Journal of Computational Physics 533 (2025) 113996

Contents lists available at ScienceDirect

Journal of Computational Physics

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

Tensor decomposition-based neural operator with dynamic mode decomposition for parameterized time-dependent problems

Yuanhong Chen [a], Yifan Lin [b], Xiang Sun [a], Chunxin Yuan [a], Zhen Gao [a,*]

[a] School of Mathematical Sciences, Ocean University of China, Qingdao 266100, PR China
[b] School of Physical and Mathematical Sciences, Nanjing Tech University, Nanjing 211816, PR China

ARTICLE INFO

Keywords:
DeepONet
Parameterized time-dependent PDEs
Tensor train decomposition
Dynamic mode decomposition

ABSTRACT

Deep operator networks (DeepONets), as a powerful tool to approximate nonlinear mappings between different function spaces, have gained significant attention recently for applications in modeling parameterized partial differential equations. However, limited by the poor extrapolation ability of purely data-driven neural operators, these models tend to fail in predicting solutions with high accuracy outside the training time interval. To address this issue, a novel operator learning framework, TDMD-DeepONet, is proposed in this work, based on tensor train decomposition (TTD) and dynamic mode decomposition (DMD). We first demonstrate the mathematical agreement of the representation of TTD and DeepONet. Then the TTD is performed on a higher-order tensor consisting of given spatial-temporal snapshots collected under a set of parameter values to generate the parameter-, space- and time-dependent cores. DMD is then utilized to model the evolution of the time-dependent core, which is combined with the space-dependent cores to represent the trunk net. Similar to DeepONet, the branch net employs a neural network, with the parameters as inputs and outputs merged with the trunk net for prediction. Furthermore, the feature-enhanced TDMD-DeepONet (ETDMD-DeepONet) is proposed to improve the accuracy, in which an additional linear layer is incorporated into the branch network compared with TDMD-DeepONet. The input to the linear layer is obtained by projecting the initial conditions onto the trunk network. The proposed methods' good performance is demonstrated through several classical examples, in which the results demonstrate that the new methods are more accurate in forecasting solutions than the standard DeepONet.

1. Introduction

Many real-world problems in science and engineering can be modeled as parameterized partial differential equations (PDEs). With the development of machine learning (ML), ML-based methods are becoming increasingly well-known as an alternative to traditional high-fidelity numerical solvers [1,2], which are computationally expensive and impose a significant computational burden in scenarios that require repeated solving for different parameter values. In particular, the emergence of neural operators aimed at learning nonlinear mapping between different function spaces, such as deep operator networks (DeepONets) [3] and Fourier operator networks (FNOs) [4], have provided a new direction in the solution of the PDEs.

* Corresponding author.
E-mail addresses: chenyuanhong@stu.ouc.edu.cn (Y. Chen), linyifan@njtech.edu.cn (Y. Lin), sunxiang@ouc.edu.cn (X. Sun), yuanchunxin@ouc.edu.cn (C. Yuan), zhengao@ouc.edu.cn (Z. Gao).

https://doi.org/10.1016/j.jcp.2025.113996
Received 16 June 2024; Received in revised form 15 January 2025; Accepted 6 April 2025

Available online 9 April 2025

DeepONet consists of a branch net and trunk net, and can be applied to problems with different initial conditions, boundary conditions, parameter domains, and forcing terms, which are fed into the network as input functions. Compared with the feedforward neural networks (FNNs), DeepONet exhibits higher accuracy and generalization ability. Once trained, DeepONet can solve new problems with different parameters in real-time, making it a valuable tool in contexts such as uncertainty quantification and optimization. Currently, it is employed in a variety of applications, including the prediction of the linear evolution of unstable waves in high-speed boundary layer [5], the approximation of highly non-smooth dynamics [6], the prediction of the response of offshore structures to waves [7], the simulation of bubble growth dynamics [8], the implementation of fast multi-scale modeling [9], and the inference of the multi-physical field of electric convection [10]. Furthermore, various enhanced DeepONets have been proposed, including the multi-input DeepONet [11,12] capable of receiving multiple input functions, the physics-informed DeepONet [13,14] that considers data and physical information as constraints, the multi-fidelity DeepONet [15–17] that combines different fidelity data, the POD-DeepONet [18] that is combined with the proper orthogonal decomposition (POD), and the SVD-DeepONet [19] that is enhanced from the singular value decomposition (SVD) perspective. Neural Implicit Flow [20] has recently emerged as a promising method, a mesh-agnostic nonlinear dimensionality reduction approach similar to DeepONets that enables modal analysis of spatio-temporal dynamics on adaptive meshes.

Despite achieving satisfactory results in numerous cases, DeepONet is constrained by the inherent limitations of purely data-driven neural networks, which are more suited to interpolation, and accurate extrapolation is challenging [21]. In particular, the DeepONet model does not consider time dependencies, which may fail to predict solutions in the time domain that are not visible in the training set. However, in practical applications, constructing a prediction model to achieve long-term prediction is often necessary. One potential solution to this problem is to employ physics-informed knowledge [13,14]. However, these approaches necessitate an understanding of the underlying physics. An alternative approach is to incorporate time-dependent variables into the neural network to learn the temporal evolution of the system. Lin et al. [22] input the current state into the branch net to learn a local approximation of the transition from the current state to the next state, thereby recursively modeling the system's solution. Similarly, Liu et al. developed Causality-DeepONet [23], which inputs time-dependent states into the branch net and employs zero-padding and shifted convolutional windows to capture causality in the input data. However, the introduction of zero-padding data may introduce extra errors. In [24], the predicted sequences of DeepONet are fed into the long short-term memory (LSTM), providing a recursive structure that captures temporal dependencies. In [25,26], the LSTM is incorporated into the branch net to facilitate the learning of the relationship between time-dependent inputs and outputs. Furthermore, the proposed B-LSTM-MIONet in [26] can handle variable-length real-time data. However, the limitation of these methods is that they can only predict the state after a fixed time interval following the prediction of the current state. Besides, a significant number of samples are inevitably required to train the network sufficiently to avoid error accumulation.

To construct a predictive system that describes the evolution of the system state from a dataset, with the goal of predicting the future state, numerous ML-based algorithms have been proposed [27,28]. Among these, DMD [29] identifies the optimal linear system and models the temporal evolution of the system. This approach was originally proposed in the field of fluid dynamics, which is easy to implement and can establish a close connection to nonlinear systems via Koopman theory [30,31]. Many variants have been developed and widely used for different problems [32–38]. However, DMD can not be directly applicable to parameterized PDEs, to extend DMD to parameterized PDEs, several works have been proposed [39–43]. These works consider single or multiple parameters taking values within a certain range and do not consider a wider range of parameters such as initial conditions, boundary conditions, and other parameters belonging to a certain function space. In this paper, we aim to fully take advantage of the powerful operator learning performance of DeepONet, and at the same time, exploit the predictive power of DMD-based method to predict solutions at any time under varying parameters. As in [43], we establish the reduced order modeling by employing DMD and TTD [44], which aims to represent a higher-order tensor through lower-order tensors. Unlike methods that enforce vectorization, TTD permits a more structured arrangement of data, facilitating a more rational representation. We show that the tensor cores are combined in a way consistent with the standard DeepONet output.

In this paper, a TTD and DMD based operator learning model, TDMD-DeepONet, is proposed for high-dimensional parameterized PDEs, which allows rapid prediction of solutions for different parameters (input functions) at any given time. To achieve this, a set of solutions under different parameter values in the training time domain is collected and combined into a tensor, and then the resulting tensor is decomposed by TTD into three components, namely parameter-, space- and time-dependent cores. As in standard DeepONet, we utilize branch net to learn the mapping between parameters and parameter-dependent cores (viewed as branch net outputs). The combinations of space-dependent, as well as time-dependent cores, are used to represent the trunk net, where the DMD is utilized to model the evolution of time-dependent core for prediction. The prediction of the TDMD-DeepONet can be derived by combining the outputs of the branch net and trunk net. Furthermore, the feature-enhanced TDMD-DeepONet, referred to as ETDMD-DeepONet, is proposed to improve the accuracy. In contrast with TDMD-DeepONet, ETDMD-DeepONet incorporates a linear layer within the branch net for residual training. The input to the linear layer is computed by projecting the initial conditions onto the respective space-time basis (trunk net). The output of the linear layer is combined with the original branch net to compute the branch net's final output. The interpolation and extrapolation performance of the proposed methods are evaluated by comparing them with standard DeepONet through several classical examples. The results indicate that the proposed methods achieve accurate predictions, whereas the standard DeepONet is prone to failure in extrapolation. Additionally, the ETDMD-DeepONet method exhibits superior interpolation performance.

This paper is organized as follows, Section 2 provides a brief overview of the methods involved. Section 3 describes the proposed methods in detail, in which the computational complexities and error analysis are also provided. Several examples are provided in Section 4. Finally, Section 5 presents a summary and outlook.

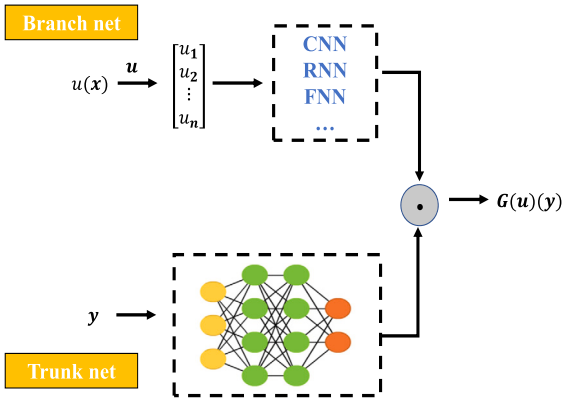

Fig. 1. Architecture of the standard DeepONet.

2. Problem statement and related methods review

2.1. Problem statement

Consider a state variable $s(\boldsymbol{y}, t; u)$ whose dynamics are governed by the following parametrized time-dependent problems

$$\begin{cases} \dfrac{\partial s(\boldsymbol{y}, t; u)}{\partial t} = \mathcal{N}(s(\boldsymbol{y}, t; u)), & (\boldsymbol{y}, t) \in \mathcal{D} \times \mathcal{T}, \\ \mathcal{B}(s(\boldsymbol{y}, t; u)) = b(\boldsymbol{y}, t; u), & (\boldsymbol{y}, t) \in \partial\mathcal{D} \times \mathcal{T}, \\ s(\boldsymbol{y}, t_0; u) = s_0(\boldsymbol{y}; u), & \boldsymbol{y} \in \mathcal{D}, \end{cases} \tag{1}$$

where $\mathcal{N}$ is a nonlinear differential operator, $\boldsymbol{y}$ denotes the spatial coordinates, and $\mathcal{D} \subset \mathbb{R}^d$ is the simulation domain bounded by $\partial\mathcal{D}$. $t \in \mathcal{T} = [t_0, T]$ denotes the time, u is the parameter (input function), where the parameter can be initial conditions, boundary conditions, forced terms, etc., and $\mathcal{B}$ is the boundary differential operator, $s_0(\boldsymbol{x}; u)$ represents the initial state.

To solve the parameterized PDEs above, there are many traditional numerical methods, such as finite difference, finite element, etc. These methods require repeated solving of PDEs for different parameters, which is computationally expensive when the spatial dimensions are very high. Recently, the rapid development of neural networks provided new ways to solve PDEs, in which DeepONets can learn mappings between different spaces, making them highly effective for solving parameterized PDEs. Compared with traditional numerical methods, DeepONets can significantly reduce computational costs, as the primary computational expense occurs during the offline model training phase.

2.2. DeepONet

Inspired by the universal approximation theorem of operators, Lu et al. proposed DeepONet [3] to learn the operators between different spaces, whose architecture consists of two subnetworks with different inputs, one called branch net, which is related to the input function u, and the other called trunk net, which takes the position of the output function as input. DeepONets have been widely used in many fields, they can learn parameterized PDE operators efficiently and flexibly and have higher precision and generalization than fully connected neural networks.

