Computer Methods in Applied Mechanics and Engineering 444 (2025) 118084

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

Convolutional neural network-based reduced-order modeling for parametric nonlocal PDEs

Yumeng Wang [a], Shiping Zhou [b], Yanzhi Zhang [a,*]

[a] Department of Mathematics and Statistics, Missouri University of Science and Technology, Rolla, 65409, MO, USA
[b] Department of Computational Mathematics, Science and Engineering, Michigan State University, East Lansing, 48824, MI, USA

ARTICLE INFO

Keywords:
Reduced order modeling
Parametric model reduction
Nonlocal problems
Convolutional autoencoder
Long-short term memory
Nonlocal inhomogeneous boundary conditions

ABSTRACT

In this paper, we propose a convolutional neural network (CNN) based reduced-order modeling (ROM) to solve parametric nonlocal partial differential equations (PDEs). Our method consists of two main components: dimensional reduction with convolutional autoencoder (CAE) and latent-space modeling with CNN or long short-term memory (LSTM) networks. Our neural network-based ROM bypasses the main challenges faced by intrusive approaches for nonlocal problems, such as non-affine parameter dependence and kernel singularities. To address nonlocal inhomogeneous boundary conditions, we introduce two effective strategies. Additionally, we present two approaches for incorporating parameters into the latent space and demonstrate that CNN mappings are particularly efficient for problems with high-dimensional parameter spaces. Our results provide the evidence that deep CAEs can successfully capture nonlocal behaviors, highlighting the promising potential of neural network-based ROMs for nonlocal PDEs. To the best of our knowledge, our method is the first neural network-based ROM methods developed for nonlocal problems. Extensive numerical experiments, including spatial and temporal nonlocal models, demonstrate that our neural network-based ROMs are effective in solving nonlocal problems. Moreover, our studies show that the compression capability of CAE outperforms traditional projection-based methods, especially when handling complex nonlinear problems.

1. Introduction

In recent decades, nonlocal models have emerged as a powerful tool for modeling complex phenomena, such as anomalous transport [1], fracture and crack propagation [2], and long-range quantum interactions [3], which cannot be accurately described by classical integer-order partial differential equations (PDEs). However, compared to classical PDEs, the nonlocality of nonlocal models also introduces significant challenges for their accurate and efficient numerical simulations; see [4–9] and references therein. The discretization of spatially nonlocal models results in systems with dense stiffness matrices, unlike the sparse matrices from classical PDEs. Consequently, solving these systems incurs significantly higher computational and storage costs. For temporal nonlocal models, the discrete system usually depends on the entire history of solutions from time $t = 0$, leading to significant storage costs, especially when computing the long-term dynamics of high-dimensional problems [10]. Moreover, additional numerical discretization challenges arise when the nonlocal operator involves singular kernels or complex interacting regions [6].

On the other hand, many applications involve parametric nonlocal PDEs that require repeatedly solving these equations for different parameter values [2,11,12]. For example, in peridynamic theory, the nonlocal kernel is often selected a posteriori, with

* Corresponding author.
E-mail address: zhangyanz@mst.edu (Y. Zhang).

https://doi.org/10.1016/j.cma.2025.118084
Received 6 November 2024; Received in revised form 17 April 2025; Accepted 9 May 2025
Available online 4 June 2025

parameters tuned to match experimental data. Identifying these parameters can be formulated as an optimization problem that minimizes the discrepancy between experimental data and the numerical solutions of nonlocal problems, which requires repeatedly solving the nonlocal problems with updated parameters [2]. However, as previously discussed, solving nonlocal problems even once can be prohibitively expensive due to high computational and storage costs, making it impractical to address parametric nonlocal PDEs at every point in the parameter space. Consequently, developing more efficient methods for solving parametric nonlocal models is a critical challenge for these applications.

In the literature, many neural network-based methods have been proposed to solve nonlocal problems. These works can be roughly divided into two groups: (i) Equation-driven approaches, also known as physics-informed neural networks (PINNs), include fPINN [13], nPINN [14], and their extensions [15]. This approach embeds the nonlocal PDEs, along with their boundary and initial conditions, into the loss function, ensuring that the neural network learns solutions that satisfy the nonlocal equations. However, compared to local problems, a key challenge of PINN for nonlocal problems is the difficulty of computing the integral operator in the loss function. As a result, significant efforts have focused on approximating the nonlocal operators, including numerical discretization [13,14], Monte Carlo approaches [15], and others [16]. Additionally, spatial nonlocal problems often involve nonlocal boundary conditions (such as volume constraints [6]), which can significantly increase the computational costs of training a PINN. (ii) Data-driven approaches. This approach learns a surrogate model from solution data, either directly [17,18] or indirectly via learning the features of nonlocal models [19]. While this approach avoids the complexity of approximating the nonlocal operators, existing methods still incur high computational costs when solving parametric PDEs. Other data-driven approaches for model discovery, such as learning kernel functions or nonlocal constitutive laws, are discussed in [16,20,21] and the references therein.

In this work, we propose a neural network-based reduced-order modeling approach for efficiently solving parametric nonlocal PDEs. Let $\Omega \subset \mathbb{R}^d$ be an open bounded domain. Consider the parametric nonlocal problem of the general form:

$$\mathcal{L}_t^{\boldsymbol{\mu}} u(\mathbf{x}, t; \boldsymbol{\mu}) = \mathcal{L}_{\mathbf{x}}^{\boldsymbol{\mu}} u(\mathbf{x}, t; \boldsymbol{\mu}) + \mathcal{N}(\mathbf{x}, t, u; \boldsymbol{\mu}), \quad \text{for } \mathbf{x} \in \Omega, \quad t \in (0, T], \tag{1.1}$$

$$u(\mathbf{x}, t; \boldsymbol{\mu}) = g(\mathbf{x}, t; \boldsymbol{\mu}), \quad \text{for } \mathbf{x} \in \Omega_\Gamma, \quad t \in [0, T], \tag{1.2}$$

where the solution $u(\mathbf{x}, t; \boldsymbol{\mu})$ depends on space $\mathbf{x}$, time t, and parameter $\boldsymbol{\mu} \in \mathbb{R}^p$. The linear differential operators $\mathcal{L}_t^{\boldsymbol{\mu}}$ and $\mathcal{L}_{\mathbf{x}}^{\boldsymbol{\mu}}$ are associated with time t and space $\mathbf{x}$, respectively, where the superscript $\boldsymbol{\mu}$ indicates that these operators may depend on the parameter $\boldsymbol{\mu}$. The operator $\mathcal{N}$ may include nonlinear terms of u. The general form in (1.1) covers both local and nonlocal PDEs. If $\mathcal{L}_{\mathbf{x}}^{\boldsymbol{\mu}}$ is a nonlocal operator, volume constraints are imposed on the region Ω_Γ adjacent to Ω, where Ω_Γ depends on the specific form of $\mathcal{L}_{\mathbf{x}}^{\boldsymbol{\mu}}$. In contrast, if $\mathcal{L}_{\mathbf{x}}^{\boldsymbol{\mu}}$ is a local operator, boundary conditions are given along $\partial\Omega$ (i.e. $\Omega_\Gamma = \partial\Omega$ in this case). The parameters $\boldsymbol{\mu}$ may appear in the differential operators, nonlinear terms, boundary conditions, or in initial conditions if a time-dependent problem is considered. We remark that a key challenge in developing ROMs for parametric nonlocal problems arises from the dependence of the nonlocal operator on parameters [12,22,23]; see more discussion in Section 2.

In this paper, as a proof-of-concept study, we propose a non-intrusive, data-driven neural-network based ROM approach for solving parametric nonlocal problems. To the best of our knowledge, neural network-based ROMs have not yet been explored in the context of nonlocal problems. Our method leverages a convolutional autoencoder (CAE) for dimensionality reduction and a convolutional neural network (CNN) to approximate the solution in the latent space. The main contributions of this work are summarized as follows:

(i) While CAEs are widely used for local problems, their application to nonlocal settings remains largely unexplored due to concerns that convolutional filters may fail to capture nonlocal interactions. We provide preliminary evidence that deep CAEs can effectively capture nonlocal behaviors. Our results suggest promising potential of neural network-based ROMs for nonlocal problems.

(ii) Existing ROMs for nonlocal problems are all intrusive and face significant challenges, including non-affine parameter dependence and kernel singularities. We propose a non-intrusive, data-driven ROM based on neural networks to bypass these challenges faced by intrusive approaches. In particular, we introduce two strategies to handle nonlocal inhomogeneous boundary conditions.

(iii) While existing ROMs for nonlocal problems focus mainly on linear cases, there is limited work on nonlocal nonlinear problems. We conduct ROM studies for various nonlinear problems across different applications, including both spatial and temporal nonlocal models. Moreover, we propose two approaches for incorporating parameters into the latent space, and demonstrate that CNN mappings in the latent space are efficient for problems with high-dimensional parameter space.

Unlike traditional ROMs, both the CAE-CNN and CAE-LSTM methods are equation-free, which greatly simplifies implementation and reduces computational costs during online prediction. Extensive numerical experiments are presented and demonstrate the effectiveness of our methods in solving parametric nonlocal problems. Moreover, our studies suggest that the compression capability of CAE outperforms principal component analysis (PCA), especially for handling complex nonlinear problems. Comparisons with traditional numerical methods further suggest that our ROM methods are highly efficient for solving parametric nonlocal PDEs, significantly reducing both computational and storage costs.

The paper is organized as follows. Section 2 reviews some existing methods and discusses the challenges and issues in developing ROMs for nonlocal problems. Section 3 introduces two strategies for handling nonlocal inhomogeneous boundary conditions, followed by a detailed presentation of our neural network-based ROMs. Section 4 presents benchmark problems to demonstrate the performance of our method, and Section 5 extends the study to nonlocal nonlinear problems in various applications. Finally, Section 6 concludes with a summary and discussion.

2. ROMs for nonlocal problems

Solving nonlocal problems requires prohibitively high computational and storage costs, along with complications in implementation. Hence, as pointed out in [6], "*reduced-order modeling is needed much more for nonlocal models than for corresponding local PDE models*". ROM methods can be broadly categorized into intrusive and non-intrusive approaches. Intrusive approaches require access to the original PDE to construct the reduced-order approximation [24], whereas non-intrusive methods treat the original system as a black box, offering greater flexibility in handling problems where the PDEs are unknown [25].

So far, the development of ROMs for nonlocal PDEs is relatively recent and remains very limited. To the best of our knowledge, existing ROMs for nonlocal problems are all intrusive methods, such as the projection-based techniques [12,22,26,27]. Moreover, these methods mainly address linear nonlocal models, with most studies focusing on the fractional Poisson problem

$$(-\Delta)^{\frac{\alpha}{2}} u(\mathbf{x}) = f(\mathbf{x}), \qquad \text{for } \alpha \in (0, 2), \tag{2.1}$$

and its variations, where $(-\Delta)^{\frac{\alpha}{2}}$ denotes the fractional Laplacian. We refer to recent works on ROM for the integral [12,27] and spectral [22,23,28] fractional Poisson problems. These studies have demonstrated the significant advantages of ROMs in solving nonlocal problems compared to traditional numerical methods [12,22,26,27]. However, parametric nonlocal problems introduce new and significant challenges compared to classical PDEs, making the development of ROMs more difficult, particularly for intrusive methods.

First, the parameters of nonlocal problems may arise from the nonlocal operators, resulting in a lack of affine dependence on these parameters. For instance, the fractional Poisson equation does not have affine dependence on parameter α, as it comes from the fractional Laplacian $(-\Delta)^{\frac{\alpha}{2}}$ [12,22]. On the other hand, affine dependence is crucial for the efficient implementation of intrusive ROMs through an offline-online decomposition. Consequently, this lack of affine dependence complicates the development of ROMs for parametric nonlocal problems, even in linear cases [12,22,26].

Second, many nonlocal operators are defined in terms of singular integrals, and the strength of the singularity may vary with the parameter. Hence, the parameter-dependent singularity even impacts the regularity properties of the solutions. For example, the integral fractional Laplacian has a kernel function $1/|\mathbf{x} - \mathbf{y}|^{d+\alpha}$, which becomes singular as $\mathbf{y} \to \mathbf{x}$, with the strength of the singularity increasing as the parameter α. Moreover, the kernel function may also exhibit discontinuities, with the locations of these discontinuities depending on the parameter (see the nonlocal model in [6,12]).

The combination of non-affinity and singularities presents significant challenges in developing ROMs for nonlocal problems, often making classical techniques fail. In the literature, a well-known approach to addressing non-affine parameter dependence is the empirical interpolation method (EIM) [29]. However, the discontinuity and singularity inherent in nonlocal operators hinder the direct application of EIM, as it is designed for continuous and bounded functions. Even though the continuity requirement can be relaxed through a generalized EIM, selecting an appropriate interpolating function for these nonlocal operators remains challenging [12]. Consequently, numerous studies have been reported to address these issues. In [12], to achieve an affine approximation, interpolation with local polynomials is proposed, and moreover a "cutoff" strategy is used to treat the singularity of the kernel. In [22], a Dirichlet-to-Neumann mapping is used to reformulate the fractional Poisson equation into a local problem with an additional spatial dimension. Consequently, the dependence of the nonlocal operator on the parameter is transferred to the coefficient function of the local operator. Since this coefficient function exhibits singularities for $\alpha \in [1, 2)$, a modified EIM must be designed and applied to achieve an affine approximation. Similar approaches have been used in [23,28], where nonlocal problems are reformulated as local problems using Kato's integral formula or rational approximations. However, it is important to note that transforming nonlocal problems into local ones does not automatically resolve the issue of non-affine parameter dependence, and techniques such as EIM remain essential for achieving an affine approximation.

In summary, the development of ROMs for nonlocal problems faces significant challenges compared to classical PDEs. Moreover, existing ROM methods require numerical discretization of nonlocal models in the reduced basis space, which not only increases computational costs but also complicates implementation [12,26,27,30]. These challenges hinder the development and application of ROMs for nonlocal problems.

3. Neural network-based ROMs

In this section, we introduce non-intrusive, data-driven neural network-based ROM methods for solving parametric nonlocal problems. Our methods are the first neural network-based ROMs for nonlocal problems, offering a novel alternative to bypass some of the main challenges faced by intrusive ROM approaches in this area. In Section 3.1, we show how the challenges of nonlocal problems are treated in our method and propose two approaches to addressing inhomogeneous nonlocal boundary constraints. In Section 3.2, we introduce the CAE for dimension reduction. In Section 3.3, two different models are introduced to approximate the solution in the latent space. In Section 3.4, we summarize the algorithms for offline training and online prediction of our methods.

3.1. Challenges in nonlocal problems

As discussed previously, the combination of non-affinity and singularities presents significant challenges in developing intrusive ROMs for nonlocal problems, often making classical techniques (such as EIM) fail. This motivates the development of data-driven, neural network-based ROMs to avoid directly handling these challenges during the ROM construction process. Autoencoders provide a fundamentally different framework from traditional projection-based methods. As a result, the lack of affine dependence is no

longer a limitation. Moreover, as a purely data-driven approach, our method eliminates the need to handle singularities during the development of the ROM. Instead, singularity-related difficulties are confined to the generation of high-accuracy training data, where well-established numerical methods can be effectively applied [5,7,31].

