Interestingly, the model does not exhibit any signs of instability, although the results may not be physically correct. Blank fields represent missing ground truth data.

models as a part of classic numerical methods that are more capable of handling these generalizations by design [66]. Another quality that our model lacks compared to the competing models is being applicable to arbitrary resolutions. Since the convolutional autoencoder is designed and trained for a specific input resolution, it needs to be replaced or modified if one wants to make this framework applicable to arbitrary input resolutions like neural operators.

This framework also introduces several hyperparameters such as the number of time-scales and time-step sizes to use, as well as the balance between the spatial dimensions and the number of channels of the encoding. In future work, coming up with a systematic and quick way of finding the best choice of training rollout or the number of time scales can be a promising venue for subsequent studies. Moreover, a theoretical investigation of the model and its weights and the connection between the dynamical models of different time scales may be able to provide new intuition and insight into the attention mechanism and its capabilities in modeling dynamic systems. The modifications introduced in this work can be utilized on any model trying to learn the solution of time-dependent PDEs, as well as models based on the transformer architecture. The model also has the potential to be equipped with additional tools to handle and incorporate time-variant forcing terms, which can be of high interest in many applications such as model-predictive control of fluid flows and therefore a promising venue for future research. We hope this work encourages researchers to move toward finding the best ways of incorporating the amazing transformer architecture in scientific computation applications.

CRediT authorship contribution statement

AmirPouya Hemmasian: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Conceptualization. **Amir Barati Farimani:** Writing – review & editing, Supervision, Resources, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code and data will be publicly available.

Acknowledgment

This work was supported by the National Science Foundation, United States under Grant No. 1953222.