Fig. 1 presents a typical DeepONet structure. To encode the input function $u(\boldsymbol{x})$ into the branch net, function values are computed at a set of discrete positions $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ to form the input vector $\boldsymbol{u} = [u(\boldsymbol{x}_1), u(\boldsymbol{x}_2), \ldots, u(\boldsymbol{x}_n)]$. The forward calculation of the branch net yields the outputs $\boldsymbol{b} = [b_1, b_2, \ldots, b_p]$. By inputting the position coordinate $\boldsymbol{y}$ into the trunk net, the corresponding output is computed and represented as $\boldsymbol{\phi} = [\phi_1, \phi_2, \ldots, \phi_p]$. Finally, the output of the DeepONet is obtained by the inner product of the output of the subnetworks

$$G_\theta(\boldsymbol{u})(\boldsymbol{y}) = \sum_{l=1}^{p} \underbrace{b_l\left(\boldsymbol{u}\left(\boldsymbol{x}_1\right), \boldsymbol{u}\left(\boldsymbol{x}_2\right), \ldots, \boldsymbol{u}\left(\boldsymbol{x}_n\right)\right)}_{\text{branch}} \underbrace{\phi_l(\boldsymbol{y})}_{\text{trunk}}.$$

The structure of DeepONet is highly flexible, with numerous options available for the branch net and trunk net, such as FNN and convolutional neural network (CNN), among others. These greatly enhance DeepONet's applicability in a wide range of scenarios. DeepONet can be utilized not only for learning PDE solution operators but also for modeling and displaying mathematical operators. In this paper, we focus on simulating dynamical systems with different parameters (initial conditions/boundary conditions/source terms) using DeepONet and improving its extrapolation performance with the incorporation of DMD.

2.3. The standard DMD

In this section, a brief introduction to the standard DMD algorithm [45] is given. By discretizing the simulation domain $\mathcal{D}$, the Eq. (1) is transformed into a high-dimensional dynamic system:

$$\begin{cases} \dfrac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{y}, t; \boldsymbol{u}), \\ \boldsymbol{s}(t_0; \boldsymbol{u}) = \boldsymbol{s}_0(\boldsymbol{u}), \end{cases} \tag{2}$$

where $\boldsymbol{s} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ represents the state variable, with $n_1 \times n_2 \times \cdots \times n_d = N$ being the total number of spatially discrete nodes. Here, n_k denotes the number of discrete nodes along the k-th spatial variable and the function $\boldsymbol{f}$ is derived from nonlinear differential operator $\mathcal{N}$.

However, in many cases, the function $f(\boldsymbol{y}, t; \boldsymbol{u})$ describing the dynamic system is unknown, making it challenging to predict the future state of the system. Consequently, identifying the unknown dynamics from the increasingly rich data is a crucial area of focus. In this context, DMD, which aims to linearly approximate the evolution of a system from a set of uniformly time-sampled snapshots to predict states at any given time, has received increasing attention. The algorithm is data-driven, and the initial step is to collect the system's snapshot pairs $\{\boldsymbol{s}(t_k), \boldsymbol{s}(t'_k)\}_{k=1}^{M}$ at a set of sequential times, where $t'_k = t_k + \Delta t$. The collected snapshot pairs are then arranged into two snapshot matrices:

$$\boldsymbol{X} = \left[\boldsymbol{s}(t_1), \boldsymbol{s}(t_2), \cdots, \boldsymbol{s}(t_M)\right], \boldsymbol{Y} = \left[\boldsymbol{s}(t'_1), \boldsymbol{s}(t'_2), \cdots, \boldsymbol{s}(t'_M)\right]. \tag{3}$$

DMD assumes that the state evolution can be described by a linear operator $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, i.e. $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X}$. The operator $\boldsymbol{A}$ can be computed by solving the least squares (LS) problem

$$\boldsymbol{A} = \underset{\hat{\boldsymbol{A}} \in \mathbb{R}^{N \times N}}{\operatorname{argmin}} \frac{1}{M} \sum_{k=1}^{M} \left\| \boldsymbol{s}^{k+1} - \hat{\boldsymbol{A}} \boldsymbol{s}^{k} \right\|^2, \tag{4}$$

where $\boldsymbol{s}^k = \boldsymbol{s}(t_k)$. In fact, to predict the system's state at any time, DMD provides a mode decomposition along with a model for temporal evolution:

$$\boldsymbol{s}^k = \boldsymbol{\Phi} \boldsymbol{\Lambda}^k \boldsymbol{b}, \tag{5}$$

where $\boldsymbol{\Phi}$ is a matrix of the main eigenvectors of $\boldsymbol{A}$, and $\boldsymbol{b} = \boldsymbol{\Phi}^{\dagger} \boldsymbol{s}^1$ is the amplitude. Considering that the system state $\boldsymbol{s}$ is high dimensional, the explicit calculation of $\boldsymbol{A}$ will result in significant computational costs. DMD employs the projection of $\boldsymbol{A}$ to compute the main eigenvalues and eigenvectors of $\boldsymbol{A}$, the steps are as follows:

Step 1 Perform SVD on $\boldsymbol{X}$: $\boldsymbol{X} \approx \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{N \times r}$ and $\boldsymbol{V} \in \mathbb{R}^{M \times r}$ are the left and right singular matrices, $\boldsymbol{\Sigma} \in \mathbb{R}^{r \times r}$ is a diagonal matrix contains the singular values, which are ordered from largest to smallest as $\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_r$, and r is the approximate rank.

Step 2 Project the high-dimensional operator $\boldsymbol{A} \approx \boldsymbol{Y}\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T$ onto subspace of the column vectors of $\boldsymbol{U}$ to derive a low-dimensional matrix $\tilde{\boldsymbol{A}} = \boldsymbol{U}^T\boldsymbol{A}\boldsymbol{U} = \boldsymbol{U}^T\boldsymbol{Y}\boldsymbol{V}\boldsymbol{\Sigma}^{-1}$, whose eigenvalues are the same as the eigenvalues of $\boldsymbol{A}$.

Step 3 Calculate the eigendecomposition of $\tilde{\boldsymbol{A}}$: $\tilde{\boldsymbol{A}}\boldsymbol{W} = \boldsymbol{W}\boldsymbol{\Lambda}$, where the columns of matrix $\boldsymbol{W}$ are the eigenvectors of $\tilde{\boldsymbol{A}}$, and the diagonal elements of matrix $\boldsymbol{\Lambda}$ are the eigenvalues of $\tilde{\boldsymbol{A}}$.

Step 4 Reconstruct the DMD modes: $\boldsymbol{\Phi} = \boldsymbol{Y}\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{W}$.

By introducing continuous eigenvalues $\boldsymbol{\Omega} = \ln(\boldsymbol{\Lambda})/\Delta t$, the continuous time evolution of the system can be expressed as

$$\boldsymbol{s}(t) = \boldsymbol{\Phi} \exp(\boldsymbol{\Omega} t) \boldsymbol{b}. \tag{6}$$

Let $\Phi_{\Delta t} : \mathbb{R}^N \to \mathbb{R}^N$ denote the flow map relating the state $u(t)$ to $u(t + \Delta t)$, that is

$$\boldsymbol{s}(t + \Delta t) = \Phi_{\Delta t}(\boldsymbol{s}(t)),$$

we have

$$\boldsymbol{s}^{n+1} = \Phi_{\Delta t}(\boldsymbol{s}^n).$$

And there is the following lemma:

Lemma 1. *[3] Define the global truncation error*

$$\begin{aligned} \mathrm{e}^n &= \widetilde{\boldsymbol{s}}^n - \mathbf{s}^n_{DMD} \\ &= \boldsymbol{A}^{n-1}\mathbf{s}_0 - \boldsymbol{\Phi}\boldsymbol{\Lambda}^n\mathbf{b} \end{aligned} \tag{7}$$

Then, for $n \geq m$,

$$\|\mathbf{e}^n\|_2 < \|\mathbf{\Phi}\|_2 \left\|\mathbf{\Phi}^{-1}\right\|_2 \left[\|\mathbf{e}^m\|_2 + (n-m)\varepsilon_m\right],$$

where the constant ε_m depends only on the number of snapshots m.

DMD is parameter-dependent, which means that for different parameters $\boldsymbol{u}$, it is required to construct different DMD models for the time-based dynamic modeling under the corresponding parameters. In this paper, to apply DMD to parameterized PDEs, the TTD is utilized to separate the parameter-spatial-temporal snapshots into different cores to build the DMD model, which is introduced in the next section.

2.4. TTD

The representation and computation of higher-order tensors is a challenging problem, and numerous tensor decomposition algorithms have been developed to address this issue [46–48,44]. The TTD is a notable approach that decomposes a d-dimensional tensor $\mathcal{K} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ into d reduced and concatenated forms of second-order and third-order matrices (called cores of TTD):

$$\boldsymbol{K} = \sum_{\alpha_0=1}^{r_0} \cdots \sum_{\alpha_d=1}^{r_d} \boldsymbol{G}_1(\alpha_0, :, \alpha_1) \otimes \boldsymbol{G}_2(\alpha_1, :, \alpha_2) \otimes \cdots \otimes \boldsymbol{G}_d(\alpha_{d-1}, :, \alpha_d), \tag{8}$$

where $\boldsymbol{G}_k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$, with $\{r_i\}_{i=0}^d$ represent the ranks of TTD and $r_0 = r_d = 1$. Accordingly, each element of $\boldsymbol{K}$ can be represented as

$$\boldsymbol{K}(i_1, \cdots, i_d) = \sum_{\alpha_0=1}^{r_0} \cdots \sum_{\alpha_d=1}^{r_d} \boldsymbol{G}_1(\alpha_0, i_1, \alpha_1) \boldsymbol{G}_2(\alpha_1, i_2, \alpha_2) \cdots \boldsymbol{G}_d(\alpha_{d-1}, i_d, \alpha_d). \tag{9}$$

The TT-SVD algorithm [44] can be used to realize TTD with $d-1$ SVD. To achieve the optimal low-rank approximation of $\mathcal{K}$, the truncated rank r_k is determined through a pre-defined accuracy $\boldsymbol{\varepsilon} = [\varepsilon_1, \varepsilon_2, ..., \varepsilon_{d-1}]$ during the SVD. And the approximation error of TT-SVD is bounded in the Frobenius norm:

$$\|\mathcal{K} - \boldsymbol{K}\|_F^2 \leq \sum_{k=1}^{d-1} \varepsilon_k^2. \tag{10}$$

3. Proposed methods

The limited extrapolation capability of neural networks constrains the ability of the trained DeepONet to provide reliable performance outside the training time domain, thereby limiting its application in real-time prediction of dynamic systems. To address this limitation, we first propose an operator learning algorithm that combines TTD and DMD, called TDMD-DeepONet, which is based on TTD and exploits the excellent predictive capability of DMD. It can extend DMD to the prediction of parameterized PDEs, and at the same time enables DeepONet to predict the dynamical behaviors in the unseen time domain.