In addition to non-affinity and singularity, nonlocal boundary conditions further complicate the development of ROMs for nonlocal problems. Unlike classical PDEs, where boundary conditions are imposed along the domain's boundary, nonlocal problems often have boundary conditions specified over a region adjacent to the domain. Special care must be taken to ensure that the reduced-order system satisfies the extended boundary conditions, particularly in the case of inhomogeneous nonlocal boundary conditions. Currently, most existing ROMs for nonlocal problems focus on homogeneous Dirichlet boundary conditions [12,22,23,26]. Nonlocal inhomogeneous boundary conditions pose significant challenges and remain largely underexplored in the literature.

In this work, we introduce two strategies for addressing nonlocal inhomogeneous boundary conditions: hard and soft boundary constraints. The *hard boundary constraint* approach exactly imposes nonlocal inhomogeneous boundary conditions by constructing a function:

$$w(\mathbf{x},t)=\begin{cases}\sum_{i=1}^{M}\lambda_i(t)\phi(|\mathbf{x}-\mathbf{x}_i|), & \text{for } \mathbf{x}\in\Omega,\\ g(\mathbf{x},t), & \text{for } \mathbf{x}\in\Omega_\Gamma,\end{cases} \tag{3.1}$$

where $\phi(|\mathbf{x}-\mathbf{x}_i|)$ represents a radial basis function (RBF) centered at $\mathbf{x}_i$, and M is the number of RBF interpolation points. Here, the coefficients λ_i can be determined by requiring $w(\mathbf{x},t)=g(\mathbf{x},t)$ for $\mathbf{x}\in\Omega_\Gamma$ and $t\in[0,T]$ [7]. Then we can rewrite the solution $u(\mathbf{x},t)=v(\mathbf{x},t)+w(\mathbf{x},t)$, where $v(\mathbf{x},t)$ satisfies homogeneous boundary conditions. Taking the fractional Poisson equation as an example, Eq. (4.1) can be reformulated as

$$(-\Delta)^{\frac{\alpha}{2}}v(\mathbf{x})=f(\mathbf{x})-(-\Delta)^{\frac{\alpha}{2}}w(\mathbf{x}) \tag{3.2}$$

with homogeneous boundary conditions for $v(\mathbf{x})$. In this formulation, the complexity introduced by the inhomogeneous boundary conditions is shifted to constructing the function $w(\mathbf{x})$ and computing $(-\Delta)^{\frac{\alpha}{2}}w(\mathbf{x})$. This allows the development of ROMs to focus on homogeneous boundary conditions, which is crucial for CNN-based methods where zero padding is often applied.

Compared to the hard boundary constraint approach, the *soft boundary constraint* method is relatively straightforward. In this approach, training data are sampled from an extended domain $\widehat{\Omega}$ such that $\bar{\Omega}\subseteq\widehat{\Omega}$. Specifically, data within Ω are generated by solving the nonlocal PDE, while data in $\widehat{\Omega}\backslash\Omega$ are derived directly from the prescribed boundary conditions. As a result, the neural network is trained on a larger domain than Ω, allowing it to implicitly learn and account for the nonlocal inhomogeneous boundary conditions.

In the following, we provide the details of our neural network-based ROMs. The training data are sampled on the physical domain Ω for problems with homogeneous boundary conditions or with inhomogeneous conditions handled using the hard boundary constraint approach. For inhomogeneous conditions treated using the soft boundary constraint approach, the data are sampled on the extended domain $\widehat{\Omega}$.

3.2. Dimension reduction with CAE

First, we introduce the CAE for dimension reduction. Autoencoders have proven effective for complex nonlinear problems. They consist of two components: an encoder that compresses high-dimensional data into low-dimensional representations, and a decoder that reconstructs the data from the latent space. In this work, we use convolutional neural networks for both the encoder and the decoder.

Let N denote the dimension of the original high-dimensional data, while n denotes the latent dimension, and $n\ll N$. Denote the encoder as $\Psi:\mathbb{R}^N\to\mathbb{R}^n$, mapping high-dimensional data $\mathbf{u}\in\mathbb{R}^N$ to a low-dimensional representation $\mathbf{v}\in\mathbb{R}^n$:

$$\mathbf{v}=\Psi(\mathbf{u};\boldsymbol{\vartheta}_{\mathrm{e}}), \tag{3.3}$$

where $\boldsymbol{\vartheta}_{\mathrm{e}}$ denotes the set of neural network parameters in the encoder. The low-dimensional representation $\mathbf{v}\in\mathbb{R}^n$ is also referred to as the "latent space representation" of $\mathbf{u}\in\mathbb{R}^N$. The decoder, denoted as $\boldsymbol{\Phi}:\mathbb{R}^n\to\mathbb{R}^N$, reconstructs the high-dimensional data $\hat{\mathbf{u}}\in\mathbb{R}^N$ from the low-dimensional representation $\mathbf{v}\in\mathbb{R}^n$, i.e.,

$$\hat{\mathbf{u}}=\boldsymbol{\Phi}(\mathbf{v};\boldsymbol{\vartheta}_{\mathrm{d}}), \tag{3.4}$$

where $\boldsymbol{\vartheta}_{\mathrm{d}}$ represents the set of parameters in the decoder. Here, we use $\hat{\mathbf{u}}$ to represent the reconstructed data from the decoder, distinguishing it from the original data $\mathbf{u}$. The encoder mainly includes convolutional, pooling, and fully-connected layers, while the decoder consists of convolutional, upsampling, and fully-connected layers; see the illustration in Fig. 1.

The training data are taken as the high-fidelity solutions of (1.1)–(1.2) with representative parameters and/or time steps, i.e. $\left\{\mathbf{u}(\boldsymbol{\mu}_i,t_m)\in\mathbb{R}^N \mid 1\le i\le N_{\mathrm{tr}}^{\boldsymbol{\mu}},\ 1\le m\le N_{\mathrm{tr}}^{t}\right\}$. Suppose numerical discretization of (1.1) results in N_t time steps and $N_{x^{(k)}}$ spatial points in the kth direction for $1\le k\le d$. Here, N is the total number of spatial points, i.e., $N=\prod_{k=1}^{d}N_{x^{(k)}}$. The number of time snapshots $N_{\mathrm{tr}}^{t}\le N_t$, and $N_{\mathrm{tr}}^{\boldsymbol{\mu}}$ is the number of distinct parameters $\boldsymbol{\mu}$ considered in the training data. These high-fidelity training data serve as the input of our encoder, while the outputs are their latent space representations $\left\{\mathbf{v}(\boldsymbol{\mu}_i,t_m)\in\mathbb{R}^n \mid 1\le i\le N_{\mathrm{tr}}^{\boldsymbol{\mu}},\ 1\le m\le N_{\mathrm{tr}}^{t}\right\}$. Conversely, starting from the low-dimensional representation, the decoder reconstructs and outputs the high-dimensional data, denoted as $\left\{\hat{\mathbf{u}}(\boldsymbol{\mu}_i,t_m)\in\mathbb{R}^N \mid 1\le i\le N_{\mathrm{tr}}^{\boldsymbol{\mu}},\ 1\le m\le N_{\mathrm{tr}}^{t}\right\}$.

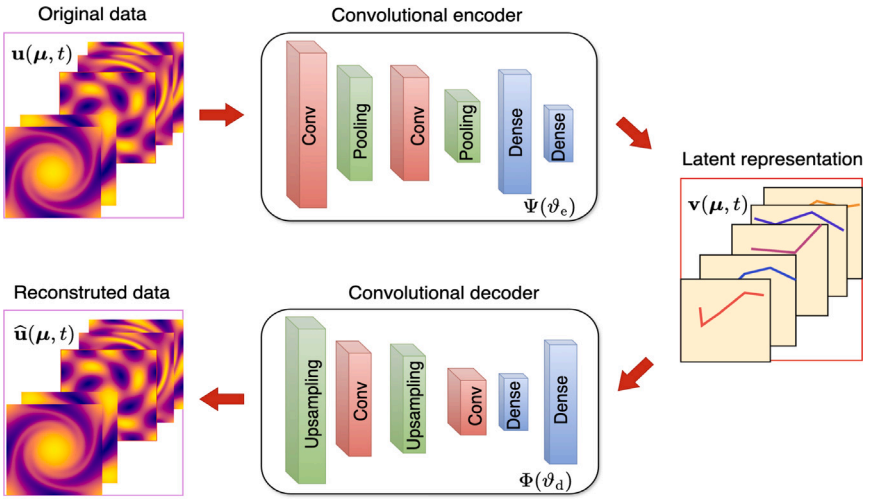

Fig. 1. Illustration of the convolutional autoencoder for dimension reduction.

In our method, we use a decoupled training strategy – training the CAE and the latent-space model separately. More discussion of other training strategies can be found in Section 3.4. Define the loss function of CAE as:

$$\begin{aligned}\mathcal{L}_{\text{CAE}}(\vartheta_e, \vartheta_d) &= \frac{1}{N_{\text{tr}}^t N_{\text{tr}}^{\mu}} \sum_{i=1}^{N_{\text{tr}}^{\mu}} \sum_{m=1}^{N_{\text{tr}}^t} \|\mathbf{u}(\boldsymbol{\mu}_i, t_m) - \hat{\mathbf{u}}(\boldsymbol{\mu}_i, t_m)\|_{\text{mse}} \\ &= \frac{1}{N_{\text{tr}}^t N_{\text{tr}}^{\mu}} \sum_{i=1}^{N_{\text{tr}}^{\mu}} \sum_{m=1}^{N_{\text{tr}}^t} \left\|\mathbf{u}(\boldsymbol{\mu}_i, t_m) - \boldsymbol{\Phi}\big(\boldsymbol{\Psi}\big(\mathbf{u}(\boldsymbol{\mu}_i, t_m); \vartheta_e\big); \vartheta_d\big)\right\|_{\text{mse}},\end{aligned} \tag{3.5}$$

where $\|\cdot\|_{\text{mse}}$ represents the mean squared error (MSE). The encoder and decoder are trained together by minimizing the difference between the original data $\mathbf{u}$ and its reconstruction $\hat{\mathbf{u}}$, i.e. $\min_{\vartheta_e, \vartheta_d} \mathcal{L}_{\text{CAE}}(\vartheta_e, \vartheta_d)$. During training, the parameters ϑ_e and ϑ_d are updated and learned simultaneously.

3.3. Solution approximation in latent space

As discussed, the encoder compresses the high-dimensional solution data $\mathbf{u}(\boldsymbol{\mu}_i, t_m) \in \mathbb{R}^N$ into the low-dimensional latent representation $\mathbf{v}(\boldsymbol{\mu}_i, t_m) \in \mathbb{R}^n$. In this section, we focus on approximating the latent-space solution $\mathbf{v}(\boldsymbol{\mu}, t) \in \mathbb{R}^n$ in terms of parameters $\boldsymbol{\mu}$ and time t. We introduce two different approaches. In Section 3.3.1, we introduce a CNN to learn the mapping from parameters and time to the encoded solution. For time-dependent problems, we also propose an alternative approach using an LSTM network to learn sequential relationships over discrete time steps, detailed in Section 3.3.2.

3.3.1. Latent space mapping

Here, we propose a CNN to learn the mapping from parameters $\boldsymbol{\mu}$ and time t to the latent-space solution, i.e., $(\boldsymbol{\mu}, t) \mapsto \mathbf{v}(\boldsymbol{\mu}, t)$. Denote the CNN as $\Theta : \mathbb{R}^p \times \mathbb{R} \to \mathbb{R}^n$, i.e.,

$$\hat{\mathbf{v}} = \Theta(\boldsymbol{\mu}, t; \vartheta_m), \tag{3.6}$$

where ϑ_m denotes the set of CNN parameters, and $\hat{\mathbf{v}} \in \mathbb{R}^n$ represents the learned latent space solution for parameters $\boldsymbol{\mu} \in \mathbb{R}^p$ and time $t \in \mathbb{R}$. Fig. 2 illustrates the proposed CNN for mapping model parameters $\boldsymbol{\mu}$ and time t to the latent space representation. It includes convolutional layers and fully-connected layers. The Rectified Linear Unit (ReLU) activation function is applied to all layers, except the last fully-connected layer. Here, we treat the parameters $\boldsymbol{\mu}$ and time t as separate channels to maintain their individual representations. Consequently, the input layer has $p+1$ channels with respect to the parameters and time, and the filters in the first convolution layer have a depth of $p+1$. In contrast to FNN, CNN treats parameters by different channels. The advantage of CNNs becomes more significant as the number of parameters increases. More comparisons and discussion can be found in Section 4.

As mentioned, we use a decoupled training strategy, training the latent-space model separately from the CAE. Here, the latent-space mapping model is obtained by $\min_{\vartheta_m} \mathcal{L}_{\text{Map}}(\vartheta_m)$, where the loss function is defined as

$$\mathcal{L}_{\text{Map}}(\vartheta_m) = \frac{1}{N_{\text{tr}}^t N_{\text{tr}}^{\mu}} \sum_{i=1}^{N_{\text{tr}}^{\mu}} \sum_{m=1}^{N_{\text{tr}}^t} \|\mathbf{v}(\boldsymbol{\mu}_i, t_m) - \Theta(\boldsymbol{\mu}_i, t_m; \vartheta_m)\|_{\text{mse}}.$$

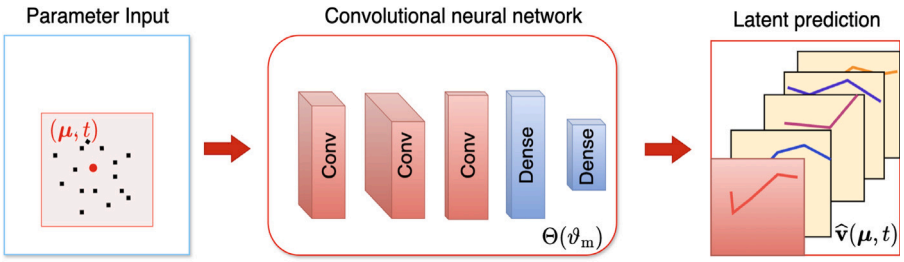

Fig. 2. Illustration of the CNN for latent-space mapping.

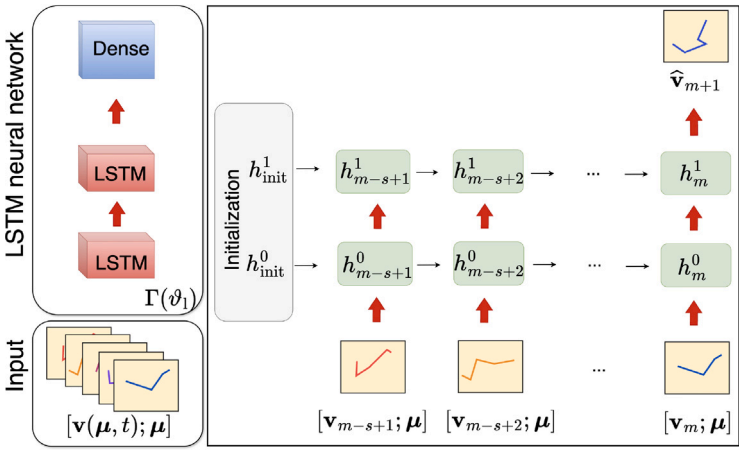

Fig. 3. Illustration of the LSTM network for time-marching in latent space.

Our CNN learns a continuous mapping from parameters μ and time t to the latent-space solution $\mathbf{v}(\mu, t)$. It works for both time-dependent and time-independent problems. For time-independent problems, the loss function $\mathcal{L}_{\text{Map}}(\vartheta_m)$ involves summation only over parameters μ_i. For time-dependent problems, our CNN can directly predict the solution in the latent space for any given time $t \in (0, T]$.