References

[1] S.L. Brunton, J.N. Kutz, Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control, Cambridge University Press, 2019.
[2] M. Frank, D. Drikakis, V. Charissis, Machine-learning methods for computational science and engineering, Computation 8 (1) (2020) 15.
[3] S. Cuomo, V.S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics–informed neural networks: Where we are and what's next, J. Sci. Comput. 92 (3) (2022) 88.
[4] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, S. Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Comput. Mech. 64 (2) (2019) 525–545, http://dx.doi.org/10.1007/s00466-019-01740-0.
[5] K. Lee, K.T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, J. Comput. Phys. 404 (2020) 108973.
[6] K. Fukami, T. Nakamura, K. Fukagata, Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data, Phys. Fluids 32 (9) (2020) 095110.
[7] H. Gao, L. Sun, J.-X. Wang, PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain, J. Comput. Phys. 428 (2021) 110079.
[8] A.T. Mohan, D.V. Gaitonde, A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks, 2018, arXiv preprint arXiv:1804.09269.
[9] K. Hasegawa, K. Fukami, T. Murata, K. Fukagata, CNN-LSTM based reduced order modeling of two-dimensional unsteady flows around a circular cylinder at different Reynolds numbers, Fluid Dyn. Res. 52 (6) (2020) 065501.
[10] R. Maulik, B. Lusch, P. Balaprakash, Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders, Phys. Fluids 33 (3) (2021) 037106.
[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), in: Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017.
[12] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, 2018, http://dx.doi.org/10.48550/ARXIV.1810.04805, URL: https://arxiv.org/abs/1810.04805.
[13] K. Tunyasuvunakool, J. Adler, Z. Wu, T. Green, M. Zielinski, A. Žídek, A. Bridgland, A. Cowie, C. Meyer, A. Laydon, S. Velankar, G. Kleywegt, A. Bateman, R. Evans, A. Pritzel, M. Figurnov, O. Ronneberger, R. Bates, S. Kohl, D. Hassabis, Highly accurate protein structure prediction for the human proteome, Nature 596 (2021) 1–9, http://dx.doi.org/10.1038/s41586-021-03828-1.
[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. 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.
[15] G. Berkooz, P. Holmes, J.L. Lumley, The proper orthogonal decomposition in the analysis of turbulent flows, Annu. Rev. Fluid Mech. 25 (1) (1993) 539–575.
[16] J.S. Anttonen, P.I. King, P.S. Beran, POD-based reduced-order models with deforming grids, Math. Comput. Modelling 38 (1–2) (2003) 41–62.
[17] T. Murata, K. Fukami, K. Fukagata, Nonlinear mode decomposition with convolutional neural networks for fluid dynamics, J. Fluid Mech. 882 (2020) A13.
[18] J.S. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, J. Comput. Phys. 363 (2018) 55–78.
[19] S.L. Brunton, B.R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (2020) 477–508.
[20] S. Wiewel, M. Becher, N. Thuerey, Latent space physics: Towards learning the temporal evolution of fluid flow, in: Computer Graphics Forum, vol. 38, Wiley Online Library, 2019, pp. 71–82.
[21] V. Shankar, G. Portwood, A. Mohan, P. Mitra, C. Rackauckas, L. Wilson, D. Schmidt, V. Viswanathan, Learning non-linear spatio-temporal dynamics with convolutional neural ODEs, in: Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020), 2020.
[22] K. Lee, K.T. Carlberg, Deep conservation: A latent-dynamics model for exact satisfaction of physical conservation laws, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, 2021, pp. 277–285.
[23] Y. Khoo, J. Lu, L. Ying, Solving parametric PDE problems with artificial neural networks, Eur. J. Appl. Math. 32 (3) (2021) 421–435.
[24] J.K. Gupta, J. Brandstetter, Towards multi-spatiotemporal-scale generalized PDE modeling, 2022, arXiv:2209.15616.
[25] S. JANNY, A. Bénéteau, M. Nadri, J. Digne, N. THOME, C. Wolf, EAGLE: Large-scale learning of turbulent fluid dynamics with mesh transformers, in: The Eleventh International Conference on Learning Representations, 2023, URL: https://openreview.net/forum?id=mfIX4QpsARJ.
[26] Z. Li, A.B. Farimani, Graph neural network-accelerated Lagrangian fluid simulation, Comput. Graph. 103 (2022) 201–211, http://dx.doi.org/10.1016/j.cag.2022.02.004.
[27] P. Pant, R. Doshi, P. Bahl, A. Barati Farimani, Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations, Phys. Fluids 33 (10) (2021) 107101, http://dx.doi.org/10.1063/5.0062546, arXiv:https://doi.org/10.1063/5.0062546.
[28] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, P.W. Battaglia, Learning mesh-based simulation with graph networks, 2021, arXiv:2010.03409.
[29] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P.W. Battaglia, Learning to simulate complex physics with graph networks, 2020, arXiv:2002.09405.
[30] F.D.A. Belbute-Peres, T. Economon, Z. Kolter, Combining differentiable PDE solvers and graph neural networks for fluid flow prediction, in: International Conference on Machine Learning, PMLR, 2020, pp. 2402–2411.
[31] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[32] G. Karniadakis, Y. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, 2021, pp. 1–19, http://dx.doi.org/10.1038/s42254-021-00314-5.
[33] S. Cai, Z. Mao, Z. Wang, M. Yin, G.E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: A review, Acta Mech. Sin. 37 (12) (2021) 1727–1738.
[34] G. Pang, L. Lu, G.E. Karniadakis, FPINNs: Fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626.
[35] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Eng. 361 (2020) 112732.
[36] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Netw. 6 (4) (1995) 911–917, http://dx.doi.org/10.1109/72.392253.