3.1. TDMD-DeepONet

Sample N_u input functions from the parameter space, and for each input function, collect the state vector $\boldsymbol{s}$ over the n_t uniformly sampled time instants. All resulting snapshots are merged into a data tensor $\boldsymbol{X}_p \in \mathbb{R}^{N_u \times n_1 \times n_2 \times \cdots \times n_d \times n_t}$. To implement the separation of variables and the low-rank approximation of $\boldsymbol{X}_p$, the TTD is applied to $\boldsymbol{X}_p$, representing $\boldsymbol{X}_p$ as the $d+2$ cores $\{\boldsymbol{G}_k\}_{k=1}^{d+2}$ in a reduced and concatenated form, where each element of $\boldsymbol{X}_p$ is approximated as

$$\begin{aligned}
\boldsymbol{X}_p(i_1, \cdots, i_{d+2}) &\approx \sum_{\alpha_0=1}^{r_0} \cdots \sum_{\alpha_{d+2}=1}^{r_{d+2}} \boldsymbol{G}_1(\alpha_0, i_1, \alpha_1) \boldsymbol{G}_2(\alpha_1, i_2, \alpha_2) \cdots \boldsymbol{G}_{d+2}(\alpha_{d+1}, i_{d+1}, \alpha_{d+2}) \\
&= \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_{d+1}=1}^{r_{d+1}} \boldsymbol{G}_1(i_1, \alpha_1) \boldsymbol{G}_2(\alpha_1, i_2, \alpha_2) \cdots \boldsymbol{G}_{d+2}(\alpha_{d+1}, i_{d+1}) \\
&= \sum_{\alpha_1=1}^{r_1} \boldsymbol{G}_1(i_1, \alpha_1) \sum_{\alpha_2=1}^{r_2} \cdots \sum_{\alpha_{d+1}=1}^{r_{d+1}} \boldsymbol{G}_2(\alpha_1, i_2, \alpha_2) \cdots \boldsymbol{G}_{d+2}(\alpha_{d+1}, i_{d+1}).
\end{aligned} \tag{11}$$

The core $\boldsymbol{G}_1 \in \mathbb{R}^{N_u \times r_1}$ is parameter-dependent, $\boldsymbol{G}_i \in \mathbb{R}^{r_{i-1} \times n_i \times r_i} (i = 2, 3, ...d+1)$ are space-dependent, and $\boldsymbol{G}_{d+2} \in \mathbb{R}^{r_{d+1} \times n_t}$ is time-dependent. Let

$$g_{\alpha_1}(i_2, \cdots, i_{d+2}) = \sum_{\alpha_2=1}^{r_2} \cdots \sum_{\alpha_{d+1}=1}^{r_{d+1}} \boldsymbol{G}_2(\alpha_1, i_2, \alpha_2) \cdots \boldsymbol{G}_{d+2}(\alpha_{d+1}, i_{d+1}), \tag{12}$$

which can be seen as the output of trunk net, then Eq. (11) can be written as

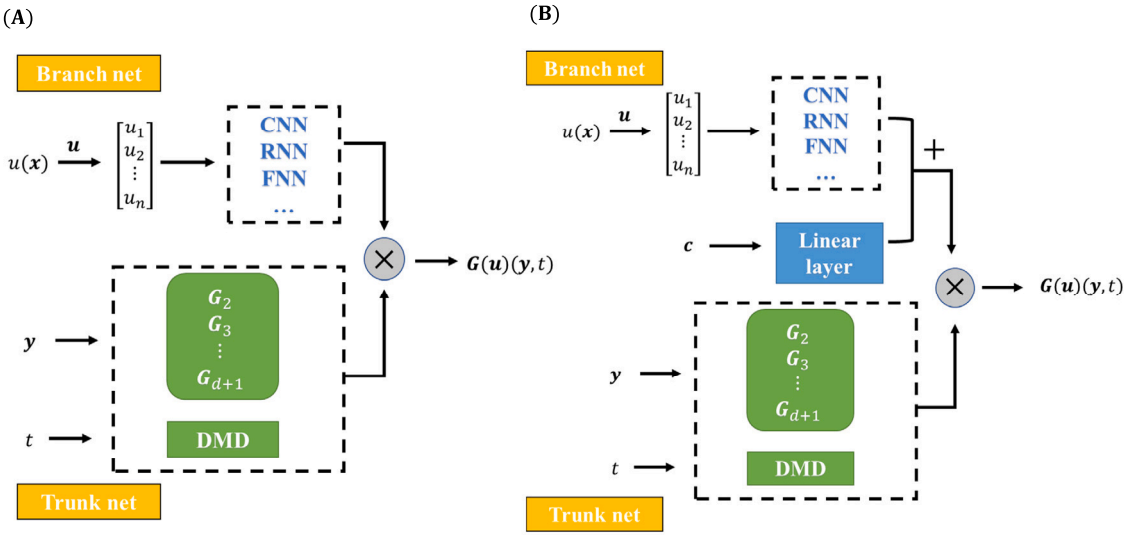

Fig. 2. Architectures of (A) TDMD-DeepONet and (B) ETDMD-DeepONet.

$$\boldsymbol{X}_p(i_1,\cdots,i_{d+2}) \approx \sum_{\alpha_1=1}^{r_1} \boldsymbol{G}_1(i_1,\alpha_1) g_{\alpha_1}(i_2,\cdots,i_{d+2}), \tag{13}$$

which is consistent with the output of the DeepONet. Similar to the idea of POD-DeepONet, by using a branch net to learn the decomposition coefficients when training the DeepONet and using the pre-computed cores as the output of the trunk net, one can estimate the solutions under different parameters. However, it should be noted that this strategy is not suitable for predicting solutions outside the training time domain. Consequently, DMD is employed to estimate the time-dependent core, which is then used to predict the system's state at any given time. Specifically, a continuous DMD model $\boldsymbol{g}_{DMD}(t)$ can be built based on the core $\boldsymbol{G}_{d+2}$ that has been computed. In this case, instead of performing DMD directly on the original state vector $\boldsymbol{s}$, the data matrix $\boldsymbol{X}$ and $\boldsymbol{Y}$ are defined as:

$$\boldsymbol{X} = \left[\boldsymbol{g}_{d+2}^1, \boldsymbol{g}_{d+2}^2, \cdots, \boldsymbol{g}_{d+2}^{n_t-1}\right], \boldsymbol{Y} = \left[\boldsymbol{g}_{d+2}^2, \boldsymbol{g}_{d+2}^3, \cdots, \boldsymbol{g}_{d+2}^{n_t}\right], \tag{14}$$

where $\boldsymbol{g}_{d+2}^k \in \mathbb{R}^{r_{d+1}\times 1}$ is the k-th column of $\boldsymbol{G}_{d+2}$. In this case, according to the assumptions of the DMD algorithm, there is the following relation:

$$\boldsymbol{g}_{d+2}^{k+1} = \tilde{\boldsymbol{A}} \boldsymbol{g}_{d+2}^k, \tag{15}$$

$\tilde{\boldsymbol{A}}$ is the linear operator to be computed, and since r_{d+1} is quite small indeed, the eigenvalues $\{\tilde{\lambda}_k\}_{k=1}^{r_{d+1}}$ and eigenvectors of $\tilde{\boldsymbol{A}}$ can be computed without truncation, and finally the following DMD model is obtained:

$$\boldsymbol{g}_{DMD}(t) = \tilde{\boldsymbol{\Phi}} \exp(\tilde{\boldsymbol{\Omega}} t) \tilde{\boldsymbol{b}}, \tag{16}$$

where $\tilde{\boldsymbol{b}} = \tilde{\boldsymbol{\Phi}}^\dagger \boldsymbol{g}_{d+2}^1$ and $\tilde{\boldsymbol{\Omega}}$ is a diagonal matrix with diagonal elements $\tilde{\omega}_k = \ln(\tilde{\lambda}_k)/\Delta t$.

To predict the system's state at any time for any parameter value, a FNN $\boldsymbol{b}(\boldsymbol{u})$ is trained to map the parameter to the parameter-dependent core using the training data $\{\boldsymbol{u}_k, \boldsymbol{g}_1^k\}_{k=1}^{N_u}$, where $\boldsymbol{g}_1^k$ is the k-th row of $\boldsymbol{G}_1$. And before training the neural network, the inputs are normalized to facilitate the training process, during which the variance $var(\boldsymbol{g}_1)$ and mean $mean(\boldsymbol{g}_1)$ of the cores $\{\boldsymbol{g}_1^k\}_{k=1}^{N_u}$ are computed. Subsequently, the final output of the network is rescaled as

$$\boldsymbol{b}(\boldsymbol{u}) = \boldsymbol{b}(\boldsymbol{u}) \times var(\boldsymbol{g}_1) + mean(\boldsymbol{g}_1).$$

For a new parameter $\boldsymbol{u}^*$ and time t^*, one only needs to input $\boldsymbol{u}^*$ into the trained branch net to get the output $\boldsymbol{b}(\boldsymbol{u}^*)$ and use the established DMD model to compute $\boldsymbol{g}_{DMD}(t^*)$, and the corresponding solution $\boldsymbol{s}(\boldsymbol{u}^*, t^*)$ can be approximated in the following form:

$$\hat{\mathrm{s}}(\boldsymbol{u}^*, t^*) = \boldsymbol{b}(\boldsymbol{u}^*) \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+1} \otimes \boldsymbol{g}_{DMD}(t^*). \tag{17}$$

The left side of Fig. 2 shows the structure of TDMD-DeepONet, with the details of the specific procedure provided in Algorithm 1.

Remark 1. In contrast with the standard DeepONet, the proposed TDMD-DeepONet does not train the trunk net, instead, it performs the TTD on the training data and constructs the DMD model, which requires a computational complexity of $\mathcal{O}(N_u^2 n^d n_t + \hat{r}^2 n^{d+1} n_t)$ and $\mathcal{O}(n_t r_{d+1}^2)$, respectively, where $\hat{r} = \max\{r_0, \cdots, r_{d+1}\}$, $n = \max\{n_1, \cdots, n_d\}$. Predicting the solution for a new spatial-temporal point

Algorithm 1 TDMD-DeepONet.

Offline Stage

Input: Training parameter set $\{\boldsymbol{u}_k\}_{k=1}^{N_u}$, $d+2$ dimensional tensor $\boldsymbol{X}_p \in \mathbb{R}^{N_u \times n_1 \times n_2 \times \cdots \times n_d \times n_t}$, prescribed accuracy $\boldsymbol{\varepsilon} = \{\varepsilon_1, \cdots, \varepsilon_{d+1}\}$.

Output: Trained branch net $\boldsymbol{b}(\boldsymbol{u})$, DMD model $\boldsymbol{g}_{DMD}(t)$.

1: Perform a TTD on $\boldsymbol{X}_p$ as

$$\boldsymbol{X}_p \approx \sum_{\alpha_0=1}^{r_0} \cdots \sum_{\alpha_d=1}^{r_{d+2}} \boldsymbol{G}_1(\alpha_0, :, \alpha_1) \otimes \boldsymbol{G}_2(\alpha_1, :, \alpha_2) \otimes \cdots \otimes \boldsymbol{G}_{d+2}(\alpha_{d+1}, :, \alpha_{d+2}).$$

2: Construct and train a FNN $\boldsymbol{b}(\boldsymbol{u})$ for representing the parameter-dependent core as a function of the parameter $\boldsymbol{u}$ based on the training data $\{\boldsymbol{u}_k, \boldsymbol{g}_1^k\}_{k=1}^{N_u}$.

3: Built a DMD model $\boldsymbol{g}_{DMD}(t)$ based on matrices $\boldsymbol{X} = \left[\boldsymbol{g}_{d+2}^1, \boldsymbol{g}_{d+2}^2, \cdots, \boldsymbol{g}_{d+2}^{n_t-1}\right]$ and $\boldsymbol{Y} = \left[\boldsymbol{g}_{d+2}^2, \boldsymbol{g}_{d+2}^3, \cdots, \boldsymbol{g}_{d+2}^{n_t}\right]$ for predicting time-dependent core at different times.

Online Stage

Input: Parameter $\boldsymbol{u}^*$ and time t^* corresponding to the solution to be predicted.

Output: Predicted solution $\hat{\boldsymbol{s}}(\boldsymbol{u}^*, t^*)$.

1: Evaluate the well-trained branch net at the new input parameter $\boldsymbol{u}^*$ to predict the core $\boldsymbol{b}(\boldsymbol{u}^*)$.

2: Evaluate the built DMD at the new time t^* to predict the core $\boldsymbol{g}_{DMD}(t^*)$.

3: Combine the predicted cores $\boldsymbol{b}(\boldsymbol{u}^*)$ and $\boldsymbol{g}_{DMD}(t^*)$ to compute the approximate state at $(\boldsymbol{u}^*, t^*)$:

$$\hat{\boldsymbol{s}}(\boldsymbol{u}^*, t^*) = \boldsymbol{b}(\boldsymbol{u}^*) \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+1} \otimes \boldsymbol{g}_{DMD}(t^*).$$

involves computing the output Eq. (12) of the trunk net, different from the forward computation of the neural network, the proposed method requires the computation of the product of $r_1 \times r_2 \cdots \times r_{d+1}$ times.

3.2. Feature enhanced TDMD-DeepONet: ETDMD-DeepONet

In TDMD-DeepONet, the spatial cores obtained by performing TTD on the training tensor and the time core predicted by DMD are used as the trunk net, and the neural network is taken as the branch net to learn the parameter core (coefficient), thereby enabling to predict the system response at any time under different parameters. Different from the standard DeepONet, which employs neural networks for the trunk net trained simultaneously with the branch net, the proposed method uses a pre-determined trunk net. In many applications, the parameters are related to the distribution of the system state, including initial and boundary conditions. In this situation, the parameters are also informative with regard to the output of the branch net, thereby enhancing the predictive capabilities. We discuss here the case where the initial condition $\boldsymbol{s}_0$ is known, i.e. $\boldsymbol{u} = \boldsymbol{s}_0$ and other cases are similar. Specifically, the construction of TDMD-DeepONet shows that $\boldsymbol{s}_0$ can be approximated as follows:

$$\boldsymbol{s}_0(\boldsymbol{u}) \approx \boldsymbol{b}(\boldsymbol{u}) \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+1} \otimes \boldsymbol{g}_{DMD}(t_0).$$

Let $\boldsymbol{G} = \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+1} \otimes \boldsymbol{g}_{DMD}(t_0)$, then $\boldsymbol{G}$ is known, and by solving the following minimization problem:

$$\boldsymbol{c} = \underset{\hat{\boldsymbol{c}} \in \mathbb{R}^{r_1}}{\operatorname{argmin}} \left\| \boldsymbol{s}_0 - \hat{\boldsymbol{c}} \otimes \boldsymbol{G} \right\|^2, \tag{18}$$

an approximation $\boldsymbol{c}$ for $\boldsymbol{b}$ can be computed.