Our latent-space mapping approach does not require uniform time sampling in the training data. Moreover, the number of time snapshots can differ across different solution trajectories. This feature is particularly beneficial for solving temporal multiscale problems. For example, it allows us to capture rapidly changing solutions by increasing the number of time snapshots, similar to the idea of using adaptive time steps in traditional numerical methods. Hence, when solving temporal multiscale problems, we can avoid uniform sampling with small time steps across the entire time range, thus reducing computational costs.

3.3.2. *Time marching in latent space*

For time-dependent problems, we propose an alternative approach – an LSTM-based time-marching method – to learn the latent-space solution at time t_{m+1} (i.e. $\widehat{\mathbf{v}}(\mu, t_{m+1})$) from the parameters μ and solution of previous $s \in \mathbb{N}$ time steps.

Suppose that the training data are sampled at uniform time steps. Fig. 3 illustrates our LSTM network. It includes both LSTM layers and fully-connected layers. Let h^j_k denote the jth hidden state at time t_k. For $j \geq 1$ and $(m-s)+1 \leq k \leq m$, the hidden state h^j_k is computed from h^{j-1}_k and h^j_{k-1}. Specifically, for $j = 0$, the 0th hidden state h^0_k is computed from the solution data $\mathbf{v}(\mu, t_k)$ and hidden state h^0_{k-1}. Note that for $k = (m-s)+1$, the hidden state h^j_{k-1} is given by a randomly initialized state. Finally, the last hidden state at time t_m is passed to the fully-connected layers to predict the solution $\widehat{\mathbf{v}}(\mu, t_{m+1})$. The ReLU activation function is applied to all LSTM and fully-connected layers, except for the last fully-connected layer, which has no activation function. It is worth mentioning that LSTM networks are typically designed for handling data sampled at uniform time steps. To handle non-uniform time data, additional modifications to the LSTM network are required [32].

In contrast to the mapping approach, an LSTM learns the temporal dynamics of the latent-space solutions, with time t being handled implicitly. One challenge in solving parametric problems is how to incorporate parameters into the LSTM network. In [33], an additional neural network is introduced to interpret the parameters, and its output is subsequently combined with the LSTM's output to generate the final prediction. In our approach, we directly combine parameters with the latent-space solutions to form augmented data $[\mathbf{v}(\mu, t); \mu]$, which serves as the input to the LSTM network. Denote the LSTM as $\Gamma : \mathbb{R}^{n+p} \to \mathbb{R}^n$, i.e.,

$$\widehat{\mathbf{v}}(\mu, t_{m+1}) = \Gamma\left(\left\{[\mathbf{v}(\mu, t_{m-k+1}); \mu]\right\}_{k=1}^{s}; \vartheta_l\right), \tag{3.7}$$

for $s \geq 1$, where ϑ_l denotes the set of LSTM parameters. The LSTM is trained by minimizing the difference between the predicted latent-space solution $\hat{\mathbf{v}}(\boldsymbol{\mu}, t)$ and the ground truth solution $\mathbf{v}(\boldsymbol{\mu}, t)$ encoded by CAE, i.e., $\min_{\vartheta_l} \mathcal{L}_{\text{LSTM}}(\vartheta_l)$, where the loss function

$$\mathcal{L}_{\text{LSTM}}(\vartheta_l) = \frac{1}{(N_{\text{tr}}^t - s)N_{\text{tr}}^{\mu}} \sum_{i=1}^{N_{\text{tr}}^{\mu}} \sum_{m=s}^{N_{\text{tr}}^t - 1} \left\| \mathbf{v}(\boldsymbol{\mu}_i, t_{m+1}) - \Gamma\left(\left\{ [\mathbf{v}(\boldsymbol{\mu}_i, t_{m-k+1}); \boldsymbol{\mu}_i] \right\}_{k=1}^{s}; \vartheta_l \right) \right\|_{\text{mse}},$$

for $s \geq 1$. Generally, using more steps (i.e. a larger s) in the LSTM may improve accuracy, but also introduces challenges if choosing $s > 1$. For time-dependent problems, only the initial solution at $t = 0$ is typically provided, meaning data is available at a single time step. Therefore, when $s > 1$, the challenge lies in generating the solutions for these extra $(s-1)$ steps in order to train or make predictions with the LSTM [34,35]. Moreover, a larger s increases the computational costs in training the model. Therefore, in our study, we set $s = 1$ for its simplicity and efficiency.

The LSTM and CNN represent two different types of latent-space modeling approaches. LSTM models temporal dependencies in sequential data by learning how encoded solutions evolve over consecutive time steps, while CNN learns a mapping between the encoded solutions and the input $(\boldsymbol{\mu}, t)$. LSTM usually requires the training data to be sampled at uniform time steps. It begins with the encoded initial conditions and iteratively predicts the solution dynamics until the desired time. This iterative process can be time-consuming and may lead to error accumulation in long-term predictions. Moreover, in the online stage, the encoder is required for LSTM to encode the solutions from the initial s time steps. In contrast, the CNN mapping allows for predicting solutions at any time within the studied time frame, including time points not present in the training data. It does not require the training data to be sampled at uniform time steps. Furthermore, the number of time snapshots can vary between different training data. More comparison can be found in Section 5.2.

3.4. Model training and prediction

Our neural network-based ROMs consist of two parts: dimension reduction with CAE and latent-space solution approximation. It can be summarized in two stages: the offline stage and the online stage. In the offline stage, we train the model using high-fidelity solution data. In the online stage, the trained model is used to predict solutions for new parameters.

In the **offline stage**, both the CAE (i.e. $\Psi(\vartheta_{\text{e}})$ and $\Phi(\vartheta_{\text{d}})$) and latent-space neural network (i.e. $\Theta(\vartheta_{\text{m}})$ or $\Gamma(\vartheta_l)$) are trained. Generally, two different strategies can be used to train these two components: decoupled training and joint training. The decoupled approach trains the CAE and the latent-space neural network sequentially and separately. In contrast, joint approach trains them simultaneously. In our study, we use the decoupled training strategy [36]. More discussion of joint training approach can be found in the following remark:

Remark 3.1 (*Joint Training Approach*)**.** In the joint training approach, the CAE and the latent-space neural network are trained simultaneously with a weighted loss function [37,38], i.e.,

$$\min_{\vartheta_{\text{e}}, \vartheta_{\text{d}}, \vartheta_{\text{latent}}} \left(\omega_1 \mathcal{L}_{\text{CAE}}(\vartheta_{\text{e}}, \vartheta_{\text{d}}) + \omega_2 \mathcal{L}_{\text{latent}}\left(\{\mathbf{v}(\boldsymbol{\mu}_i, t_m; \vartheta_{\text{e}})\}; \vartheta_{\text{latent}} \right) \right), \tag{3.8}$$

where $\omega_1, \omega_2 > 0$ are the weights of two loss functions. For notational simplicity, we use $\mathcal{L}_{\text{latent}}$ and $\vartheta_{\text{latent}}$ to represent the loss function of the latent-space model and its corresponding parameters. The dependence on the latent-space representation $\mathbf{v}$ in the loss function $\mathcal{L}_{\text{latent}}$ is explicitly stated for clarity and easy comparison. In both the decoupled and joint training approaches, the loss function $\mathcal{L}_{\text{latent}}$ implicitly depends on the encoder parameters through the low-dimensional data $\mathbf{v}$. Specifically, in the decoupled training, the loss function depends on the optimal parameters ϑ_{e}^*, whereas in the joint approach, it depends on the parameters ϑ_{e}. Hence, the encoder, decoder, and latent-space models are trained simultaneously in the joint approach.

The joint training in (3.8) can be viewed as training the CAE with a regularization term from the latent-space neural network. Balancing the weights ω_1 and ω_2 of these two loss functions can be challenging in practice [37,39]. In the literature (e.g. [37,39,40]), various weights have been tested, demonstrating that the choice of weights affects model performance. However, in most cases, $\omega_1 = \omega_2 = 0.5$ is used [37]. Additionally, joint training significantly increases the number of parameters, which can negatively impact the training process.

The detailed offline training algorithm of our method is summarized in Algorithm 1. For brevity, we omit the index ranges for the solution data $\mathbf{u}$ and $\mathbf{v}$. The algorithm includes two separate steps. In the first step, the CAE is trained using the high-fidelity solution data $\{\mathbf{u}(\boldsymbol{\mu}_i, t_m)\}$. At the end of this step, the loss function $\mathcal{L}_{\text{CAE}}$ is minimized, resulting in the trained encoder $\Psi(\vartheta_{\text{e}}^*)$ and decoder $\Phi(\vartheta_{\text{d}}^*)$, where ϑ_{e}^* and ϑ_{d}^* denote the optimal parameters of the encoder and decoder. In the second step, the latent-space neural network (CNN or LSTM) is trained. Here, the training solution data $\{\mathbf{v}(\boldsymbol{\mu}_i, t_m)\}$ are encoded from $\{\mathbf{u}(\boldsymbol{\mu}_i, t_m)\}$ using the trained encoder $\Psi(\vartheta_{\text{e}}^*)$ from the first step. In other words, the latent-space training data depend on the parameters ϑ_{e}^*.

In the **online stage**, the choice of latent space models from Sections 3.3.1 or 3.3.2 leads to different models. If the latent-space mapping is used, the prediction process is illustrated in Fig. 4, and the algorithm is summarized in Algorithm 2. When given with new parameters $(\boldsymbol{\mu}_{\text{new}})$ and/or time (t_{new}), the CNN predicts the corresponding latent-space solution, which is then decoded into the predicted high-dimensional solution. The process can be described as

$$(\boldsymbol{\mu}_{\text{new}}, t_{\text{new}}) \xrightarrow{\Theta_m(\vartheta_{\text{m}}^*): \mathbb{R}^p \times \mathbb{R} \to \mathbb{R}^n} \hat{\mathbf{v}}(\boldsymbol{\mu}_{\text{new}}, t_{\text{new}}) \xrightarrow{\Phi(\vartheta_{\text{d}}^*): \mathbb{R}^n \to \mathbb{R}^N} \hat{\mathbf{u}}(\boldsymbol{\mu}_{\text{new}}, t_{\text{new}}), \tag{3.9}$$

where t_{new} can be any time within the studied time frame $[0, T]$.

Algorithm 1 Offline stage

Step 1: Train the convolutional autoencoder (CAE)

Input: High-fidelity solution data $\{\mathbf{u}(\boldsymbol{\mu}_i, t_m)\}$ and hyperparameters
Output: Optimal autoencoder parameters $\boldsymbol{\vartheta}_{\mathrm{e}}^*$ and $\boldsymbol{\vartheta}_{\mathrm{d}}^*$

1: Initialize $\boldsymbol{\vartheta}_{\mathrm{e}}$ and $\boldsymbol{\vartheta}_{\mathrm{d}}$ randomly
2: Set $N_{\mathrm{iter}} = 0$
3: **while** (**!** early-stopping **and** $N_{\mathrm{iter}} \leq N_{\mathrm{epo}}$) **do**
4: **for** $k = 1 : N_{\mathrm{batch}}$ **do**
5: $q = N_{\mathrm{batch}} N_{\mathrm{iter}} + k$
6: Sample batch data $\{\mathbf{u}(\boldsymbol{\mu}_i, t_m)\}_{\mathrm{batch}}$
7: $\{\mathbf{v}(\boldsymbol{\mu}_i, t_m)\}_{\mathrm{batch}} = \Psi\left(\{\mathbf{u}(\boldsymbol{\mu}_i, t_m)\}_{\mathrm{batch}};\, \boldsymbol{\vartheta}_{\mathrm{e}}^q\right)$
8: $\{\widehat{\mathbf{u}}(\boldsymbol{\mu}_i, t_m)\}_{\mathrm{batch}} = \boldsymbol{\Phi}\left(\{\mathbf{v}(\boldsymbol{\mu}_i, t_m)\}_{\mathrm{batch}};\, \boldsymbol{\vartheta}_{\mathrm{d}}^q\right)$
9: Compute the gradient $\nabla_{(\boldsymbol{\vartheta}_{\mathrm{e}}^q, \boldsymbol{\vartheta}_{\mathrm{d}}^q)} \mathcal{L}_{\mathrm{CAE}}$
10: Update $\left(\boldsymbol{\vartheta}_{\mathrm{e}}^{q+1},\, \boldsymbol{\vartheta}_{\mathrm{d}}^{q+1}\right) = \mathrm{ADAM}\left((\boldsymbol{\vartheta}_{\mathrm{e}}^q, \boldsymbol{\vartheta}_{\mathrm{d}}^q),\, \nabla_{(\boldsymbol{\vartheta}_{\mathrm{e}}^q, \boldsymbol{\vartheta}_{\mathrm{d}}^q)} \mathcal{L}_{\mathrm{CAE}}(\boldsymbol{\vartheta}_{\mathrm{e}}^q, \boldsymbol{\vartheta}_{\mathrm{d}}^q),\, \eta\right)$
11: **end for**
12: $N_{\mathrm{iter}} = N_{\mathrm{iter}} + 1$
13: **end while**

Step 2: Train the latent-space CNN

Input: Paired data $\{(\boldsymbol{\mu}_i, t_m),\, \mathbf{v}(\boldsymbol{\mu}_i, t_m)\}$ and hyperparameters
Output: Optimal CNN parameters $\boldsymbol{\vartheta}_{\mathrm{m}}^*$

14: Initialize $\boldsymbol{\vartheta}_{\mathrm{m}}$ randomly
15: Set $N_{\mathrm{iter}} = 0$
16: **while** (**!** early-stopping **and** $N_{\mathrm{iter}} \leq N_{\mathrm{epo}}$) **do**
17: for $k = 1 : N_{\mathrm{batch}}$ do
18: $q = N_{\mathrm{batch}} N_{\mathrm{iter}} + k$
19: Sample batch data $\{(\boldsymbol{\mu}_i, t_m),\, \mathbf{v}(\boldsymbol{\mu}_i, t_m)\}_{\mathrm{batch}}$
20: $\{\widehat{\mathbf{v}}(\boldsymbol{\mu}_i, t_m)\}_{\mathrm{batch}} = \Theta\left(\{(\boldsymbol{\mu}_i, t_m)\}_{\mathrm{batch}};\, \boldsymbol{\vartheta}_{\mathrm{m}}^q\right)$
21: Compute the gradient $\nabla_{\boldsymbol{\vartheta}_{\mathrm{m}}^q} \mathcal{L}_{\mathrm{Map}}(\boldsymbol{\vartheta}^q)$
22: Update $\boldsymbol{\vartheta}_{\mathrm{m}}^{q+1} = \mathrm{ADAM}\left(\boldsymbol{\vartheta}_{\mathrm{m}}^q,\, \nabla_{\boldsymbol{\vartheta}_{\mathrm{m}}^q} \mathcal{L}_{\mathrm{Map}}(\boldsymbol{\vartheta}_{\mathrm{m}}^q),\, \eta\right)$
23: **end for**
24: $N_{\mathrm{iter}} = N_{\mathrm{iter}} + 1$
25: **end while**

Step 2 (alternative): Train the latent-space LSTM

Input: Augmented data $\left\{[\mathbf{v}(\boldsymbol{\mu}_i, t_m); \boldsymbol{\mu}_i]\right\}$ and hyperparameters
Output: Optimal LSTM parameters $\boldsymbol{\vartheta}_l^*$

26: Randomly initialize $\boldsymbol{\vartheta}_l$;
27: Minimize $\mathcal{L}_{\mathrm{LSTM}}(\boldsymbol{\vartheta}_l)$ following similar steps outlined in lines 15 to 25.