[37] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218–229, http://dx.doi.org/10.1038/s42256-021-00302-5.
[38] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, 2020, arXiv:2003.03485.
[39] S. Cao, Choose a transformer: Fourier or galerkin, Adv. Neural Inf. Process. Syst. 34 (2021) 24924–24940.
[40] Z. Li, K. Meidani, A.B. Farimani, Transformer for partial differential equations operator learning, Trans. Mach. Learn. Res. (2023) URL: https://openreview.net/forum?id=EPPqt3uERT.
[41] G. Gupta, X. Xiao, P. Bogdan, Multiwavelet-based operator learning for differential equations, 2021, arXiv:2109.13459.
[42] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, 2020, arXiv preprint arXiv:2010.08895.
[43] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, S.M. Benson, U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow, Adv. Water Resour. 163 (2022) 104180.
[44] A. Tran, A. Mathews, L. Xie, C.S. Ong, Factorized Fourier neural operators, 2023, arXiv:2111.13802.
[45] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, S. Zagoruyko, End-to-end object detection with transformers, 2020, arXiv:2005.12872.
[46] N. Geneva, N. Zabaras, Transformers for modeling physical systems, Neural Netw. 146 (2022) 272–289.
[47] A. Solera-Rico, C.S. Vila, M. Gómez, Y. Wang, A. Almashjary, S. Dawson, R. Vinuesa, β-Variational autoencoders and transformers for reduced-order modeling of fluid flows, 2023, arXiv preprint arXiv:2304.03571.
[48] A. Hemmasian, A. Barati Farimani, Reduced-order modeling of fluid flows with transformers, Phys. Fluids 35 (5) (2023).
[49] R. Guo, S. Cao, L. Chen, Transformer meets boundary value inverse problems, 2022, arXiv preprint arXiv:2209.14977.
[50] Z. Hao, C. Ying, Z. Wang, H. Su, Y. Dong, S. Liu, Z. Cheng, J. Zhu, J. Song, GNOT: A general neural operator transformer for operator learning, 2023, arXiv preprint arXiv:2302.14376.
[51] O. Ovadia, A. Kahana, P. Stinis, E. Turkel, G.E. Karniadakis, ViTO: Vision transformer-operator, 2023, arXiv preprint arXiv:2303.08891.
[52] G. Kissas, J. Seidman, L.F. Guilhoto, V.M. Preciado, G.J. Pappas, P. Perdikaris, Learning operators with coupled attention, 2022, arXiv:2201.01032.
[53] X. Liu, B. Xu, L. Zhang, HT-Net: Hierarchical transformer based operator learning model for multiscale PDEs, 2022, arXiv preprint arXiv:2210.10890.
[54] T. Nguyen, M. Pham, T. Nguyen, K. Nguyen, S. Osher, N. Ho, Fourierformer: Transformer meets generalized fourier integral theorem, Adv. Neural Inf. Process. Syst. 35 (2022) 29319–29335.
[55] Z. Li, D. Shu, A.B. Farimani, Scalable transformer for PDE surrogate modeling, 2023, arXiv preprint arXiv:2305.17560.
[56] X. Han, H. Gao, T. Pfaff, J.-X. Wang, L.-P. Liu, Predicting physics in mesh-reduced space with temporal attention, 2022, arXiv preprint arXiv:2201.09113.
[57] J.W. Rae, A. Potapenko, S.M. Jayakumar, T.P. Lillicrap, Compressive transformers for long-range sequence modelling, 2019, arXiv preprint arXiv:1911.05507.
[58] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, A. Ahmed, Big bird: Transformers for longer sequences, 2021, arXiv:2007.14062.
[59] I. Beltagy, M.E. Peters, A. Cohan, Longformer: The long-document transformer, 2020, arXiv:2004.05150.
[60] N. Kitaev, L. Kaiser, A. Levskaya, Reformer: The efficient transformer, in: International Conference on Learning Representations, 2020, URL: https://openreview.net/forum?id=rkgNKkHtvB.
[61] Z. Shen, M. Zhang, H. Zhao, S. Yi, H. Li, Efficient attention: Attention with linear complexities, 2020, arXiv:1812.01243.
[62] Y. Liu, J.N. Kutz, S.L. Brunton, Hierarchical deep learning of multiscale differential equation time-steppers, Phil. Trans. R. Soc. A 380 (2229) (2022) 20210200.
[63] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, http://dx.doi.org/10.48550/ARXIV.1412.6980, URL: https://arxiv.org/abs/1412.6980.
[64] N.H. Nelsen, A.M. Stuart, The random feature model for input-output maps between banach spaces, SIAM J. Sci. Comput. 43 (5) (2021) A3212–A3243.
[65] R. Wang, K. Kashinath, M. Mustafa, A. Albert, R. 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, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1457–1466, http://dx.doi.org/10.1145/3394486.3403198.
[66] V. Shankar, R. Maulik, V. Viswanathan, Differentiable turbulence II, 2023, arXiv preprint arXiv:2307.13533.