However, since $\boldsymbol{c}$ is only computed using the initial conditions, it may not accurately represent the solution over the entire time domain. Nonetheless, it can still be considered a low-fidelity coefficient or an enhanced feature. Consequently, a linear layer l is incorporated into the branch net, with $\boldsymbol{c}$ is input and $l(\boldsymbol{c})$ is output. The final output of the branch net is denoted $\boldsymbol{b}(\boldsymbol{u}, \boldsymbol{c}) = \boldsymbol{b}(\boldsymbol{u}) + l(\boldsymbol{c})$. The branch net structure is illustrated in the right figure of Fig. 2. By training both networks simultaneously and learning the relationship between the different inputs, $\boldsymbol{c}$ is made an additional feature and the corresponding output is used to help in prediction.

Remark 2. ETDMD-DeepONet differs from TDMD-DeepONet by adding a linear layer to the branch net, where $\boldsymbol{c}$ is seen as an augmented feature, an idea that has appeared in much of the literature, and the complexity required for $\boldsymbol{c}$ is $\mathcal{O}(n^d {r_1}^2)$. Considering that $\boldsymbol{c}$ is not guaranteed to be applicable globally and to save training costs, we take only a simple linear layer here, which can be chosen differently depending on the problem.

3.3. Error analysis

In this subsection, an error analysis of the proposed method is presented. For a new parameter $\boldsymbol{u}^*$, let $\boldsymbol{s}^n \in \mathbb{R}^N$ be the reference solution at time t^n, while $\hat{\boldsymbol{s}}^n$ represents the corresponding solution approximated by the proposed model. The approximation error can be computed as

$$
\begin{aligned}
\epsilon^n &= \|\boldsymbol{s}^n - \hat{\boldsymbol{s}}^n\|_2^2 \\
&= \|\boldsymbol{s}^n - \boldsymbol{b}(\boldsymbol{u}^*) \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+1} \otimes \boldsymbol{g}_{DMD}^n\|_2^2 \\
&\leq \|\boldsymbol{u}^n - \boldsymbol{G}_1^* \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+2}^n\|_2^2 + \|\boldsymbol{G}_1^* \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+2}^n - \boldsymbol{b}(\boldsymbol{u}^*) \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+2}^n\|_2^2 \\
&+ \|\boldsymbol{b}(\boldsymbol{u}^*) \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+2}^n - \boldsymbol{b}(\boldsymbol{u}^*) \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+1} \otimes \boldsymbol{g}_{DMD}^n\|_2^2 \\
&\triangleq \epsilon_{TT}^n + \epsilon_{nn}^n + \epsilon_{DMD}^n,
\end{aligned} \tag{19}
$$

where $\boldsymbol{G}_1^* \otimes \boldsymbol{G}_2 \otimes \cdots \otimes \boldsymbol{G}_{d+2}^n$ is the TT-format of $\boldsymbol{s}^n$. Therefore, the approximation error, denoted by ϵ^n, is decomposed into three components: ϵ_{TT}^n, which is the error resulting from the TTD and bounded by Eq. (10), ϵ_{nn}^n, which is the error associated with the branch net, and ϵ_{DMD}^n, which is the error caused by the DMD.

Denote $\boldsymbol{g}^n = \boldsymbol{G}_{d+2}^n$, and from Lemma 1, the error of DMD can then estimated as follows:

$$
\begin{aligned}
\epsilon_{DMD}^n &= \|\boldsymbol{g}^n - \boldsymbol{g}_{DMD}^n\|^2 \\
&= \|\tilde{\boldsymbol{\Phi}}_{\Delta t}(\boldsymbol{g}^{n-1}) - \tilde{\boldsymbol{A}}\boldsymbol{g}^{n-1} + \tilde{\boldsymbol{A}}\boldsymbol{g}^{n-1} - \tilde{\boldsymbol{\Phi}}\Lambda^n \mathbf{b}\|^2 \\
&\leq \|\tilde{\boldsymbol{\Phi}}_{\Delta t}(\boldsymbol{g}^{n-1}) - \tilde{\boldsymbol{A}}\boldsymbol{g}^{n-1}\|^2 + \|\tilde{\boldsymbol{A}}\boldsymbol{g}^{n-1} - \tilde{\boldsymbol{\Phi}}\Lambda^n \mathbf{b}\|^2 \\
&< \|\tilde{\boldsymbol{\Phi}}_{\Delta t} - \tilde{\boldsymbol{A}}\|_{L^\infty} + \|\tilde{\boldsymbol{\Phi}}\|_2 \left\|\tilde{\boldsymbol{\Phi}}^{-1}\right\|_2 \left[\left\|\mathbf{e}^{n_t-1}\right\|_2 + (n - n_t + 1)\varepsilon_{n_t-1}\right].
\end{aligned} \tag{20}
$$

Remark 3. It is clear that the more snapshots collected and the more accurate the approximation of flow map $\tilde{\boldsymbol{\Phi}}_{\Delta t}$ by the linear operator $\tilde{\boldsymbol{A}}$, the more accurate the approximation of the DMD will be. In this work, we have focused on the application of standard DMD, there are also many variants of DMD. In [49], the authors update the DMD model online over time, enabling real-time updating of the system dynamics approximation as new data becomes available. A number of studies have mapped state variables to observation variable by introducing a nonlinear observation function and modeling the evolution of observation variable by DMD in order to predict state variables. In [30], the observation function is derived from a linear combination of functions in a given dictionary. The observation function is defined implicitly by the feature map associated with a user-defined kernel function in [50]. In [51], autoencoder is used to learn the optimal dictionary function. In [52], extended DMD with invertible dictionary learning is proposed, which utilize invertible neural networks to learn dictionary functions. Another category of methods adds constraints based on the laws of physics when certain priori information is known [53,54]. The application of the various variants of DMD within the presented framework may enhance the predictive accuracy of the proposed methodologies.

4. Numerical examples

In this section, the performance of the proposed models, TDMD-DeepONet and ETDMD-DeepONet, are validated using several numerical examples. A consistent prescribed accuracy of $\varepsilon_i = 10^{-8}(i = 1, 2, ..., d+1)$ for the TT-SVD algorithm, is maintained in all examples. The training set is sampled from the time interval $\mathcal{T}_{\text{train}} \triangleq [t_0, T^*]$, where $T^* < T$. Prediction errors for $t \in \mathcal{T}_{\text{train}}$ reflect interpolation performance, while errors for $t \in \mathcal{T}_{\text{test}} \triangleq [T^*, T]$ reflect extrapolation performance. The accuracy of both interpolation and extrapolation is evaluated to assess the models' performance. This is quantified using the relative error, which is calculated as follows:

$$
\varepsilon_{\text{RE}}(\boldsymbol{u}, t_k) = \frac{\|\hat{\boldsymbol{s}}(t_k; \boldsymbol{u}) - \boldsymbol{s}_{\text{ref}}(t_k; \boldsymbol{u})\|_2}{\|\boldsymbol{s}_{\text{ref}}(t_k; \boldsymbol{u})\|_2}, \quad t_k \in \mathcal{T}. \tag{21}
$$

The training dataset is divided into two distinct portions: 95% is utilized for training TDMD-DeepONet and ETDMD-DeepONet, while the remaining 5% is used for validation. TDMD-DeepONet and ETDMD-DeepONet are iterated for 1,000 epochs, after which training is terminated when the best accuracy is not achieved for 100 consecutive epochs. And the structure of TDMD-DeepONet and ETDMD-DeepONet is the same as that of DeepONet's branch net, except that the final output layer dimension is r_1. The input functions for each problem are summarized in Table 1, where $\mathcal{G}(0, \exp\left(-\frac{\|x_1 - x_2\|_2^2}{2\ell^2}\right))$ and $\mathcal{R}(0, \sigma^2(-\Delta + 25I)^{-2})$ represent Gaussian random fields with a radial basis function kernel and a Riesz kernel, respectively. Here, Δ denotes the Laplacian operator, and I is the identity matrix.

4.1. Relaxation kinetics equation

The following equation that describes the one-dimensional relaxation phenomenon is first used to test the performance of the proposed methods:

$$
\begin{cases}
\partial_t s(x,t) = -ks(x,t), \quad (x,t) \in [-1,1] \times [0,2], \\
u(x = \pm 1) = 0,
\end{cases} \tag{22}
$$

where $k = 1$, and the analytical solution for this problem is $s = -e^{kt} s_0(x)$, which is contingent upon the initial condition $s_0(x) = s(x, t = 0)$. In this problem, our goal is to learn the mapping from initial condition to solution: $s_0(x) \longmapsto s(x,t)$. For relaxation kinetics,

Table 1
A description of the input function for each problem.

Case	Input Function	Space	Length Scales
4.1	Initial condition $s_0(x)$	$s_0(x) \sim \mathcal{G}(0, \exp\left(-\frac{\|x_1-x_2\|_2^2}{2\ell^2}\right))$	$l = 0.2$
4.2	Source term $a(x)$	$a(x) \sim \mathcal{G}(0, \exp\left(-\frac{\|x_1-x_2\|_2^2}{2\ell^2}\right))$	$l = 0.2$
4.3	Initial condition $s_0(x)$	$s_0(x) \sim \mathcal{R}\left(0, 625(-\Delta + 25I)^{-2}\right)$	-
4.4	Initial field $h_2(x)$	$h_2(x) \sim \mathcal{G}(0, 0.15\exp\left(-\frac{\|x-x'\|_2^2}{2\ell_x^2} - \frac{\|y-y'\|_2^2}{2\ell_y^2}\right))$	$l_x = 0.3$ $l_y = 0.4$
4.5	Vorticity field at time [0,0.45] determined by initial condition	$w_0(x) \sim \mathcal{R}\left(0, 7^{3/2}(-\Delta + 49I)^{-2.5}\right)$	-

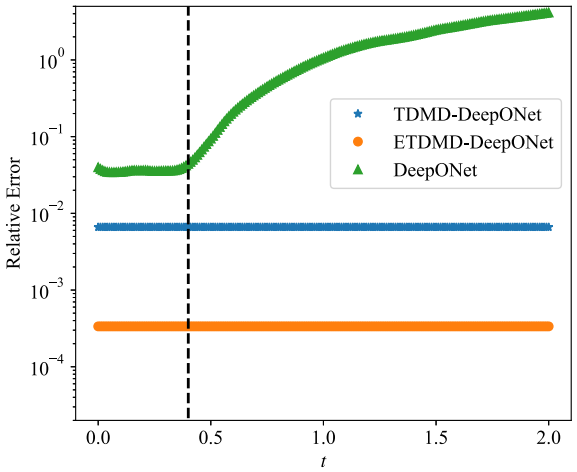

Fig. 3. The relative errors of different models in time region $[0, 2]$ for Example 4.1.

the DMD has the ability to accurately capture the dynamic evolution, thus the errors in TDMD-DeepONet and ETDMD-DeepONet mainly come from the estimation error of the brunch net. Therefore, this example is used to illustrate how the proposed method performs when the spatiotemporal evolution can be modeled accurately. To train the three different models, $N_u = 1000$ different initial conditions obeying a Gaussian distribution are generated and evaluated at $N_x = 100$ equally spaced points as input to the branch net. The reference solutions $s(x)$ are derived from the analytical solution $s = -e^{kt}s_0(x)$, with the temporal domain uniformly discretized into $N_t = 201$ points. The snapshots over the time interval $[0, 0.4]$ are used for training, while $[0, 2]$ is reserved for testing. This configuration generates 1000 snapshots, defined on a grid of 100 spatial points and 41 uniformly distributed spatiotemporal points, which are utilized as inputs for both TTD and model training. For the standard DeepONet, $N_x \times n_t = 100 \times 41$ uniformly distributed data pairs in the space-time domain are chosen as inputs to the trunk network, resulting in 4100000 $(1000 \times 100 \times 41)$ training points, where the network structure is set up in [55].