Algorithm 2 Online stage with CAE-CNN

Input: New parameters $\boldsymbol{\mu}_{\mathrm{new}}$ and time t_{new}
Output: Predicted solution $\widehat{\mathbf{u}}(\boldsymbol{\mu}_{\mathrm{new}}, t_{\mathrm{new}}) \in \mathbb{R}^N$

1: Load the parameters $\boldsymbol{\vartheta}_{\mathrm{m}}^*$ and $\boldsymbol{\vartheta}_{\mathrm{d}}^*$ for the CNN and decoder, respectively
2: Input $(\boldsymbol{\mu}_{\mathrm{new}}, t_{\mathrm{new}})$ into the trained CNN to obtain $\widehat{\mathbf{v}} = \Theta\left(\boldsymbol{\mu}_{\mathrm{new}}, t_{\mathrm{new}};\, \boldsymbol{\vartheta}_{\mathrm{m}}^*\right)$;
3: Input $\widehat{\mathbf{v}}$ into the trained decoder to obtain $\widehat{\mathbf{u}}(\boldsymbol{\mu}_{\mathrm{new}}, t_{\mathrm{new}}) = \boldsymbol{\Phi}\left(\widehat{\mathbf{v}};\, \boldsymbol{\vartheta}_{\mathrm{d}}^*\right)$.

For time-dependent problems, if time-marching with LSTM is chosen, the prediction process is similar to that shown in Fig. 4, but with LSTM network instead. The corresponding algorithm is outlined in Algorithm 3. To predict the solution at time t_{m+1} for new parameters $\boldsymbol{\mu}_{\mathrm{new}}$, we need the solutions at the initial s steps, which are then encoded into the latent-space solution $\{\mathbf{v}(\boldsymbol{\mu}_{\mathrm{new}}, t_k)\}_{k=1}^s$. Then, the augmented data are input into the LSTM to predict the latent-space solution $\widehat{\mathbf{v}}(\boldsymbol{\mu}_{\mathrm{new}}, t_{m+1})$ step by step. Here, t_{m+1} is a time point where the training data are sampled, distinguishing it from t_{new} which is an arbitrary time in $[0, T]$. For time step t_p with $s+1 \leq p \leq 2s$, the mixed data of $\{\mathbf{v}(\boldsymbol{\mu}_{\mathrm{new}}, t_k)\}_{k=p-s}^s$ and $\{\widehat{\mathbf{v}}(\boldsymbol{\mu}_{\mathrm{new}}, t_k)\}_{k=s+1}^{p-1}$ are used in the latent space. For $p > 2s$, the LSTM inputs

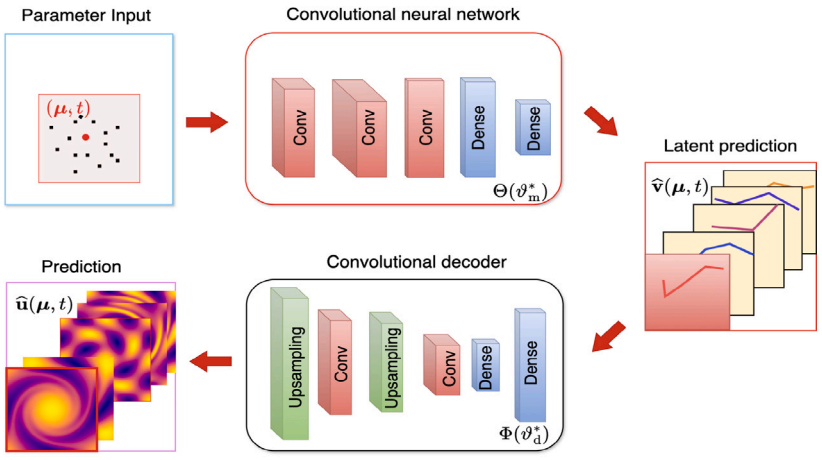

Fig. 4. Illustration of the online stage model with latent-space CNN mapping.

are the latent-space predictions $\{\hat{\mathbf{v}}(\boldsymbol{\mu}_{\text{new}}, t_k)\}_{k=p-s}^{p-1}$ from the previous s steps. Finally, the predicted latent-space solution $\hat{\mathbf{v}}(\boldsymbol{\mu}_{\text{new}}, t_{m+1})$ is decoded back to the high-dimensional space.

Algorithm 3 Online stage with CAE-LSTM

Input: New parameters $\boldsymbol{\mu}_{\text{new}}$, and initial solutions $\{\mathbf{u}(\boldsymbol{\mu}_{\text{new}}, t_k)\}_{k=1}^{s}$
Output: Predicted solution $\hat{\mathbf{u}}(\boldsymbol{\mu}_{\text{new}}, t_{m+1}) \in \mathbb{R}^N$

1: Load the parameters ϑ_e^*, ϑ_d^*, and ϑ_l^* for the CAE and LSTM
2: Input $\{\mathbf{u}(\boldsymbol{\mu}_{\text{new}}, t_k)\}_{k=1}^{s}$ into the trained encoder to obtain $\{\mathbf{v}(\boldsymbol{\mu}, t_k)\}_{k=1}^{s} = \Psi(\{\mathbf{u}(\boldsymbol{\mu}_{\text{new}}, t_k)\}_{k=1}^{s};\ \vartheta_e^*)$
3: **for** $p = s+1 : m+1$ **do**
4: $\quad \hat{\mathbf{v}}(\boldsymbol{\mu}_{\text{new}}, t_p) = \Gamma(\{[\mathbf{v}(\boldsymbol{\mu}_{\text{new}}, t_k); \boldsymbol{\mu}_{\text{new}}]\}_{k=p-s}^{s}, \{[\hat{\mathbf{v}}(\boldsymbol{\mu}_{\text{new}}, t_k); \boldsymbol{\mu}_{\text{new}}]\}_{k=s+1}^{p-1};\ \vartheta_l^*)$
5: **end for**
6: Input $\hat{\mathbf{v}}(\boldsymbol{\mu}_{\text{new}}, t_{m+1})$ into the trained decoder to obtain the predicted solution $\hat{\mathbf{u}}(\boldsymbol{\mu}_{\text{new}}, t_{m+1}) = \Phi(\hat{\mathbf{v}}(\boldsymbol{\mu}_{\text{new}}, t_{m+1});\ \vartheta_d^*)$

4. Benchmark tests: Nonlocal Poisson problems

In this section, we test the performance of our method on the fractional Poisson equation, which is a widely used benchmark for testing ROMs in nonlocal problems [12,22,23]. Solving the fractional Poisson problem is challenging due to the difficulties in approximating the fractional Laplacian (see, e.g. [5–7,9,31,41]). Its numerical discretization results in a full stiffness matrix, thus demanding significant storage and computational costs for matrix assembly and matrix–vector products, especially in high ($d > 1$) dimensions. Furthermore, the strong singularity of the fractional Laplacian requires special numerical treatments. However, some applications require solving the fractional Poisson equation repeatedly for different powers of α, which can be prohibitively costly [11,42]. To reduce computational costs, there has been growing interest in reduced-order methods, leading to the recent development of various reduced basis techniques for solving fractional Poisson equations [12,22,23].

Consider the d-dimensional Poisson equation on the domain $\Omega \subset \mathbb{R}^d$:

$$\begin{aligned} (-\Delta)^{\frac{\alpha}{2}} u(\mathbf{x}) &= f(\mathbf{x}), \qquad \text{for} \quad \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= g(\mathbf{x}), \qquad \text{for} \quad \mathbf{x} \in \mathbb{R}^d \backslash \Omega, \end{aligned} \tag{4.1}$$

for $\alpha \in (0, 2]$. If $\alpha < 2$, it is usually called as the fractional Poisson equation, and the d-dimensional fractional Laplacian $(-\Delta)^{\frac{\alpha}{2}}$ is defined in a hypersingular form [43]:

$$(-\Delta)^{\frac{\alpha}{2}} u(\mathbf{x}) = \frac{2^{\alpha-1}\alpha\,\Gamma\left(\frac{\alpha+d}{2}\right)}{\sqrt{\pi^d}\,\Gamma\left(1-\frac{\alpha}{2}\right)} \int_{\mathbb{R}^d} \frac{u(\mathbf{x}) - u(\mathbf{y})}{|\mathbf{x}-\mathbf{y}|^{d+\alpha}}\, \mathrm{d}\mathbf{y}, \qquad \text{for} \quad \alpha \in (0, 2).$$

If $\alpha = 2$, (4.1) gives the classical Poisson equation with boundary conditions on $\partial\Omega$. Without loss of generality, we assume that the right-hand side function f may also depend on parameters (see, e.g. (4.2) and (4.3)).

Throughout the paper, the training and testing data are high-fidelity numerical solutions computed using the numerical methods specified in each example. The ground truth solutions are also high-fidelity numerical solutions, computed with the same number of points as the training and testing data. For all examples, the detailed neural network architecture is summarized in Appendix A,

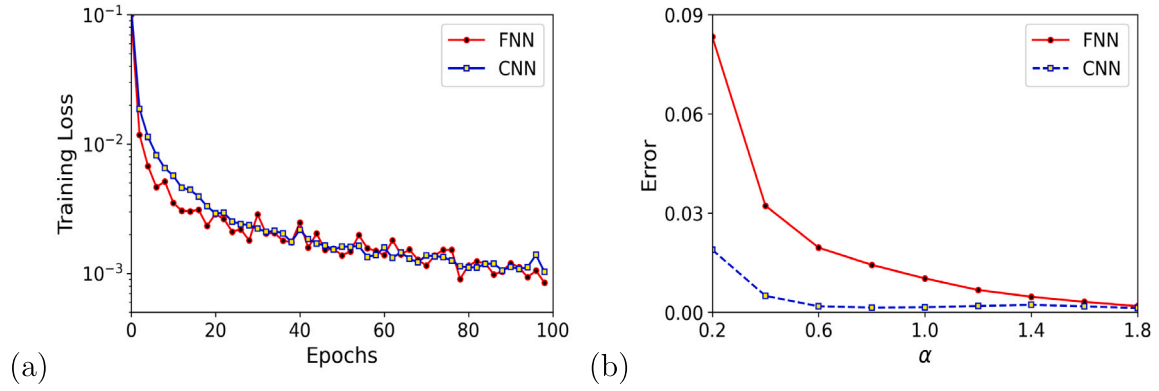

Fig. 5. Comparison of CNN and FNN mappings in the latent space, where the same CAE is used with the latent dimension $n = 6$. (a) Training loss versus epochs; (b) Errors across all test data versus α in the latent space.

while the specifics of the training and testing data are provided in Appendix B. The Adam algorithm is used in training both the CAE and the latent-space neural networks, with moment decay rates set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$, respectively. In addition, we use an adaptive learning rate defined as $\eta_{n+1} = \eta_0/(1 + n\delta_\eta)$ with $\eta_0 = 0.0003$, $\delta_\eta = 10^{-5}$, and n being the number of iterations. During the training process, an early stopping strategy is implemented to prevent overfitting.

Example 1. Let $d = 1$ and $\Omega = (-1, 1)$. Consider the 1D Poisson problem (4.1) with

$$f(x) = \sum_{n=1}^{N_c} a_n \sin(n\pi x), \quad \text{for } x \in (-1, 1); \qquad g(x) \equiv 0, \quad \text{for } x \in \mathbb{R}\backslash(-1, 1),$$

where we choose $N_c = 10$, and the coefficient $a_n \in [-1/n, 1/n]$ for $1 \le n \le N_c$. Then the model contains a total of 11 parameters, i.e., $\boldsymbol{\mu} = (\alpha, a_1, a_2, \dots, a_{N_c})$. To study the efficiency of CNN mapping in the latent space, we compare the training and test errors of CNN and FNN mappings (see Fig. 5), where the same CAE is used for dimensionality reduction.

Fig. 5(a) shows that both CNN and FNN converge quickly during the training process. However, the CNN has better generalization ability than FNN, as evidenced by smaller errors on the test data (see Fig. 5(b)). Moreover, CNN involves fewer training parameters. For example, when the latent dimension is $n = 6$, FNN has 84,870 parameters, while CNN has 69,510 parameters. By incorporating parameters and time through separate channels, the CNN achieves better performance in the latent space. The advantages of CNN become more significant for problems with high-dimensional parametric spaces.

In the following examples, we consider the 2D Poisson problem (4.1) and test the performance of our ROM. The training and testing data are computed using the finite difference method, as described [5,41,44], with $N_x = N_y = 512$ grid points in domain Ω. Hence, the data in the original space has a dimension of $N = N_x N_y = 262,144$.

Example 2. Set domain $\Omega = (0, 1)^2$. Consider a ν-dependent function [22]:

$$f(\mathbf{x}, \nu) = \nu^2 \sin(2\pi x)\sin(2\pi y) + (1 - \nu^2)\sin(3\pi x)\sin(3\pi y)e^{xy}, \tag{4.2}$$

for $\nu \in [0, 1]$, and homogeneous boundary conditions $g(\mathbf{x}) \equiv 0$ in (4.1). It is easy to see that the solution of (4.1)–(4.2) depends on both parameters α and ν. Let $\boldsymbol{\mu} = (\alpha, \nu)$. The number of parameters in the training and test datasets are $N_{\text{tr}}^{\boldsymbol{\mu}} = 72$ and $N_{\text{te}}^{\boldsymbol{\mu}} = 667$, respectively.

Fig. 6 compares our predicted solutions with the ground truth solutions for various parameters, where the latent dimension $n = 4$. If a colorbar is used, the same scale is applied to all results in that row. It shows that our method accurately predict solutions for new parameters, demonstrating the effectiveness of our surrogate model in solving the fractional Poisson equation, even with a small latent dimension of $n = 4$. Moreover, the CAE efficiently reduces the dimension from $N = 262,144$ to $n = 4$. Fig. 7 further shows the averaged MSE versus parameters α and ν. In all plots with shaded areas, the shading represents the range of maximum and minimum errors. We find that the errors are more sensitive to the parameter α; specifically, smaller values of α lead to larger errors in the predicted solutions. On the other hand, the errors for different ν remain in a similar range.

Example 3. Set domain $\Omega = (-1, 1)^2$. Consider the function:

$$f(\mathbf{x}, \mu_1, \mu_2) = \exp[-2((x - \mu_1)^2 + (y - \mu_2)^2)], \tag{4.3}$$

for $\mu_1, \mu_2 \in [-1, 1]$, and homogeneous boundary conditions $g(\mathbf{x}) \equiv 0$ in (4.1). Let $\boldsymbol{\mu} = (\alpha, \mu_1, \mu_2)$. The number of parameters in the training and test datasets are $N_{\text{tr}}^{\boldsymbol{\mu}} = 396$ and $N_{\text{te}}^{\boldsymbol{\mu}} = 931$, respectively.

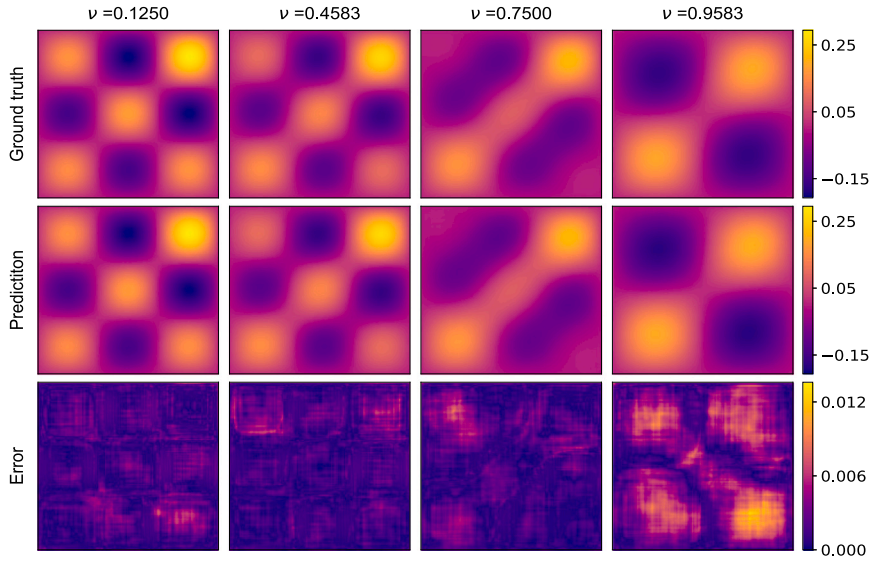

Fig. 6. Comparison of the ground truth and predicted solutions of the Poisson problem (4.1)–(4.2) with $\alpha = 0.765$ and different ν, where the latent dimension $n = 4$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

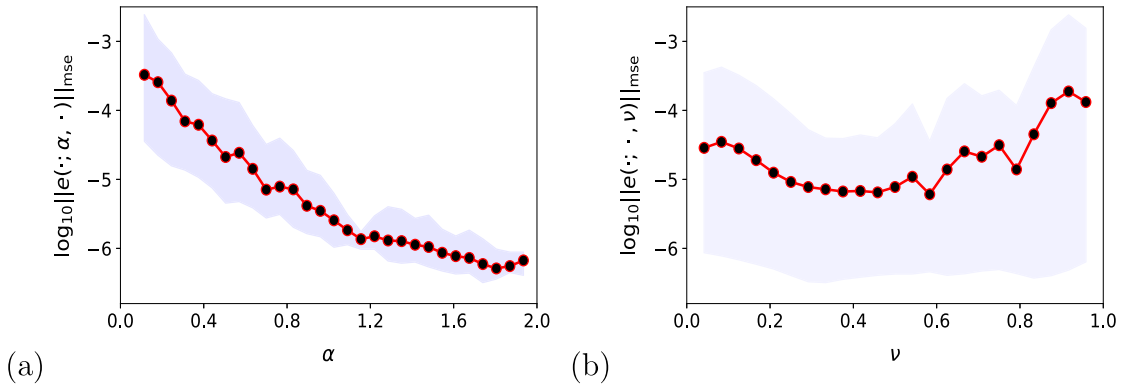

Fig. 7. MSE versus parameters α (a) and ν (b) in solving (4.1)–(4.2), where the latent dimension $n = 4$. The solid line is the average error across all test parameters α or ν.

Fig. 8 compares the predicted and ground truth solutions for various parameters, where the latent dimension $n = 4$. It shows that our method predicts accurate solutions for unseen new parameters, even with a latent dimension $n = 4$. The errors are relatively larger for smaller α. For given parameters (μ_1, μ_2), a smaller value of α results in a less smooth solution near the boundary, which generally affects the performance of numerical methods [6,41]. Moreover, Fig. 9(a) presents the errors across all parameters (μ_1, μ_2) in the test data. It again shows that our method provides accurate approximations with errors ranging from 10^{-7} to 10^{-4}. Fig. 9(b) compares errors across different latent dimensions. It shows that our model performs consistently well and has low sensitivity to latent dimensions.

To further evaluate its compression capability, Fig. 10 compares the reconstruction errors of the CAE and PCA. Note that latent-space modeling is not involved in this comparison. Fig. 10 shows that PCA's reconstruction error increases monotonically as the latent dimension (number of principal components) decreases. In contrast, the CAE's errors remain within a certain range for different latent dimensions. The CAE outperforms PCA, especially when solutions are less smooth. For instance, the solutions in Example 2 exhibit less regularity near the boundary. In such cases, the CAE achieves significantly smaller errors than PCA for latent dimensions $n \leq 12$ (see Fig. 10(b)). Generally, PCA is effective for problems that can be well approximated using low-rank representations. However, they face limitations when applied to complex nonlinear problems, especially when addressing parametric or regularity-related challenges. In such cases, CAEs, as nonlinear ROMs, offer a promising alternative.

Example 4. Set domain $\Omega = (-1, 1)^2$. Consider the 2D Poisson equation (4.1) with

$$f(\mathbf{x}) = 0, \quad \text{for } \mathbf{x} \in \Omega; \qquad g(\mathbf{x}) = \exp\left(-\nu\left((x-1)^2 + y^2\right)\right), \quad \text{for } \mathbf{x} \in \mathbb{R}^2 \backslash \Omega, \tag{4.4}$$

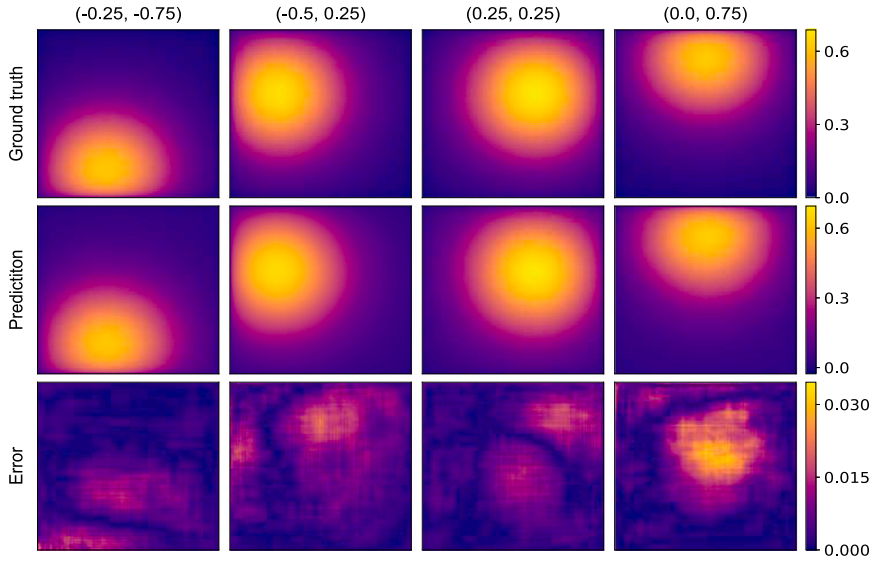

Fig. 8. Comparison of the ground truth and predicted solutions of the Poisson problem (4.1) and (4.3) for $\alpha = 0.48$ and different (μ_1, μ_2), where the latent dimension $n = 4$.

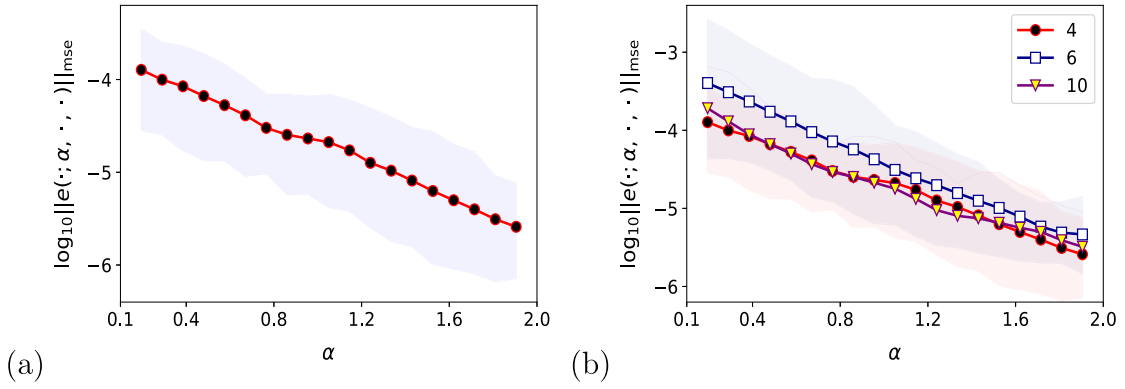

Fig. 9. MSE versus α with latent dimension $n = 4$ in (a) and $n = 4, 6, 10$ in (b). The solid line is the average error across all test parameters (μ_1, μ_2).

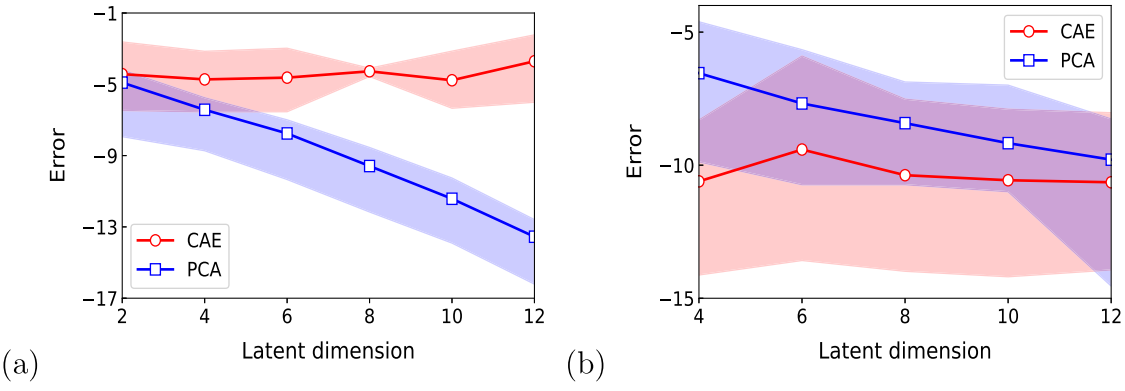

Fig. 10. Compression errors versus latent dimension of CAE and PCA in Example 2 (a) and Example 2 (b). The solid line is the average error across all test parameters.

for $\nu \in [1, 3]$. In this case, we evaluate the performance of our method in solving nonlocal Poisson problems with inhomogeneous boundary conditions. The number of parameters $\boldsymbol{\mu} = (\alpha, \nu)$ in the training and test datasets are $N_{\text{tr}}^{\boldsymbol{\mu}} = 54$ and $N_{\text{te}}^{\boldsymbol{\mu}} = 72$, respectively.

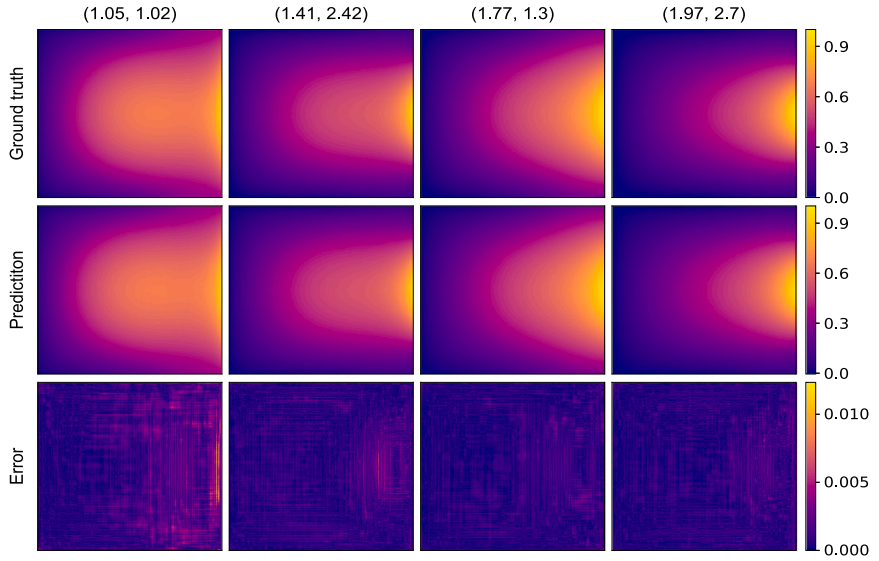

Fig. 11. Comparison of the ground truth and predicted solutions of the Poisson problem (4.1) and (4.4) for different (α, ν), where the latent dimension $n = 4$ and a hard boundary condition approach is used.

Fig. 11 compares the predicted and ground truth solutions for various parameters, where the latent dimension $n = 4$ and the hard boundary condition is implemented. In this case, the inhomogeneous boundary conditions on Ω^c are exactly incorporated via the construction function w. This leads to additional computational costs in constructing w and computing $(-\Delta)^{\frac{\alpha}{2}} w(\mathbf{x})$, however, the predicted solution is exact at the boundary $\partial\Omega$. Note that the challenge posed by the singular kernel in computing $(-\Delta)^{\frac{\alpha}{2}} w(\mathbf{x})$ can be addressed using well-established numerical methods for the fractional Laplacian [5,7,9,31]. Our results suggest that the hard boundary condition approach is effective in treating inhomogeneous nonlocal boundary conditions.

On the other hand, we also investigate the soft boundary condition approach. In this case, we consider various data domain sizes $\widehat{\Omega} = [-1, 1]^2$, $[-1.25, 1.25]^2$, $[-1.5, 1.5]^2$, and $[-2, 2]^2$, while the physical domain remains $[-1, 1]^2$. It is important to note that increasing the data domain size leads to higher computational costs when training the CAE. Our results show that when $\widehat{\Omega}$ is sufficiently large, the ROM behaves similarly to that with the hard boundary condition. However, unlike the hard approach, the soft boundary condition method may produce small, yet nonzero, errors at the boundary. In this case, the maximum errors occurs at the boundary $x = 1$.

Fig. 12 presents the errors $|u(1, y; \boldsymbol{\mu}) - \widehat{u}(1, y; \boldsymbol{\mu})|$ for different data domain $\widehat{\Omega}$. It shows that when the data domain is the same as the physical domain, the model fails to account for the influence of inhomogeneous boundary conditions outside the domain, leading to noticeably higher prediction errors on the boundary. Models trained on progressively larger data domains show clear improvements in boundary prediction accuracy, highlighting the benefit of capturing inhomogeneous boundary effects. However, this improvement requires only a moderate domain expansion (e.g., $\widehat{\Omega} = [-2, 2]^2$). Compared to the hard boundary condition approach, the soft approach reduces the computational cost of preparing training data by avoiding the construction of w and the evaluation of $(-\Delta)^{\frac{\alpha}{2}}$. However, the need for an extended data domain increases the training cost of the CAE. This highlights a distinct challenge in solving nonlocal problems. We will further explore inhomogeneous boundary conditions in future studies.

5. Nonlinear nonlocal problems

In this section, we extend our study to solving time-dependent nonlocal nonlinear problems arising in various applications. To the best of our knowledge, there are very few ROMs in the literature for nonlocal nonlinear problems [45]. Solving nonlinear nonlocal problems is particularly challenging due to the combined effects of nonlocality and nonlinearity. These problems often require significantly higher storage and computational resources. Moreover, if an implicit time-stepping method is used, numerical iterations are required at each time step, further increasing computational costs. On the other hand, numerical schemes for temporally nonlocal problems typically require historical solution data, leading to significant storage costs, particularly in high spatial dimensions. Hence, it is challenging to solve these problems if the computation time is limited, such as in real-time predictions.