The mean L^2 relative errors estimated by the different methods in $N_{test} = 200$ different initial conditions are displayed in Fig. 3. It can be seen that TDMD-DeepONet and ETDMD-DeepONet have smaller errors compared to DeepONet. In particular, the errors of TDMD-DeepONet and ETDMD-DeepONet do not increase over time when DMD can accurately capture the dynamical evolution. In contrast, due to the limited extrapolation capability of the neural network, the error of DeepONet increases rapidly outside the training time range. Moreover, ETDMD-DeepONet exhibits the highest accuracy, with an error of $\sim 1e-4$, which illustrates the feasibility of incorporating the auxiliary coefficient $\boldsymbol{c}$ into the brunch net. Furthermore, two different initial conditions are randomly selected from the test set and the solutions estimated by the various methods are plotted at three time points in Fig. 4. It can be observed that both TDMD-DeepONet and ETDMD-DeepONet accurately capture the structure of the solution, demonstrating consistency with the reference solution. However, outside of the training time range, DeepONet's performance is less satisfactory, with a larger discrepancy with the reference solution. This example demonstrates that the proposed method outperforms DeepONet at all times, which illustrates the superiority of introducing the TTD.

4.2. Diffusion-reaction system with a source term

In the second problem, we consider a reaction-diffusion system with a source term $a(x)$ of the following form:

$$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + a(x), \quad x \in [0, 1], t \in [0, 1],$$

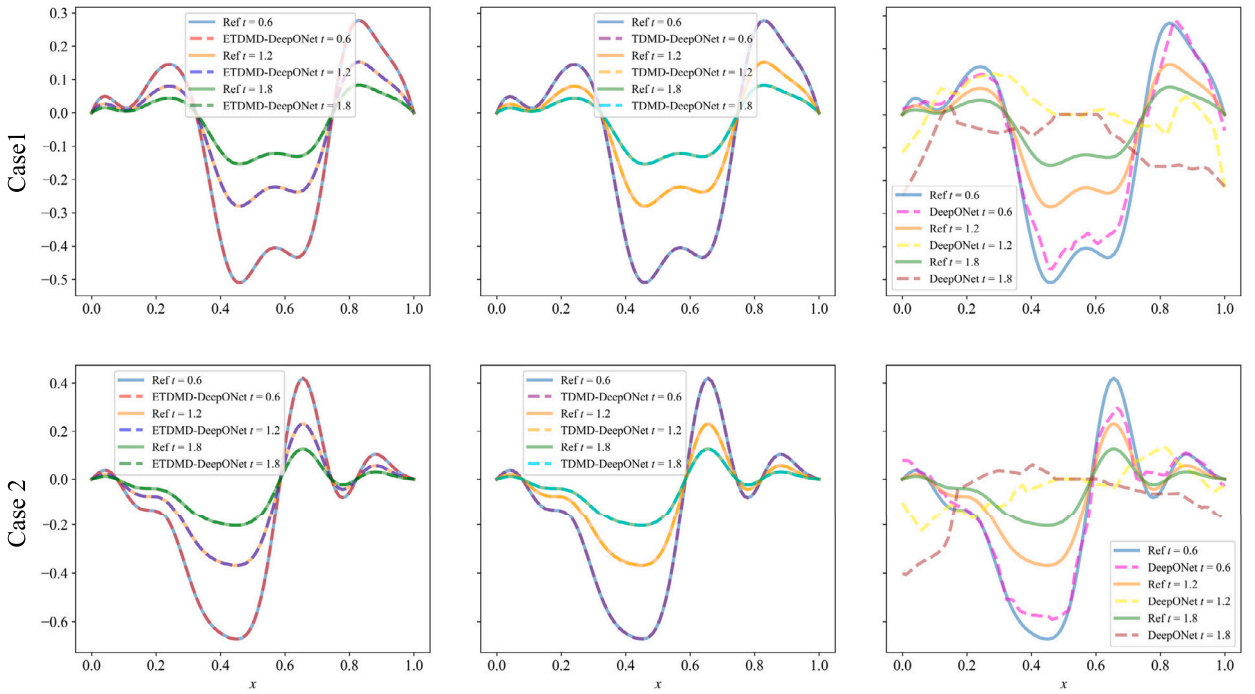

Fig. 4. Comparison between the reference solutions and the predicted solutions of (**Left**) ETDMD-DeepONet, (**Middle**) TDMD-DeepONet, and (**Right**) DeepONet for two initial conditions at the test time instants $t = 0.6, 1.2, 1.8$.

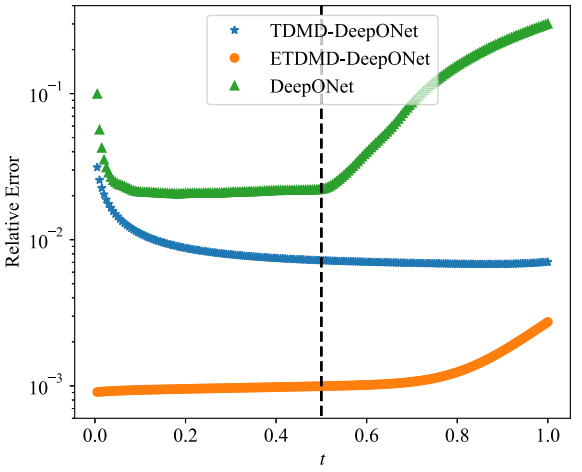

Fig. 5. The relative errors of different models in time region for Example 4.2.

with zero initial/boundary conditions, where $D = 0.2$ is the diffusion coefficient, and $k = 0.01$ is the reaction rate. In this problem, the mapping to be learned is from the source term $a(x)$ to the solution $s(x,t)$. To obtain training and test data, 1100 different source terms are sampled from a Gaussian random field with correlation length $l = 0.2$. The corresponding solutions are computed using a second-order implicit finite difference method on a spatial-temporal grid with a resolution of 100×201. In total, $N_u = 1000$ data points are used for training, and 100 data points are reserved for testing. The training time range is $[0, 0.5]$, while the test time range extends to $[0, 1]$. In order to perform the DMD and compute the auxiliary variable $\boldsymbol{c}$, we computed the solution at the first time point $t = 0.005$ as the initial condition, considering that the initial condition is 0. And the selected $N_x \times n_t = 100 \times 101$ spatial-temporal data pairs used to implement trunk net training for DeepONet, hence, the train dataset size is equal to $N_u \times N_x \times n_t$, with the network structure is same as [56].

The mean L^2 relative errors of the three models on $N_{test} = 100$ different test initial conditions are reported in Fig. 5. It is observed that the proposed TDMD-based framework allows for the achievement of smaller errors compared to DeepONet. Moreover, ETDMD-DeepONet consistently exhibits the smallest error. Furthermore, within the training time range, the errors of both DeepONet and TMDM-DeepONet demonstrate a tendency to decrease and then level off. However, the incorporation of the auxiliary variable $\boldsymbol{c}$

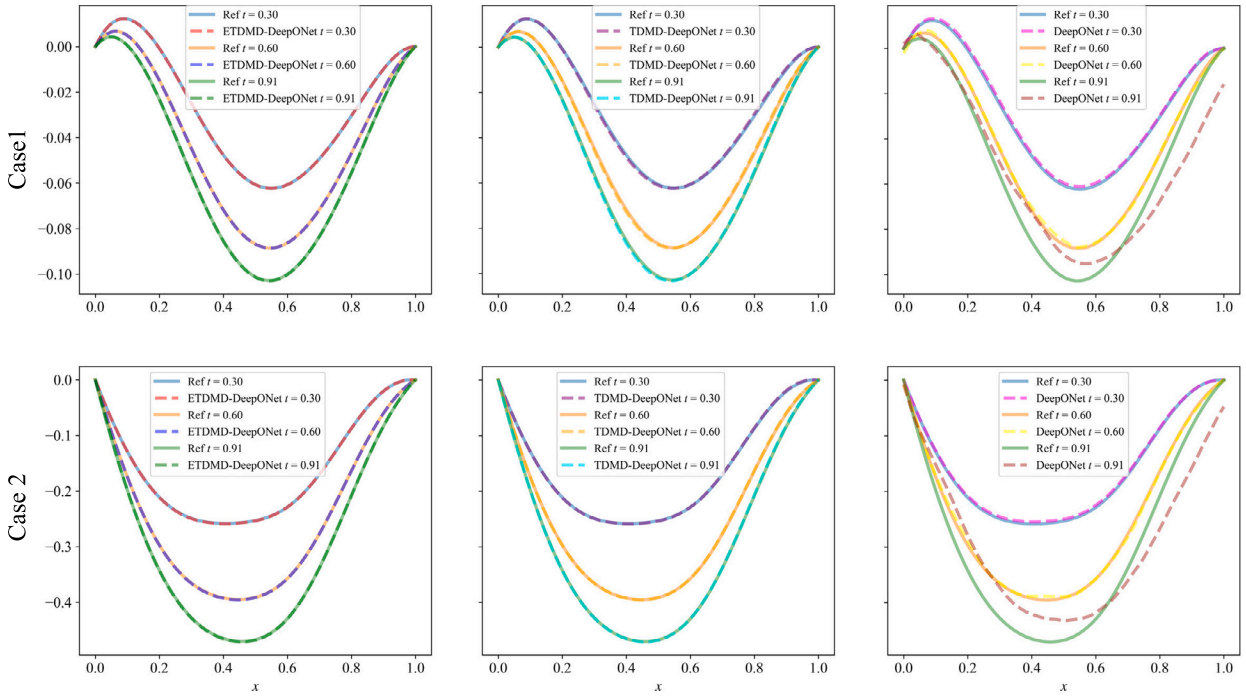

Fig. 6. Comparison between the reference solutions and the predicted solutions of (**Left**) ETDMD-DeepONet, (**Middle**) TDMD-DeepONet, and (**Right**) DeepONet for two initial conditions at the test time instants $t = 0.3, 0.6, 0.91$.

results in a stable performance for ETDMD-DeepONet, with the model achieving an exceptionally high level of accuracy, with an error in the range of $\sim 1e-4$. As anticipated, outside of the training time range, DeepONet's error increases immediately. However, ETDMD-DeepONet and TDMD-DeepONet continue to maintain reliable accuracy. Furthermore, in this example, DMD exhibits accurate extrapolation capability, and the error of TDMD-DeepONet is primarily attributable to the prediction of parameter core. Consequently, the accuracy of TDMD-DeepONet remains stable outside the training time range. With the evolution of time, the error of DMD became the dominant factor in the error of ETDMD-DeepONet. Hence, there is a trend of increasing error, but it remains the most accurate. In Fig. 6, we plot the reference solutions at three different times at the two random parameters and the solutions estimated by the different methods. The curves estimated by ETDMD-DeepONet and TDMD-DeepONet coincide with the reference solution at all test time. The DeepONet estimate is acceptable within the training time range. However, the difference between the DeepONet estimated solution and the reference solution becomes apparent at $t = 0.91$, which is due to the poor extrapolation performance of the data-driven neural network.

4.3. Burgers' equation

Let us now consider the one-dimensional Burgers' equation:

$$\frac{\partial s}{\partial t} + s\frac{\partial s}{\partial x} = \nu\frac{\partial^2 s}{\partial x^2}, \quad x \in (0,1), t \in (0,1],$$

with the viscosity coefficient ν set to 0.01. The aim of this problem is to learn the mapping of the initial condition $s(x,0) = s_0(x)$ to the solution operator $s(x,t)$, i.e.,

$$\mathcal{G} : s_0(x) \mapsto s(x,t).$$

As in [18], the initial conditions are sampled from a Gaussian random field. The reference solutions are computed using a split-step method, with the heat equation solved exactly in Fourier space and the nonlinear term advanced using a fine forward Euler method. Simulations are performed on a spatial resolution of $2^{13} = 8192$ and a temporal resolution of 201 points. From this, $N_t = 101$ uniformly distributed time instants are collected at $N_x = 101$ equally spaced spatial grid points for each simulation. The training time range is $[0, 0.6]$, thus $n_t = 61$ time points are used for training. A total of $N_u = 1000$ initial values are selected for training the different models. A set of 250,000 data points are used to train the DeepONet, and the architectures of DeepONet follows [18].