5.1. Nonlocal diffusion-reaction equation

Nonlocal reaction–diffusion models are applied in various biological applications, including tissue dynamics [46], the spread of epidemic [47], and cancer cell population's dynamics [48]. Here, we consider the 1D nonlocal diffusion-reaction of the

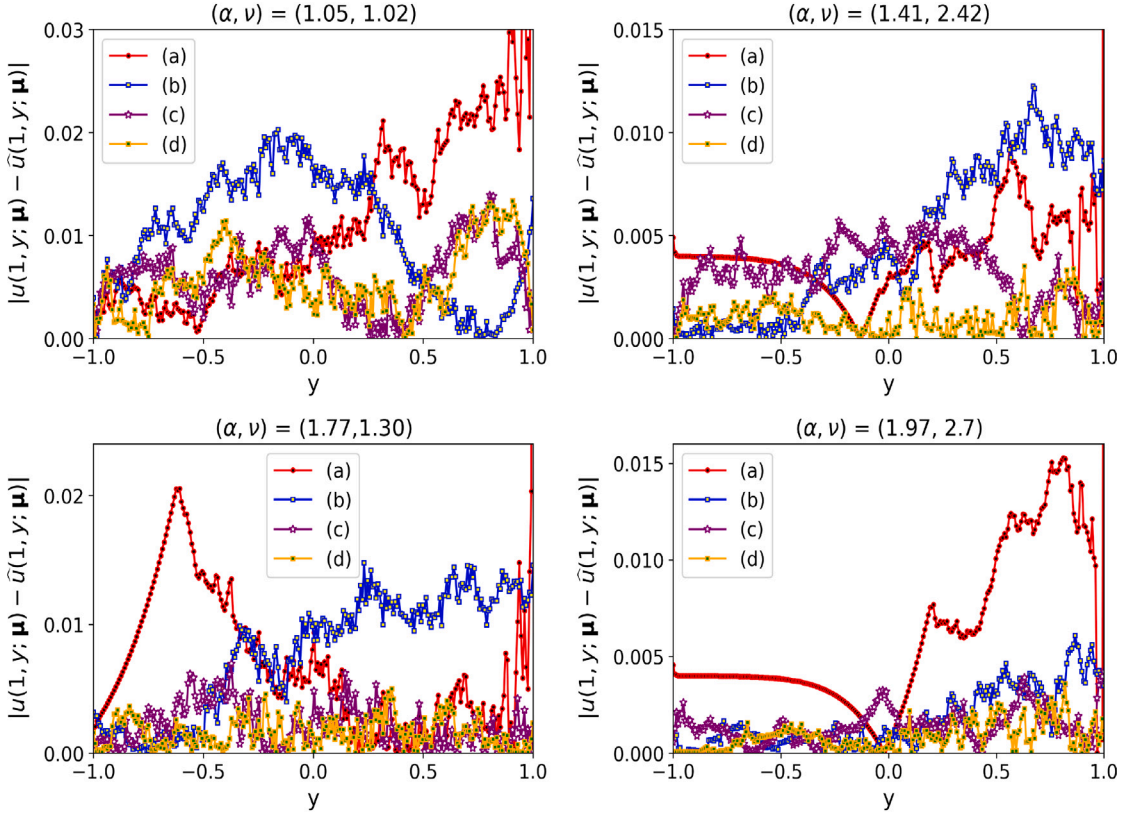

Fig. 12. Errors along the boundary $x = 1$ for various evaluated parameter sets, with $\hat{\Omega} = [-1,1]^2, [-1.25,1.25]^2, [-1.5,1.5]^2$ and $[-2,2]^2$ for cases (a)–(d), respectively. The errors at the right endpoint $y = 1$ have been truncated for better illustration.

form [6,12,27]:

$$\partial_t u(x,t) = C_{\gamma,\alpha,\delta} \int_{x-\delta}^{x+\delta} \frac{u(x,t) - u(x',t)}{|x - x'|^{1+\alpha}} \, \mathrm{d}x' + u^2, \quad \text{for } x \in \Omega, \quad t > 0, \tag{5.1}$$

$$u(x,t) = 0, \quad \text{for } x \in \Omega_\Gamma, \quad t \geq 0, \tag{5.2}$$

$$u(x,0) = \sum_{k=1,2,3} \sin(k\pi x), \quad \text{for } x \in \bar{\Omega}, \tag{5.3}$$

where $C_{\gamma,\alpha,\delta} = \gamma(\alpha - 2)/\delta^{(2-\alpha)}$, for γ, $\delta > 0$ and $\alpha \in [0, 2)$. The integral operator in (5.1) describes nonlocal diffusion with a finite interaction region, which can be viewed as a truncated fractional Laplacian [4,6,12]. Its numerical discretization yields a banded matrix, with the bandwidth depending on the horizon δ and mesh size h. Particularly, the discretization in high dimensions becomes more challenging, as the interaction region is a ball $B_\delta(\mathbf{x})$ centered at $\mathbf{x}$ with radius δ. Accurately capturing integration over this ball for every point $\mathbf{x}$ has been a long-standing challenge [6].

Set the domain $\Omega = (-1, 1)$, and then $\Omega_\Gamma = [-1-\delta, -1] \cup [1, 1+\delta]$. Let $\gamma = 0.1$, and the parameters $\alpha \in [0.1, 1.9]$ and $\delta = [0.125, 0.5]$. Here, we test the performance of our surrogate model in solving (5.1)–(5.3) with various parameter sets $\boldsymbol{\mu} = (\alpha, \delta)$. To prepare training and testing data, we numerically solve (5.1)–(5.3) using finite difference methods in space [5] and the fourth-order Runge–Kutta method in time. The solution is computed with $N_x = 513$ uniformly distributed points on $[-1, 1]$ and $N_t = 10001$ time steps on $[0, 1]$. The number of parameters in the training and test data are $N_{\text{tr}}^{\boldsymbol{\mu}} = 104$ and $N_{\text{te}}^{\boldsymbol{\mu}} = 444$, respectively. For each parameter $\boldsymbol{\mu}_i$, we include $N_{\text{tr}}^t = 11$ time snapshots, uniformly sampled over $[0, 1]$. Note that $N_{\text{tr}}^t \ll N_t$.

Fig. 13 compares the predicted and ground truth solutions at different times for various parameters, where the CNN mapping is used in latent space with latent dimension $n = 4$. The CNN structure is similar as that in Section 4, except that time t is treated as an additional parameter for time-dependent problems. It shows that the predicted solutions agree well with the ground truth solutions for unseen parameters and times. Despite training with only $N_{\text{tr}}^t = 11$ time snapshots, the CNN learns the mapping $(\alpha, \delta, t) \mapsto u(\cdot, t; \alpha, \delta)$ for any time $t \in (0, 1]$. This is a key difference from the LSTM-based model. Our extensive studies show that the CAE-LSTM predicts accurate solutions for times $t \in S_{\text{tr}}^t$. For brevity, we omit showing its results.

Fig. 14(a) shows the average MSE for α in the test data, while Fig. 14(b) displays the average MSE across all test parameters (α, δ). It shows that the errors across time t range from 10^{-7} to 10^{-2}, without increasing over time as observed in traditional numerical

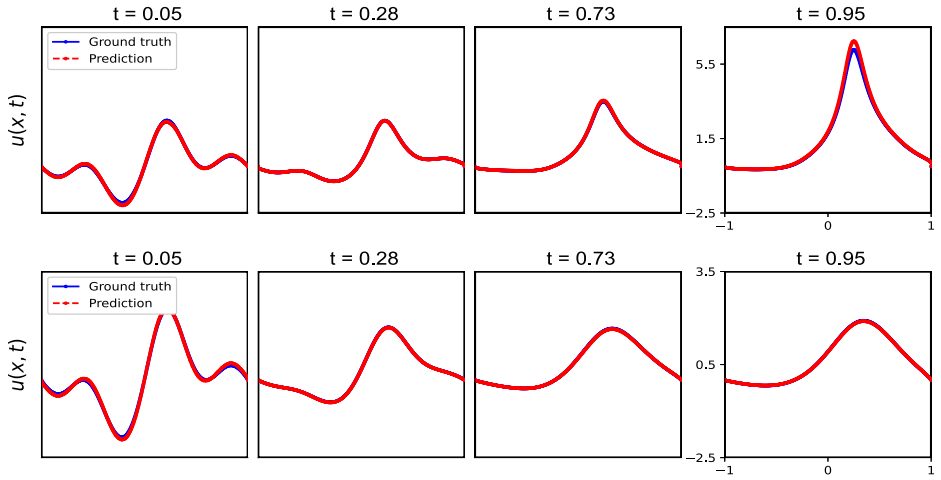

Fig. 13. Solution dynamics of (5.1)–(5.3) with $\delta = 0.48438$, and $\alpha = 0.1425$ (top row) or 1.615 (bottom row), where the CNN mapping is used with latent dimension $n = 4$.

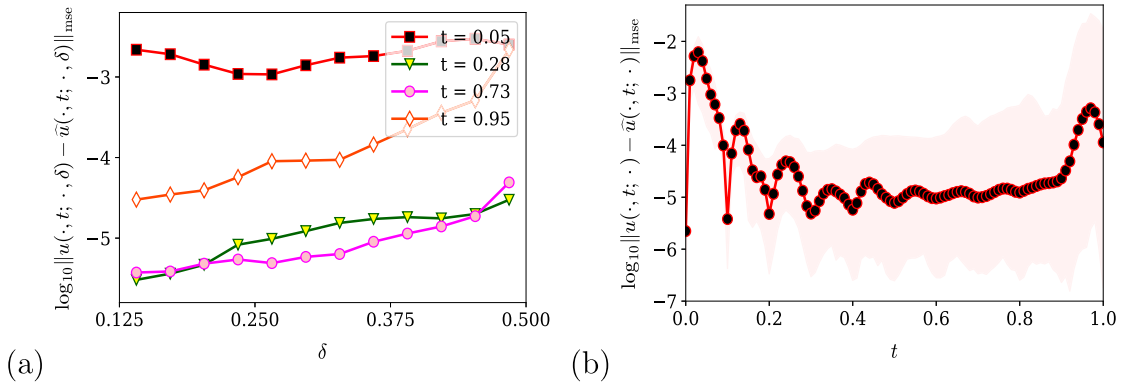

Fig. 14. (a) Average MSE over S_{te}^{α}; (b) average MSE across all test parameters (α, δ).

methods. Moreover, the errors are relatively large when t is small (e.g., $t = 0.05$), because of the rapid changes of the solution near $t = 0$. Our extensive studies show that including more time snapshots, for example, in the interval $[0, 0.2]$, can effectively reduce these errors. During the online stage, we input (α, δ, t) into the latent-space CNN to generate predictions, which are then decoded back into the original space. It is significantly faster compared to traditional numerical methods.

5.2. Fractional Burgers' equation

The fractional Burgers' equation describes the physical process of unidirectional propagation of weakly nonlinear acoustic waves through a gas-filled pipe [49]. The time-fractional derivative results from the cumulative (memory) effect of the wall friction through the boundary layer. Similar forms can be found in other systems such as shallow water wave [50] and waves in bubbly liquids [51]. Consider the time fractional Burgers' equation of the form [49,52]:

$$ {}_0^C D_t^{\beta} u(x,t) + \frac{1}{2}(u^2)_x = \nu \Delta u, \quad \text{for } x \in \Omega, \quad t \in (0,1], \tag{5.4} $$

$$ u(x,t) = 0, \quad \text{for } x \in \partial\Omega, \quad t \in [0,1], \tag{5.5} $$

$$ u(x,0) = -\sin(\pi x), \quad \text{for } x \in \bar{\Omega}. \tag{5.6} $$

The Caputo fractional derivative ${}_0^C D_t^{\beta}$ is a nonlocal operator, defined as

$$ {}_0^C D_t^{\beta} w(t) = \frac{1}{\Gamma(1-\beta)} \int_0^t \frac{w'(s)}{(t-s)^{\beta}} \, \mathrm{d}s, \qquad \text{for } \beta \in (0,1). $$

If $\beta \to 1$, it reduces to the classical first-order derivative. Due to their nonlocality in time, solving time-fractional PDEs requires significant storage costs to save the historic solutions from time $t = 0$. This makes long-time simulations challenging and has led to

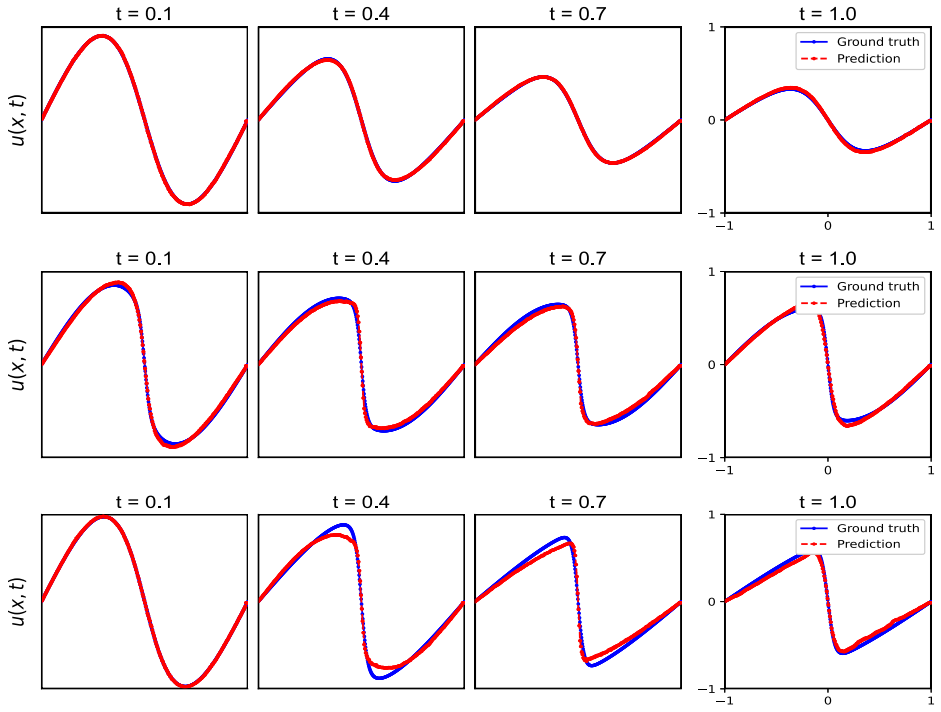

Fig. 15. Comparison of the ground truth and predicted solutions of (5.4)–(5.6), where LSTM is used in the latent space with dimension $n = 2$. From top to bottom: $(\beta, \nu) = (0.98, 0.0958)$, $(0.548, 0.0166)$, $(0.956, 0.0232)$.

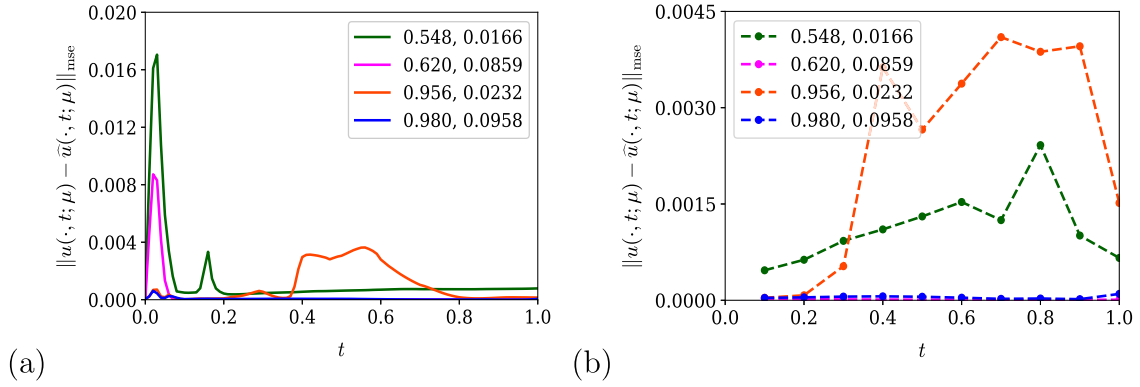

Fig. 16. MSE over time for different parameters (β, ν) for CNN (a) and LSTM (b).

numerous efforts to reduce storage burdens [10]. Moreover, the singularity in the Caputo derivative requires careful discretization. These numerical challenges create a significant bottleneck, impeding the study of time-fractional PDEs. Here, our surrogate model provides a neural network approach to addressing these challenges.

Let $\Omega = (-1, 1)$, $\beta \in [0.5, 1]$, and $\nu \in [0.01, 0.1]$. The training and testing data are generated using a finite difference approximation for the time-fractional derivative and a central difference for the spatial derivatives [10], with $N_x = 512$ and $N_t = 1001$. Let $\boldsymbol{\mu} = (\beta, \nu)$. The number of parameters in the training and test datasets are $N_{\text{tr}}^{\boldsymbol{\mu}} = 30$ and $N_{\text{te}}^{\boldsymbol{\mu}} = 540$, respectively. For each parameter $\boldsymbol{\mu}_i$, we include $N_{\text{tr}}^{t} = 11$ uniformly sampled time snapshots on $[0, 1]$. In the following, we will study both CAE-LSTM and CAE-CNN and compare their performance. Fig. 15 compares the ground truth and predicted solutions for different parameters (β, ν), where the CAE-LSTM is used with latent dimension $n = 2$. The latent-space neural network includes three LSTM layers with 128 neurons in each layer, followed by two fully-connected layers with 128 and 2 neurons, respectively. Fig. 15 shows that the predicted solutions agree well with the ground truth solutions even with latent dimension $n = 2$. Note that using the CNN-mapping in the latent space yields similar predicted solutions as those in Fig. 15.

The main differences of LSTM and CNN lie in: LSTM requires uniformly sampled time snapshots on $[0, T]$ and can only predict solutions at those specific times. In contrast, CNN learns a continuous mapping and can predict solutions at any time within $[0, T]$.

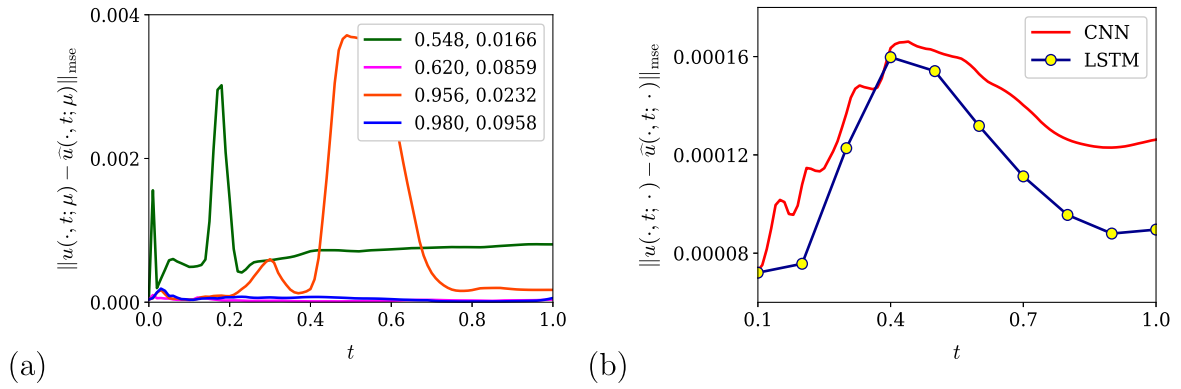

Fig. 17. (a) Same study as Fig. 16(a), showing the benefit of nonuniform time-step data with higher resolution in $[0, 0.1]$; (b) Average MSE of CAE-CNN and CAE-LSTM across all test parameters (β, ν).

This difference is further illustrated in Fig. 16. For both CAE-CNN and CAE-LSTM, the relative errors at the snapshot time points in S_{tr}^t are under 0.15%. However, the CAE-CNN can predict solutions at any time within $[0, 1]$, with overall errors remaining below 2%. If both β and ν are small (e.g., $\beta = 0.548$, $\nu = 0.0166$), the solution changes quickly at the initial stage, which results in large errors. We can include more snapshot points, e.g. in $(0, 0.1)$, to reduce the errors and improve the prediction in the initial stage. For example, Fig. 17(a) shows the results where the CNN is trained with nonuniform time points, specifically with additional snapshot points (i.e. $t = 0.02, 0.04, 0.06, 0.08$) included in the training data. Comparing Figs. 16(a) and 17(a), it is clear that using nonuniform time steps can greatly benefit time-multiscale problems by avoiding using small time steps across the entire range, thus significantly reducing training costs.

Fig. 17 presents the average MSE for all test parameters. It shows that CAE-LSTM method achieves smaller errors at snapshot time points. To predict solutions between snapshot times, it requires more training snapshots. The online prediction of our method requires significantly less computational time, while the storage costs can be ignored. Particularly, CAE-CNN method requires less time than CAE-LSTM – CNN directly maps the inputs $(\beta, \nu, t_{\text{new}})$ to the latent-space solution, while LSTM computes from t_0 to t_m step by step in the latent space, consequently taking more computational time. However, their computational and storage costs can be ignored compared to the traditional numerical methods [10]. In traditional numerical methods, to find the solution at time t_{m+1}, the numerical schemes need the information of all previous steps from t_0 to t_m. Hence, the smaller the time step, the larger the storage cost, especially in high-dimensional cases [10].

5.3. Nonlocal Allen–Cahn equation

The nonlocal Allen–Cahn equation has been used to study phase transitions in the presence of anomalous diffusion [41,53]. Consider the 2D nonlocal Allen–Cahn equation of the form:

$$\partial_t u(\mathbf{x}, t) = \gamma \int_{\Omega} J_\varepsilon(\mathbf{x} - \mathbf{x}')\big(u(\mathbf{x}', t) - u(\mathbf{x}, t)\big)\, \mathrm{d}\mathbf{x}' + \beta(u - u^3), \tag{5.7}$$

for $\mathbf{x} \in \Omega$ and $t \in (0, T]$, where $\gamma, \beta > 0$. The parameterized kernel function J_ε, depending on the distance of two points, is given by $J_\varepsilon(\mathbf{x} - \mathbf{x}') = \exp(-|\mathbf{x} - \mathbf{x}'|^2/\varepsilon^2)/\varepsilon$ for $\varepsilon > 0$. The initial condition is taken as

$$u(\mathbf{x}, 0) = 2\cos(2\pi x)\cos(2\pi y), \qquad \text{for } \mathbf{x} \in \bar{\Omega}. \tag{5.8}$$

Set $\Omega = (-1, 1)^2$, $T = 10$, and $\gamma = 10$ in (5.7). Here, we apply our method to solve (5.7)–(5.8) for parameters $\varepsilon \in [0.05, 0.1]$ and $\beta \in [0.1, 0.5]$.

The training and testing data are numerical solutions obtained using the semi-implicit finite difference method in space [53] and the fourth-order Runge–Kutta method in time, where a uniform mesh size and time step are used with $N_x = N_y = 128$ and $N_t = 10000$. Let $\boldsymbol{\mu} = (\varepsilon, \beta)$. The number of parameters in the training and test datasets are $N_{\text{tr}}^{\boldsymbol{\mu}} = 30$ and $N_{\text{te}}^{\boldsymbol{\mu}} = 336$, respectively. For each parameter $\boldsymbol{\mu}_i$, we include $N_{\text{tr}}^t = 29$ non-uniform time snapshots from $[0, 10]$.

Fig. 18 presents the ground truth and predicted solutions of the Allen–Cahn equation for $\varepsilon = 0.0955$ and $\beta = 0.468$, where the latent dimension $n = 4$. Our extensive studies show that the CAE-LSTM method produces similar results. Thus, we will omit these results for the sake of brevity. Our predicted solutions agree well with the ground truth solution for any time over $[0, 10]$. Fig. 19 further shows the performance of CAE, latent-space modeling CNN, and their combination CAE-CNN across all test parameters. It shows that the overall performance of our ROMs depends on the accuracy of CAE and latent-space modeling. Therefore, to enhance accuracy, efforts should focus on improving these two components. Our decoupled training strategy makes it easier to control this improvement process.

To further assess its efficiency, Table 1 compares the computational times (in second) of our method and the finite difference method in [53]. The computations are performed on a macOS Monterey system equipped with a 10-core 3.6 GHz Intel i9 CPU and

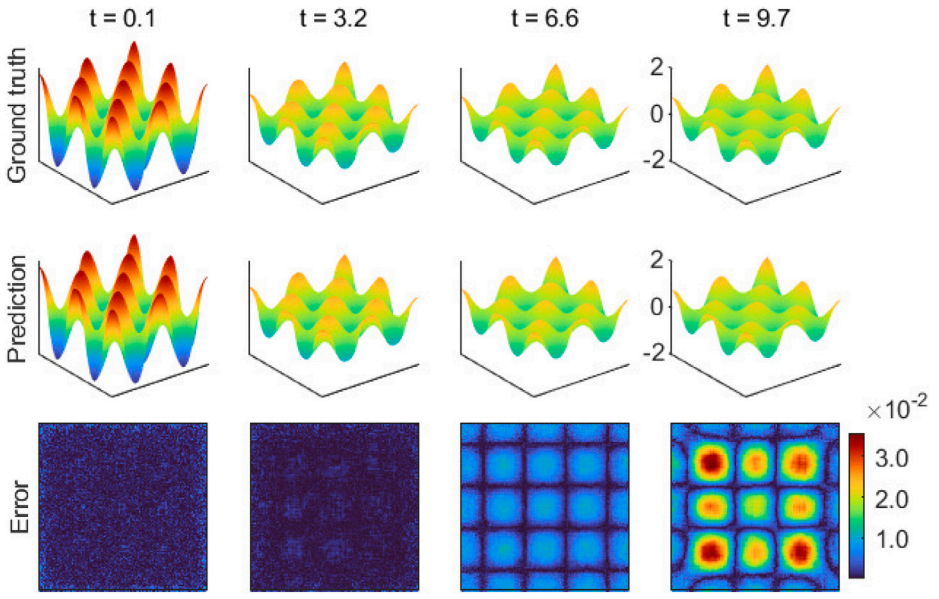

Fig. 18. Comparison of the ground truth and predicted solutions of the Allen–Cahn equation (5.7) with $\epsilon = 0.0955$ and $\beta = 0.468$, where the latent dimension $n = 4$.

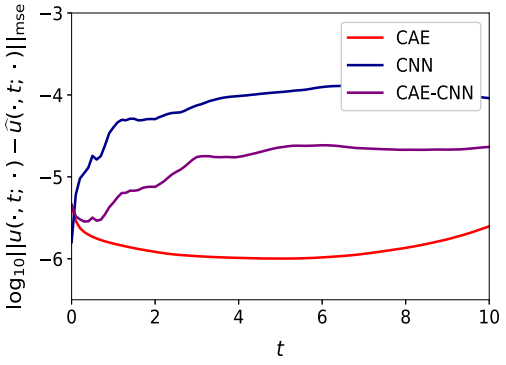

Fig. 19. Average MSE of the CAE, CNN, and their combination CAE-CNN across all test parameters.

128 GB of 2667 MHz DDR4 memory. Due to the symmetric kernel function J_ϵ, numerical discretization of the nonlocal operator in (5.7) results in a dense symmetric Toeplitz matrix. Hence, the matrix–vector multiplication in this example is performed via the fast Fourier transforms with computational cost of $\mathcal{O}(2N\log(2N))$. Note that the numerical discretization of a general nonlocal operator may lack desirable properties, preventing efficient FFT computations and significantly increasing both computational and storage demands [8].

For our method, we provide the time spent on both the offline training and online prediction. Specifically, we break down the offline training time into two parts: training the CAE and training the latent CNN. The reported training times for CAE and CNN are averaged over three independent runs. It shows that: (i) The time of training the CAE dominates the offline stage time, mainly depending on the data dimension and volume. (ii) The training time for the CNN is relatively insignificant compared to the CAE. Moreover, we present the total computational time spent to predict the solution for $N_{\text{pred}}^{\boldsymbol{\mu}} = 5000$ parameter sets $\boldsymbol{\mu}$ at different times t. For each new parameter $\boldsymbol{\mu}$, traditional numerical methods start from initial conditions and compute step by step to reach the solution at the desired time t. Consequently, the computation time is proportional to the time t – the larger the time t, the longer the computational time (cf. the times for $t = 1$ and $t = 10$). In contrast, our surrogate model predicts the solution directly as $\hat{\mathbf{u}} = \boldsymbol{\Phi}(\boldsymbol{\Theta}(\boldsymbol{\mu}, t; \vartheta_{\text{m}}^*); \vartheta_{\text{d}}^*)$, involving only the trained CNN and decoder. Hence, the computational time of our method for new parameters are considerable less than that used by traditional numerical methods. Our method is more efficient for computing solutions of parametric PDEs across different parameters (e.g., in optimal control problems).

Table 1
Computational time of our method and numerical method in [53]. The times listed under columns for $t = 1$ and $t = 10$ are the total computational time spent to compute the solution $u(\cdot, t; \epsilon, \beta)$ for 5000 parameter sets (ϵ, β).

N	Our method				Semi-implicit FDM	
	Offline training		Online prediction			
	CAE	CNN	$t = 1$	$t = 10$	$t = 1$	$t = 10$
2^{14}	3.552e+4	230	2.999	3.050	3.264e+4	3.334e+5
2^{16}	9.107e+4	443	9.074	9.095	5.335e+5	5.240e+6

6. Summary and discussion

In this work, we proposed the neural network-based ROM methods for solving nonlocal parametric problems. Our approach consists of two main components: dimensional reduction with convolutional autoencoder (CAE) and latent-space modeling with CNN or LSTM. In the CAE, the encoder compresses high-dimensional data into a low-dimensional representation, while the decoder reconstructs the high-dimensional data from the latent space approximation. In the latent space, we proposed a CNN to map the model parameters $\boldsymbol{\mu}$ and time t to the low-dimensional representation. This latent-space mapping approach works for both time-dependent and time-independent problems. For time-dependent problems, we also proposed an LSTM network to model the temporal dynamics in the latent space. To enhance the efficiency of our neural network-based ROM, we applied a decoupled training strategy for the CAE and the latent space models.

Our studies introduced the first neural network-based ROMs for nonlocal problems. Using CAE for dimension reduction has proven effective in studying ROMs for nonlocal problems. It bypasses the challenges faced by the intrusive ROM methods for nonlocal problems, such as the lack of affine dependence and the complications associated with singularity and nonlocal boundary conditions. During the online stage, our CAN-CNN and CAE-LSTM operate without using the nonlocal PDEs. Extensive numerical experiments, including the benchmark Poisson problems and nonlinear nonlocal problems, demonstrated the accuracy and efficiency of our proposed neural network-based ROMs. Compared to traditional numerical methods, our neural network-based approach significantly reduces computational and storage costs in the study of nonlocal parametric PDEs.