The mean L^2 relative errors on $N_{test} = 200$ test initial values of the various methods are presented in Fig. 7. As in the previous examples, ETDMD-DeepONet has the smallest error at all times, with a magnitude ranging from $1e-3$ to $1e-2$. A comparison of the errors of ETDMD-DeepONet with those of TDMD-DeepONet reveals that the auxiliary variable $\boldsymbol{c}$ contributes to improvements, particularly in the time range $[0, 0.2]$, which reflects the importance of feature enhancement. A less significant improvement is

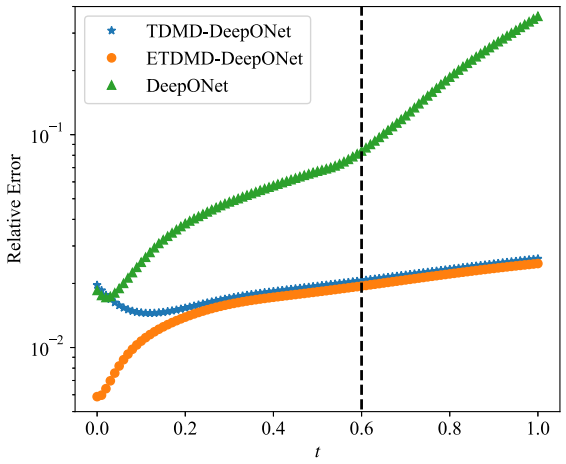

Fig. 7. The relative errors of different models in time region for Example 4.3.

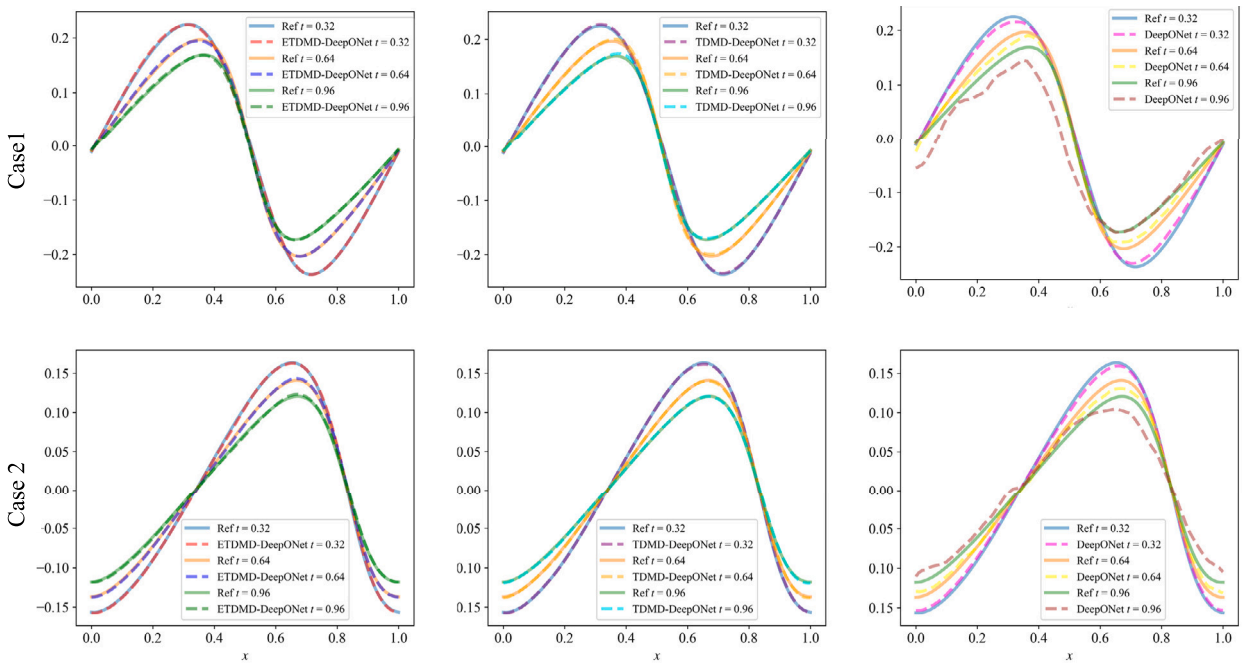

Fig. 8. Comparison between the reference solutions and the predicted solutions of (**Left**) ETDMD-DeepONet, (**Middle**) TDMD-DeepONet, and (**Right**) DeepONet for two initial conditions at the test time instants $t = 0.32, 0.64, 0.96$.

observed in the time range $[0.2, 1]$, which can be attributed to the fact that $\boldsymbol{c}$ is computed from the initial condition, and the solution approximation performance is better for the initial time. In contrast, the DeepONet estimation errors exhibit an increasing trend over time, with unsatisfactory results, particularly outside the training time range. Fig. 8 illustrates the solutions for two randomly selected samples in the test set at three different times, providing a more intuitive visual assessment. In comparison to the imprecise approximation of the standard DeepONet, the prediction of ETDMD-DeepONet and TDMD-DeepONet are in satisfactory alignment with the reference solution. Furthermore, the proposed methods are capable of accurately predicting the long-term operator evolution, both within and beyond the training time range. This further substantiates the reliability of the proposed framework.

4.4. Brusselator diffusion-reaction system

Next, let's consider a two-dimensional Brusselator diffusion-reaction system read:

$$\frac{\partial s}{\partial t} = D_0 \left(\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} \right) + a - (1 + b)s + vs^2,$$

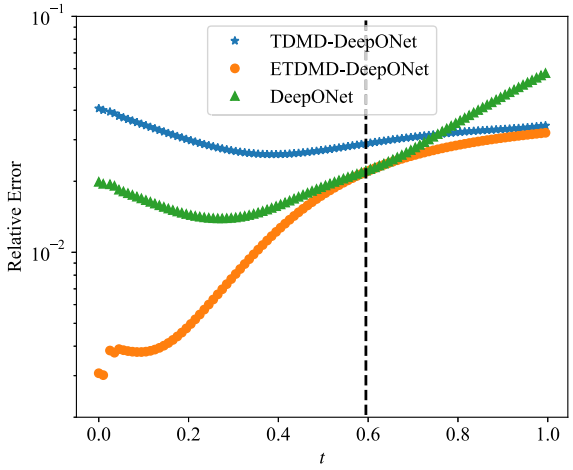

Fig. 9. The relative errors of different models for Example 4.4.

$$\frac{\partial v}{\partial t} = D_1\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + bs - vs^2, \quad \boldsymbol{x} \in [0,1]^2, t \in [0,1],$$

subject to the following initial conditions:

$$s(\boldsymbol{x}, t=0) = h_1(\boldsymbol{x}) \geq 0,$$
$$v(\boldsymbol{x}, t=0) = h_2(\boldsymbol{x}) \geq 0,$$

where $D_0 = 1$, $D_1 = 0.5$ are diffusion coefficients, $a = 1, b = 1.7$ represent constant concentrations. For this problem, the models are trained to learn the mapping between the initial field $h_2(\boldsymbol{x})$ and the solution $v(\boldsymbol{x}, t)$, i.e. $\mathcal{G}_\theta : h_2(\boldsymbol{x}) \to v(\boldsymbol{x}, t)$. The initial field $h_2(x, y)$ is modeled as a Gaussian random field with the squared exponential kernel.

To construct the training dataset, 878 input functions are sampled as initial conditions, with 744 allocated for the training set and the remainder for testing. In accordance with the methodology outlined in [57], the py-pde Python package is employed to generate reference solutions with time step 0.005, which can be fund in https://github.com/zwicker-group/py-pde. Each realization of the input stochastic field is recorded at 100 uniformly distributed time points over a 28×28 grid. The training set consisted of 744 realizations assessed on a $28 \times 28 \times 60$ spatial-temporal grid. The test set comprised 134 realizations, evaluated on a $28 \times 28 \times 100$ spatial-temporal grid, i.e. the number of time instances used for training, n_t, is equal to 60, and the entire time domain utilized for testing. In the context of DeepONet, the spatial-temporal coordinates corresponding to the 47040 points in the $28 \times 28 \times 60$ grid are utilized as inputs for the trunk network. The depth of the hidden layer corresponds to the trunk and branch nets, which are 3 and 2, respectively. Furthermore, the branch net and trunk net utilize the same network width, 196. In calculating $\boldsymbol{c}$, it is necessary to consider that the number of spatial-temporal grid points is greater than the truncation r_1. In order to avoid a significant discrepancy in accuracy between the calculated $\boldsymbol{c}$ in the training set and those outside the training set, resulting in a reduction in the model's generalizability, 101 spatial grid points are selected at random and the $\boldsymbol{c}$ calculated using the initial values of these points.

The mean L^2 relative errors on the test set are presented in Fig. 9. In the test dataset, the TDMD-DeepONet and the DeepONet test error are $\sim 1e-2$, and it is observed that within the training time range, the accuracy of TDMD-DeepONet is slightly inferior to that of DeepONet. This error mainly comes from the training part of the network. However, incorporating enhanced features can effectively help branch net training since ETDMD-DeepONet demonstrates superior performance in the range of training time. Furthermore, the auxiliary variable $\boldsymbol{c}$ enables the error to be reduced to $1e-3 \sim 1e-2$, particularly in the initial period. This results in an order of magnitude higher accuracy than the other models. To further illustrate the performance of different training models, we present the absolute errors between the predicted solutions and the reference solutions in Fig. 10. The results of the ETDMD-DeepONet demonstrate that the error between the predicted and true values is minimized, and that the evolution of each time step can be accurately simulated. Therefore, it can be concluded that the accuracy of ETDMD-DeepONet outperforms that of the TDMD-DeepONet and DeepONet. As previously observed, the reliability of DeepONet is affected by the lack of distributions beyond the training time domain within the training set. However, ETDMD-DeepONet and TDMD-DeepONet demonstrate better extrapolation performance than DeepONet, which further validates the ability of the proposed models to be implemented quickly for long-term prediction once they have been trained.

4.5. Navier-Stokes equation in the vorticity-velocity form

Finally, we apply the proposed methods to a more challenging example, which is the two-dimensional incompressible Navier-Stokes equation in the vorticity-velocity form:

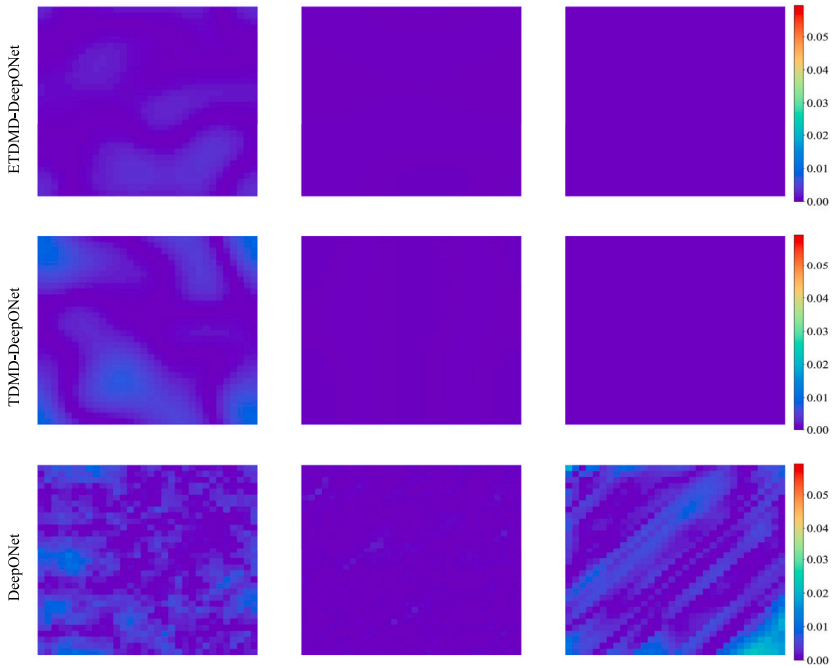

Fig. 10. The absolute errors between reference solutions and the corresponding predicted solutions at the test time instants $t = 0.3$ (left), 0.6 (middle), 0.91 (right).

$$\begin{aligned} \partial_t \omega + \boldsymbol{v} \cdot \nabla \omega &= \upsilon \Delta \omega + \boldsymbol{f}, \quad && (x, y) \in [0, 1]^2, t \in [0, T], \\ \nabla \cdot \boldsymbol{v} &= 0, && (x, y) \in [0, 1]^2, t \in [0, T], \\ \omega(x, 0) &= \omega_0(x), && (x, y) \in [0, 1]^2, \end{aligned}$$

where $\omega(x, y, t)$ and $\boldsymbol{u}(x, y, t)$ are the vorticity and velocity, respectively, and the viscosity υ is set to 0.006. The forcing term is defined as:

$$f(x, y) = 0.1 \sin(2\pi(x + y)) + 0.1 \cos(2\pi(x + y)).$$

In [3] and [4], the authors use DeepONet and Fourier operator network to learn the operator mapping the vorticity field at the time $t \in [0, 10]$ to the vorticity at a target time $T = 20$, respectively. In this case, what we learn is the vorticity in time $t \in [0, 0.45]$ to the vorticity in the target time range $[0.45, 20]$. We sample 1100 initial conditions, given by the Gaussian random field. The equation is solved following the approach in [4], using a time step of 10^{-4}. Snapshots with a spatial resolution of 32×32 are recorded at 501 uniformly spaced time steps for each initial condition, serving as the dataset for training and testing. In this example, the input to branch net is a tensor of shape $32 \times 32 \times 12$. Consequently, a CNN comprising three convolutional layers is employed, which is then connected to the FNN. The trunk net is an FNN with two hidden layers, each with 256 neurons.

Fig. 11 illustrates the mean L^2 relative errors of different methods. The accuracy of ETMD-DeepONet and TDMD-DeepONet surpasses that of the standard DeepONet. As expected, TDMD-DeepONet yields nearly optimal estimations, with the smallest errors observed between ETMD-DeepONet and the reference solutions in most cases. Notably, ETMD-DeepONet demonstrates the highest prediction accuracy within the training time range, suggesting that incorporating prior information $\boldsymbol{c}$ into the model helps network training. Here, we randomly select 156 space-time points within the time range $[0, 0.45]$ to calculate $\boldsymbol{c}$, it is evident that incorporating more snapshots with additional time steps into the calculation of $\boldsymbol{c}$ will improve the model's accuracy. Furthermore, using space cores and DMD-predicted time core in place of FNN avoids the necessity for network training, resulting in higher accuracy, particularly in regions outside the training time range.

Fig. 12 shows the absolute errors corresponding to ETMDM-DeepONet, TDMD-DeepONet, and DeepONet for the representative cases. With the availability of $\boldsymbol{c}$, the lowest error at $t = 2$ can be attained, outperforming the trained TDMD-DeepONet. The larger errors observed in TDMD-DeepONet are concentrated in the initial region of the domain. In this example, DeepONet exhibits relatively significant errors across the entire domain. Furthermore, the stability and long-term prediction capabilities of DeepONet are inferior to those of the proposed models.

The impact of the training time range on the accuracy of the proposed model is further investigated by selecting $T^* = 5$, $T^* = 10$, and $T^* = 15$, training different models accordingly, and presenting the corresponding errors in Fig. 13. Within the training time range, the performance of the three models shows no significant differences because the DMD have sufficient accuracy within the training set, and the errors primarily originate from the neural network component. Beyond the training time range, the accuracy of the models improves with increasing T^*. As analyzed in Section 3.3, a larger number of collected snapshots enhances the accuracy of DMD. Additionally, more training data contributes to the stability of the branch net.

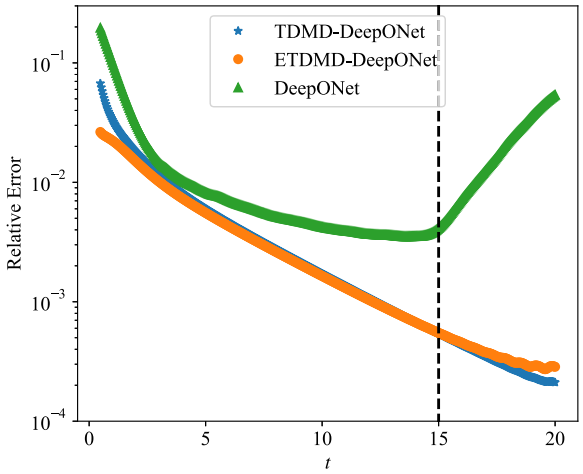

Fig. 11. The relative errors of different models for Example 4.5.

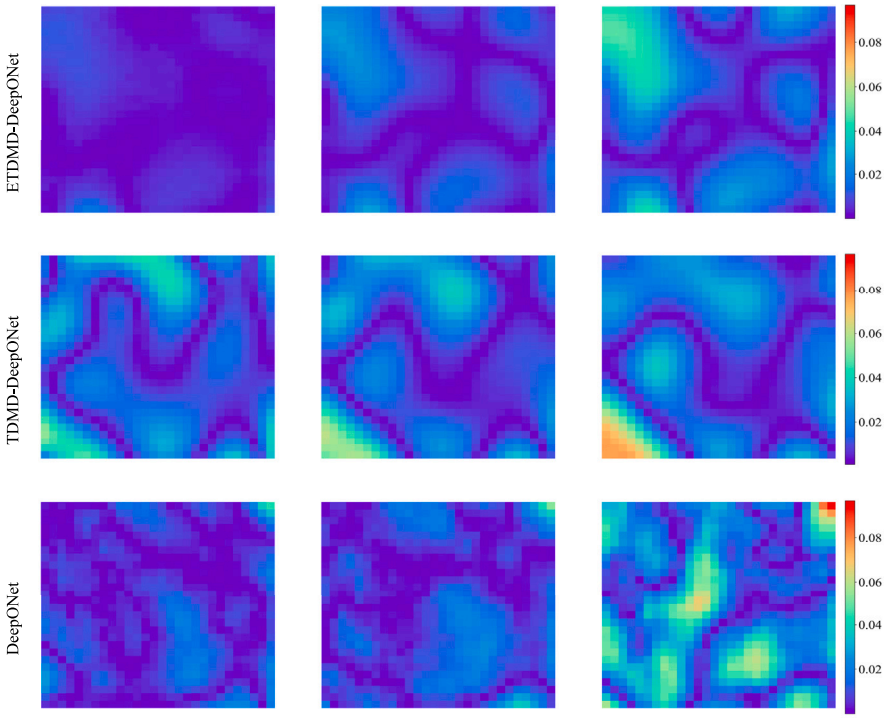

Fig. 12. The absolute errors between reference solutions and the corresponding predicted solutions at the test time instants $t = 2$ (left), 10 (middle), 18 (right).

5. Conclusion

This paper investigates the prediction of system states at different times using neural operator with various input functions, and proposes TDMD-DeepONet by fusing reduced-order model (ROM), which integrates TTD and DMD into operator learning network. We investigate the connection between DeepONet and TTD, revealing their structural consistency and showing that trunk net can be computed by the time-spatial cores obtained by TTD. To address the failure to predict solutions outside the training time domain of DeepONet, the DMD is introduced to model the dynamic evolution of the time core, a ROM that identifies the evolution of time-dependent systems and provides predictions for future states. DMD does not require iteration and generally outperforms other ROM-based methods (e.g., POD) in terms of extrapolation. Consequently, the trunk net is computed by combining the space cores and time core predicted by DMD. Branch net is a neural network with a widely varying form whose inputs are the parameters. The solution at any given time with varying parameters is computed by combining the outputs of trunk net and branch net. Furthermore, ETDMD-DeepONet is developed by incorporating the projection of the initial conditions as enhanced features into the branch net, which is accomplished by integrating a linear layer to the branch net for residual learning.

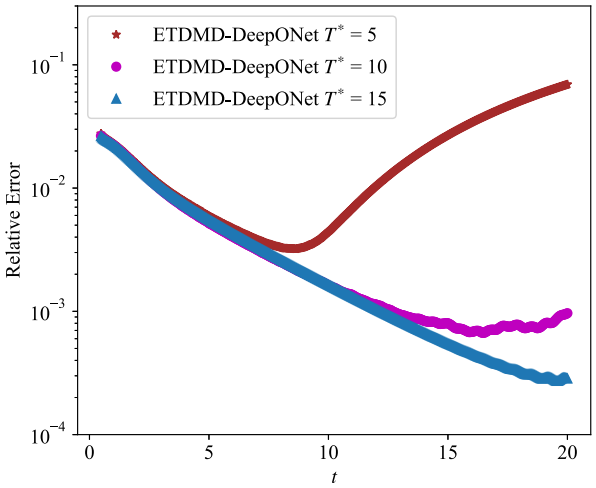

Fig. 13. The relative errors of different models for Example 4.5.

The proposed methods are applied to several examples and evaluated against the standard DeepONet. In each instance, the framework accurately predicts the evolution of complex systems with different parameters. Within the training time range, ETDMD-DeepONet has the highest accuracy, and outside the training time range, both proposed methods have comparable accuracy in most cases and are much better than the standard DeepONet, which demonstrates the excellent extrapolation ability of the proposed methods over longer time ranges. In the case where DMD can accurately model the evolution of the time core, the proposed methods demonstrate a relatively stable performance, with prediction errors remaining consistent over time. Notably, the framework proposed in this study is not limited to the standard DMD. Extensions of DMD can be applied to the proposed framework, particularly when combined with appropriate observation functions, which may enhance the accuracy of the DMD algorithm, thereby improving the accuracy of the proposed methods.

CRediT authorship contribution statement

Yuanhong Chen: Writing – original draft, Validation, Methodology, Conceptualization. **Yifan Lin:** Writing – review & editing, Investigation, Formal analysis. **Xiang Sun:** Writing – review & editing, Formal analysis. **Chunxin Yuan:** Writing – review & editing, Investigation. **Zhen Gao:** Writing – review & editing, Project administration, Methodology, Funding acquisition, Formal analysis.

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.

Acknowledgements

The authors would like to acknowledge the funding support of this research by the National Natural Science Foundation of China (1237143, 12201592), the Taishan Scholars Program (tsqn202211059) and the Shandong Provincial Natural Science Foundation (ZR2023MA043, ZR2022QA006).

Data availability

Data will be made available on request.

References

[1] S.L. Brunton, B.R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (2020) 477–508, https://doi.org/10.1146/annurev-fluid-010719-060214.

[2] T.J. Sejnowski, The unreasonable effectiveness of deep learning in artificial intelligence, Proc. Natl. Acad. Sci. 117 (48) (2020) 30033–30038, https://doi.org/10.1073/pnas.1907373117.

[3] 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 (2021) 218–229, https://doi.org/10.1038/s42256-021-00302-5.

[4] Z.-Y. Li, N.B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A.M. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv:2010.08895 [abs], 2020, https://doi.org/10.48550/arXiv.2010.08895
.

[5] P. Clark Di Leoni, L. Lu, C. Meneveau, G.E. Karniadakis, T.A. Zaki, Neural operator prediction of linear instability waves in high-speed boundary layers, J. Comput. Phys. 474 (2023) 111793, https://doi.org/10.1016/j.jcp.2022.111793.
[6] K. Kontolati, S. Goswami, M.D. Shields, G.E. Karniadakis, On the influence of over-parameterization in manifold based surrogates and deep neural operators, J. Comput. Phys. 479 (2023) 112008, https://doi.org/10.1016/j.jcp.2023.112008.
[7] Q. Cao, S. Goswami, T. Tripura, S. Chakraborty, G.E. Karniadakis, Deep neural operators can predict the real-time response of floating offshore structures under irregular waves, Comput. Struct. 291 (2024) 107228, https://doi.org/10.1016/j.compstruc.2023.107228.
[8] C. Lin, M. Maxey, Z. Li, G.E. Karniadakis, A seamless multiscale operator neural network for inferring bubble dynamics, J. Fluid Mech. 929 (2021) A18, https://doi.org/10.1017/jfm.2021.866.
[9] M. Yin, E. Zhang, Y. Yu, G.E. Karniadakis, Interfacing finite elements with deep neural operators for fast multiscale modeling of mechanics problems, Comput. Methods Appl. Mech. Eng. 402 (2022) 115027, https://doi.org/10.1016/j.cma.2022.115027.
[10] S. Cai, Z. Wang, L. Lu, T.A. Zaki, G.E. Karniadakis, Deepm & mnet: inferring the electroconvection multiphysics fields based on operator approximation by neural networks, J. Comput. Phys. 436 (2021) 110296, https://doi.org/10.1016/j.jcp.2021.110296.
[11] P. Jin, S. Meng, L. Lu, Mionet: learning multiple-input operators via tensor product, SIAM J. Sci. Comput. 44 (6) (2022) A3490–A3514, https://doi.org/10.1137/22M1477751.
[12] L. Tan, L. Chen, Enhanced deeponet for modeling partial differential operators considering multiple input functions, arXiv:2202.08942 [abs], 2022, https://doi.org/10.48550/arXiv.2202.08942.
[13] S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed deeponets, Sci. Adv. 7 (40) (2021) eabi8605, https://doi.org/10.1126/sciadv.abi8605.
[14] S. Goswami, M. Yin, Y. Yu, G.E. Karniadakis, A physics-informed variational deeponet for predicting crack path in quasi-brittle materials, Comput. Methods Appl. Mech. Eng. 391 (2022) 114587, https://doi.org/10.1016/j.cma.2022.114587.
[15] A.A. Howard, M. Perego, G.E. Karniadakis, P. Stinis, Multifidelity deep operator networks for data-driven and physics-informed problems, J. Comput. Phys. 493 (2023) 112462, https://doi.org/10.1016/j.jcp.2023.112462.
[16] X. Zhang, F. Xie, T. Ji, Z. Zhu, Y. Zheng, Multi-fidelity deep neural network surrogate model for aerodynamic shape optimization, Comput. Methods Appl. Mech. Eng. 373 (2021) 113485, https://doi.org/10.1016/j.cma.2020.113485.
[17] N. Demo, M. Tezzele, G. Rozza, A deeponet multi-fidelity approach for residual learning in reduced order modeling, Adv. Model. Simul. Eng. Sci. 10 (2023) 12, https://doi.org/10.1186/s40323-023-00249-9.
[18] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, G.E. Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data, Comput. Methods Appl. Mech. Eng. 393 (2022) 114778, https://doi.org/10.1016/j.cma.2022.114778.
[19] S. Venturi, T. Casey, SVD perspectives for augmenting deeponet flexibility and interpretability, Comput. Methods Appl. Mech. Eng. 403 (2023) 115718, https://doi.org/10.1016/j.cma.2022.115718.
[20] S. Pan, S.L. Brunton, J.N. Kutz, Neural implicit flow: a mesh-agnostic dimensionality reduction paradigm of spatio-temporal data, J. Mach. Learn. Res. 24 (41) (2023) 1–60, http://jmlr.org/papers/v24/22-0365.html.
[21] E. Barnard, L. Wessels, Extrapolation and interpolation in neural network classifiers, IEEE Control Syst. 12 (5) (1992) 50–53, https://doi.org/10.1109/37.158898.
[22] G. Lin, C. Moya, Z. Zhang, Learning the dynamical response of nonlinear non-autonomous dynamical systems with deep operator neural networks, Eng. Appl. Artif. Intell. 125 (2023) 106689, https://doi.org/10.1016/j.engappai.2023.106689.
[23] L. Liu, K. Nath, W. Cai, A causality-deeponet for causal responses of linear dynamical systems, Commun. Comput. Phys. 35 (2024) 1194–1228, https://doi.org/10.4208/cicp.OA-2023-0078.
[24] K. Michałowska, S. Goswami, G.E. Karniadakis, S. Riemer-Sørensen, Neural operator learning for long-time integration in dynamical systems with recurrent neural networks, in: 2024 International Joint Conference on Neural Networks (IJCNN), 2024, pp. 1–8, https://doi.org/10.1109/IJCNN60899.2024.10650331.
[25] J. He, S. Kushwaha, J. Park, S. Koric, D. Abueidda, I. Jasiuk, Sequential deep operator networks (S-DeepONet) for predicting full-field solutions under time-dependent loads, Eng. Appl. Artif. Intell. 127 (2024) 107258, https://doi.org/10.1016/j.engappai.2023.107258.
[26] Z. Kong, A. Mollaali, C. Moya, N. Lu, G. Lin, B-LSTM-MIONet: Bayesian lstm-based neural operators for learning the response of complex dynamical systems to length-variant multiple input functions, arXiv:2311.16519, 2023, https://doi.org/10.48550/arXiv.2311.16519.
[27] Z. Ge, Review on data-driven modeling and monitoring for plant-wide industrial processes, Chemom. Intell. Lab. Syst. 171 (2017) 16–25, https://doi.org/10.1016/j.chemolab.2017.09.021.
[28] Y. Sun, F. Haghighat, B.C. Fung, A review of the-state-of-the-art in data-driven approaches for building energy prediction, Energy Build. 221 (2020) 110022, https://doi.org/10.1016/j.enbuild.2020.110022.
[29] P.J. Schmid, Dynamic mode decomposition of numerical and experimental data, J. Fluid Mech. 656 (2010) 5–28, https://doi.org/10.1017/S0022112010001217.
[30] M.O. Williams, I.G. Kevrekidis, I.G. Kevrekidis, A data-driven approximation of the Koopman operator: extending dynamic mode decomposition, J. Nonlinear Sci. 25 (2015) 1307–1346, https://doi.org/10.1007/s00332-015-9258-5.
[31] Q. Li, F. Dietrich, E.M. Bollt, I.G. Kevrekidis, Extended dynamic mode decomposition with dictionary learning: a data-driven adaptive spectral decomposition of the Koopman operator, Chaos 27 (2017) 103111, https://doi.org/10.1063/1.4993854.
[32] J. Yin, Y. hao Chan, F.H. da Jornada, D.Y. Qiu, S.G. Louie, C. Yang, Using dynamic mode decomposition to predict the dynamics of a two-time non-equilibrium Green's function, J. Comput. Sci. 64 (2022) 101843, https://doi.org/10.1016/j.jocs.2022.101843.
[33] H. Lu, D.M. Tartakovsky, Lagrangian dynamic mode decomposition for construction of reduced-order models of advection-dominated phenomena, J. Comput. Phys. 407 (2020) 109229, https://doi.org/10.1016/j.jcp.2020.109229.
[34] D. Rafiq, M.A. Bazaz, Nonlinear model order reduction via nonlinear moment matching with dynamic mode decomposition, Int. J. Non-Linear Mech. 128 (2021) 103625, https://doi.org/10.1016/j.ijnonlinmec.2020.103625.
[35] J. Ding, S.-J. Cao, Identification of zonal pollutant diffusion characteristics using dynamic mode decomposition: towards the deployment of sensors, Build. Environ. 206 (2021) 108379, https://doi.org/10.1016/j.buildenv.2021.108379.
[36] W. Stankiewicz, Recursive dynamic mode decomposition for the flow around two square cylinders in tandem configuration, J. Fluids Struct. 110 (2022) 103515, https://doi.org/10.1016/j.jfluidstructs.2022.103515.
[37] I. Ul Haq, T. Iwata, Y. Kawahara, Dynamic mode decomposition via convolutional autoencoders for dynamics modeling in videos, Comput. Vis. Image Underst. 216 (2022) 103355, https://doi.org/10.1016/j.cviu.2021.103355.
[38] Y. Xiao, X. Zhang, X. Xu, X. Liu, J. Liu, Deep neural networks with Koopman operators for modeling and control of autonomous vehicles, IEEE Trans. Intell. Veh. 8 (1) (2023) 135–146, https://doi.org/10.1109/TIV.2022.3180337.
[39] Q.A. Huhn, M.E. Tano, J.C. Ragusa, Y. Choi, Parametric dynamic mode decomposition for reduced order modeling, J. Comput. Phys. 475 (2023) 111852, https://doi.org/10.1016/j.jcp.2022.111852.
[40] F. Andreuzzi, N. Demo, G. Rozza, A dynamic mode decomposition extension for the forecasting of parametric dynamical systems, SIAM J. Appl. Dyn. Syst. 22 (3) (2023) 2432–2458, https://doi.org/10.1137/22M1481658.
[41] Z. Gao, Y. Lin, X. Sun, X. Zeng, A reduced order method for nonlinear parameterized partial differential equations using dynamic mode decomposition coupled with k-nearest-neighbors regression, J. Comput. Phys. 452 (2022) 110907, https://doi.org/10.1016/j.jcp.2021.110907.
[42] Y. Lin, X. Sun, Y. Chen, Z. Gao, A dynamic mode decomposition based reduced-order model for parameterized time-dependent partial differential equations, J. Sci. Comput. 95 (2023) 70, https://doi.org/10.1007/s10915-023-02200-x.

[43] Y. Lin, X. Sun, J. Nie, Y. Chen, Z. Gao, An efficient reduced-order model based on dynamic mode decomposition for parameterized spatial high-dimensional pdes, Commun. Comput. Phys. 37 (2) (2025) 575–602, https://doi.org/10.4208/cicp.OA-2023-0135.
[44] I. Oseledets, Tensor-train decomposition, SIAM J. Sci. Comput. 33 (5) (2011) 2295–2317, https://doi.org/10.1137/090752286.
[45] J.H. Tu, C.W. Rowley, D.M. Luchtenburg, S.L. Brunton, J.N. Kutz, On dynamic mode decomposition: theory and applications, J. Comput. Dyn. 1 (2) (2014) 391–421, https://doi.org/10.3934/jcd.2014.1.391.
[46] O. Koch, C. Lubich, Dynamical low-rank approximation, SIAM J. Matrix Anal. Appl. 29 (2) (2007) 434–454, https://doi.org/10.1137/050639703.
[47] F.L. Hitchcock, The expression of a tensor or a polyadic as a sum of products, J. Math. Phys. 6 (1927) 164–189, https://doi.org/10.1002/sapm192761164.
[48] L.R. Tucker, Some mathematical notes on three-mode factor analysis, Psychometrika 31 (1966) 279–311, https://doi.org/10.1007/BF02289464.
[49] H. Zhang, C.W. Rowley, E.A. Deem, L.N. Cattafesta, Online dynamic mode decomposition for time-varying systems, SIAM J. Appl. Dyn. Syst. 18 (3) (2019) 1586–1609, https://doi.org/10.1137/18M1192329.
[50] M.O. Williams, C.W. Rowley, I.G. Kevrekidis, A kernel-based method for data-driven Koopman spectral analysis, J. Comput. Dyn. 2 (2) (2015) 247–265, https://doi.org/10.3934/jcd.2015005.
[51] D.J. Alford-Lago, C.W. Curtis, A.T. Ihler, O. Issan, Deep learning enhanced dynamic mode decomposition, Chaos, Interdiscip. J. Nonlinear Sci. 32 (3) (2022) 033116, https://doi.org/10.1063/5.0073893.
[52] Y. Jin, L. Hou, S. Zhong, Extended dynamic mode decomposition with invertible dictionary learning, Neural Netw. 173 (2024) 106177, https://doi.org/10.1016/j.neunet.2024.106177.
[53] P.J. Baddoo, B. Herrmann, B.J. McKeon, J. Nathan Kutz, S.L. Brunton, Physics-informed dynamic mode decomposition, Proc. Royal Soc., Math. Phys. Eng. Sci. 479 (2271) (2023) 20220576, https://doi.org/10.1098/rspa.2022.0576.
[54] Y. Yin, C. Kou, S. Jia, L. Lu, X. Yuan, Y. Luo, PF-DMD: physics-fusion dynamic mode decomposition for accurate and robust forecasting of dynamical systems with imperfect data and physics, arXiv:2311.15604, 2023, https://doi.org/10.48550/arXiv.2311.15604.
[55] W. Li, M.Z. Bazant, J. Zhu, Phase-field deeponet: physics-informed deep operator neural network for fast simulations of pattern formation governed by gradient flows of free-energy functionals, Comput. Methods Appl. Mech. Eng. 416 (2023) 116299, https://doi.org/10.1016/j.cma.2023.116299.
[56] M. Zhu, H. Zhang, A. Jiao, G.E. Karniadakis, L. Lu, Reliable extrapolation of deep neural operators informed by physics or sparse observations, Comput. Methods Appl. Mech. Eng. 412 (2023) 116064, https://doi.org/10.1016/j.cma.2023.116064.
[57] S. Goswami, K. Kontolati, M.D. Shields, G.E. Karniadakis, Deep transfer operator learning for partial differential equations under conditional shift, Nat. Mach. Intell. 4 (12) (2022) 1155–1164, https://doi.org/10.1038/s42256-022-00569-2.