Numerical experiments demonstrated that our method exhibits strong generalization performance within the studied parameter space and time domain. However, it has limitations when extrapolating beyond the studied time frame, which presents an area for potential improvement in the current approach. Generally, the extrapolation of ROMs depends on the solution properties of the problem. If the solution exhibits periodic or quasi-periodic behavior in time, our methods could provide accurate extrapolation. Our CAE-based ROM performs effectively for structured data. For unstructured data, an additional step, such as using space-filling curves [54] or interpolation techniques [55], is required to convert the data into a grid-based structure. In the future, we plan to explore graph neural network-based ROMs for unstructured data [40] and geometric deep learning for irregular domains [56].

CRediT authorship contribution statement

Yumeng Wang: Writing – original draft, Visualization, Methodology, Investigation. **Shiping Zhou:** Writing – original draft, Formal analysis, Data curation. **Yanzhi Zhang:** Writing – review & editing, Supervision, Methodology, 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.

Acknowledgments

This work was partially supported by the US National Science Foundation under grant number DMS-1953177. We sincerely thank the two anonymous reviewers for their insightful comments and helpful suggestions, which significantly improved the manuscript.

Appendix A. Neural network architecture

The encoder consists of convolutional layers, pooling layers, and fully connected layers. Zero-padding is applied to all convolutional layers. Each convolutional layer is followed by a max-pooling layer, together forming a convolutional block. The pooling size is a hyperparameter chosen based on the input data dimension and the desired output dimension. Note that pooling layers do not involve any trainable parameters. Fully-connected layers are added after the last convolutional block. The last fully-connected layer, also known as the "bottleneck layer", transforms data into a low-dimensional representation of the desired dimension. The ReLU activation functions are applied to all convolutional and fully-connected layers, except for the last fully-connected layer, which has no activation function.

Table 2
Convolutional encoder and decoder (repeated pattern omitted) for the nonlocal Poisson problems.

Encoder			Decoder		
Layer	Input	Output	Layer	Input	Output
Conv2D	$512^2 \times 1$	$512^2 \times 4$	Dense	4	50
Maxpool2D	$512^2 \times 4$	$256^2 \times 4$	Dense	50	100
Conv2D	$256^2 \times 4$	$256^2 \times 8$	Dense	100	16
Maxpool2D	$256^2 \times 8$	$128^2 \times 8$	Reshape	16	$4^2 \times 1$
⋮	⋮	⋮	Conv2D	$4^2 \times 1$	$4^2 \times 256$
Conv2D	$8^2 \times 128$	$8^2 \times 256$	Upsampling2D	$4^2 \times 256$	$8^2 \times 256$
Maxpool2D	$8^2 \times 256$	$4^2 \times 256$	⋮	⋮	⋮
Flatten	$4^2 \times 256$	4096	⋮	⋮	⋮
Dense	4096	100	Conv2D	$256^2 \times 8$	$256^2 \times 4$
Dense	100	50	Upsampling2D	$256^2 \times 4$	$512^2 \times 4$
Dense	50	4	Conv2D	$512^2 \times 4$	$512^2 \times 1$

Table 3
Latent space CNN-mapping (two convolution layers omitted) for the nonlocal Poisson problems.

Layer	Input	Output
Conv1D	1×2	1×128
Conv1D	1×128	1×128
⋮	⋮	⋮
Conv1D	1×128	1×128
Flatten	1×128	128
Dense	128	128
Dense	128	4

The decoder consists of convolutional layers, upsampling layers, and fully-connected layers. The structure of fully-connected layers and convolutional layers is similar to that in the encoder. Specifically, the number of neurons in the last fully-connected layer is determined in coordination with other hyperparameters (such as the number of upsampling layers, upsampling sizes and number of convolutional layer) and the desired output dimension N. In contrast to the encoder, the decoder uses upsampling layers to increase the dimensions of the data. There are different upsampling approaches, such as interpolations, unpooling, and transposed convolutions. In our model, we use the nearest-neighbor interpolation method. Note that the upsampling layers also do not involve any trainable parameters.

Tables 2 and 3 describe the neural network structure for the nonlocal Poisson problems in Section 4. In Table 2, the encoder input a dimension of $N_x \times N_y \times c$, where c denotes the number of channels. The convolutional encoder consists of seven convolutional blocks, each with a kernel size of $(3, 3)$ and a stride of one. In the pooling layers, a pooling size of $(2, 2)$ with a stride of two is chosen, effectively reducing the spatial dimensions by half along each axis at every pooling step. After flattening, three fully connected layers contain 100, 50, and 4 neurons, respectively, where the number of neurons in the final layer is consistent with the latent dimension. The convolutional decoder mirrors the structure of convolutional encoder, with an additional fully connected layer with 16 neurons implemented before applying the convolutional blocks.

The neural network structures for other problems are similar to those in the nonlocal Poisson problem with variations if needed. When processing one-dimensional examples, the corresponding two-dimensional structures are replaced with their one-dimensional counterparts. For example, in Section 5.1, an additional dense layer maps the data dimension from 513 to 512, followed by a similar pattern as in the Poisson problems. Then seven convolutional blocks with $4, 8, 16, 32, 64, 128, 256$ filters are followed to reduce the dimensions. Instead of using several fully connected layers, a convolutional layer with one filter is used to average among the channels. Similarly, in Section 5.2, for input data of dimension 512×1, the architecture consists of eight convolutional blocks with $2, 4, 8, 16, 32, 64, 128, 256$ filters, yielding a latent space of dimension two. In Section 5.3, for the input data of dimension $128 \times 128 \times 1$, five convolutional blocks with $8, 32, 64, 128, 256$ filters are adopted followed with two dense layers with 50 and 4 neurons, respectively.

Note that latent space modeling is independent of the problem and the data dimensions. In Table 3, the CNN-mapping takes input as $1 \times p$ and an additional parameter t is included for time-dependent PDEs. It consists of five convolutional layers, each has 128 filters with a kernel size of two, and a stride of one. Two dense layers with 128 and 4 neurons are followed, respectively. In particular, Sections 5.1–5.3 each have $2 + 1$ parameters to adjust the input of 1×3 and their latent dimension 4, 2, and 4 as last layer neurons, respectively. The ReLU activation function is applied.

The hyperparameters of our method include those related to both the neural network architecture and the model training process. For the neural network, hyperparameters such as the number of neurons, number of filters, number of layers, kernel size, pooling size, and activation function are tuned by evaluating the loss, similar to the approach in [57]. In the optimizer and training process, hyperparameters, such as the learning rate and batch size, are manually adjusted to achieve the current performance. Due to the high computational cost, automated search-based hyperparameter tuning methods [58] were not selected. Moreover, the latent dimension plays an important role in ROMs methods. Therefore, we analyze and assess the performance across different latent dimensions.

Table 4
Parameters sampling for the training and testing data. The notation $a : n : b$ represents picking every n equispaced value from a to b.

Nonlocal Poisson equation (Example 1), $\boldsymbol{\mu} = (\alpha, a_1, \dots, a_{10})$	
Training	$S_{\text{tr}}^{\alpha} = \{0.1 : 0.1 : 1.9\}$, $S_{\text{tr}}^{a_n} = \{\pm 1/n, n = 1, \dots, 10\}$
Testing	$S_{\text{te}}^{\alpha} = \{0.2 : 0.1 : 1.8\}$, $S_{\text{te}}^{a_n} = \pm\{\frac{1}{2}, \frac{1}{4}, \frac{1}{6}, \frac{3}{20}, \frac{1}{10}, \frac{14-n}{100}, n = 6, \dots, 10\}$
Nonlocal Poisson equation (Example 2), $\boldsymbol{\mu} = (\alpha, \nu)$	
Training	$S_{\text{tr}}^{\alpha} = \{0.05, 0.1 : 0.2 : 0.9, 1 : 0.2 : 2\}$, $S_{\text{tr}}^{\nu} = \{0 : 0.2 : 1\}$
Testing	$S_{\text{te}}^{\alpha} = \{0.115 : 0.065 : 1.935\}$, $S_{\text{te}}^{\nu} = \{0 : 1/22 : 1\}$
Nonlocal Poisson equation (Example 3), $\boldsymbol{\mu} = (\alpha, \mu_1, \mu_2)$	
Training	$S_{\text{tr}}^{\alpha} = \{0.1 : 0.2 : 0.9, 1 : 0.2 : 2\}$, $S_{\text{tr}}^{\mu_1} = S_{\text{tr}}^{\mu_2} = \{-1 : 0.4 : 1\}$
Testing	$S_{\text{te}}^{\alpha} = \{0.195 : 0.09 : 1.905\}$, $S_{\text{te}}^{\mu_1} = S_{\text{te}}^{\mu_2} = \{-0.75 : 0.25 : 0.75\}$
Nonlocal Poisson equation (Example 4), $\boldsymbol{\mu} = (\alpha, \nu)$	
Training	$S_{\text{tr}}^{\alpha} = \{1 : 0.2 : 1.8, 1.99\}$, $S_{\text{tr}}^{\mu} = \{1 : 0.25 : 3\}$
Testing	$S_{\text{te}}^{\alpha} = \{1.05 : 0.12 : 1.89, 1.97\}$, $S_{\text{te}}^{\mu} = \{1.02 : 0.28 : 2.98\}$
Nonlocal diffusion-reaction equation, $\boldsymbol{\mu} = (\alpha, \delta)$	
Training	$S_{\text{tr}}^{\alpha} = \{0.1 : 0.2 : 0.9, 1, 1.5, 1.9\}$, $S_{\text{tr}}^{\delta} = \{0.125 : 0.03125 : 0.5\}$
Testing	$S_{\text{te}}^{\alpha} = \{0.1474 : 0.0474 : 1.8526\}$, $S_{\text{te}}^{\delta} = \{0.1538 : 0.0288 : 0.4712\}$
Nonlocal Burgers' equation, $\boldsymbol{\mu} = (\beta, \nu)$	
Training	$S_{\text{tr}}^{\beta} = \{0.5 : 0.1 : 1\}$, $S_{\text{tr}}^{\nu} = \{0.01 : 0.02 : 0.09\}$,
Testing	$S_{\text{te}}^{\beta} = \{0.524 : 0.024 : 0.98\}$, $S_{\text{te}}^{\nu} = \{0.0133 : 0.0033 : 0.0991\}$
Nonlocal Allen–Cahn equation, $\boldsymbol{\mu} = (\varepsilon, \beta)$	
Training	$S_{\text{tr}}^{\varepsilon} = \{0.05 : 0.01 : 0.1\}$, $S_{\text{tr}}^{\beta} = \{0.1 : 0.1 : 0.5\}$
Testing	$S_{\text{te}}^{\varepsilon} = \{0.0535 : 0.0035 : 0.099\}$, $S_{\text{te}}^{\beta} = \{0.116 : 0.016 : 0.484\}$

Appendix B. Description of training and test data

In the following, we provide details of the training and test data used in each example. The datasets consist of high-fidelity numerical solutions computed using the methods specified in each case. Let $\boldsymbol{\mu} = (\mu_1, \mu_2, \dots, \mu_p)$ represent the parameter set. The training dataset includes solution $\{u(\cdot; \boldsymbol{\mu}_i)\}_{i=1}^{N_{\text{tr}}^{\boldsymbol{\mu}}}$ for $\boldsymbol{\mu}_i \in S_{\text{tr}}^{\mu_1} \times S_{\text{tr}}^{\mu_2} \times \cdots \times S_{\text{tr}}^{\mu_p}$. The test dataset includes $\{u(\cdot; \boldsymbol{\mu}_k)\}_{k=1}^{N_{\text{te}}^{\boldsymbol{\mu}}}$ for $\boldsymbol{\mu}_k \in S_{\text{te}}^{\mu_1} \times S_{\text{te}}^{\mu_2} \times \cdots \times S_{\text{te}}^{\mu_p}$. For time-dependent problems, the training data for each parameter $\boldsymbol{\mu}_i$ include N_{tr}^t time snapshots in S_{tr}^t. More detailed information is provided in Table 4.

Data availability

Data will be made available on request.

References

[1] R. Metzler, J. Klafter, The restaurant at the end of the random walk: Recent developments in the description of anomalous transport by fractional dynamics, J. Phys. A: Math. Gen. 37 (31) (2004) R161.
[2] S.A. Silling, Reformulation of elasticity theory for discontinuities and long-range forces, J. Mech. Phys. Solids 48 (1) (2000) 175–209.
[3] S. Duo, Y. Zhang, Computing the ground and first excited states of the fractional Schrödinger equation in an infinite potential well, Commun. Comput. Phys. 18 (2) (2015) 321–350.
[4] S. Duo, H. Wang, Y. Zhang, A comparative study on nonlocal diffusion operators related to the fractional Laplacian, Discret. Contin. Dyn. Syst. - B 24 (1) (2019) 231–256.
[5] S. Duo, H.W. van Wyk, Y. Zhang, A novel and accurate finite difference method for the fractional Laplacian and the fractional Poisson problem, J. Comput. Phys. 355 (2018) 233–252.
[6] M. D'Elia, Q. Du, C. Glusa, M. Gunzburger, X. Tian, Z. Zhou, Numerical methods for nonlocal and fractional models, Acta Numer. 29 (2020) 1–124.
[7] J. Burkardt, Y. Wu, Y. Zhang, A unified meshfree pseudospectral method for solving both classical and fractional PDEs, SIAM J. Sci. Comput. 43 (2) (2021) A1389–A1411.
[8] Y. Wu, Y. Zhang, Variable-order fractional Laplacian and its accurate and efficient computations with meshfree methods, J. Sci. Comput. 99 (1) (2024) 18.
[9] S. Zhou, Y. Zhang, A novel and simple spectral method for nonlocal PDEs with the fractional Laplacian, Comput. Math. Appl. 168 (2024) 133–147.
[10] S. Duo, L. Ju, Y. Zhang, A fast algorithm for solving the space–time fractional diffusion equation, Comput. Math. Appl. 75 (6) (2018) 1929–1941.
[11] O. Burkovska, C. Glusa, M. D'Elia, An optimization-based approach to parameter learning for fractional type nonlocal models, Comput. Math. Appl. 116 (2022) 229–244.
[12] O. Burkovska, M. Gunzburger, Affine approximation of parametrized kernels and model order reduction for nonlocal and fractional Laplace models, SIAM J. Numer. Anal. 58 (3) (2020) 1469–1494.
[13] G. Pang, L. Lu, G.E. Karniadakis, fPINNs: Fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626.
[14] G. Pang, M. D'Elia, M. Parks, G.E. Karniadakis, nPINNs: Nonlocal physics-informed neural networks for a parametrized nonlocal universal Laplacian operator. Algorithms and applications, J. Comput. Phys. 422 (2020).

[15] L. Guo, H. Wu, X. Yu, T. Zhou, Monte Carlo fPINNs: Deep learning method for forward and inverse problems involving high dimensional fractional partial differential equations, Comput. Methods Appl. Mech. Engrg. 400 (2022) 115523.
[16] H. You, Y. Yu, M. D'Elia, T. Gao, S. Silling, Nonlocal kernel network (NKN): A stable and resolution-independent deep neural network, J. Comput. Phys. 469 (2022) 111536.
[17] M. Zhong, Z. Yan, Data-driven soliton mappings for integrable fractional nonlinear wave equations via deep learning with Fourier neural operator, Chaos Solitons Fractals 165 (2022).
[18] S. Jafarzadeh, S. Silling, N. Liu, Z. Zhang, Y. Yu, Peridynamic neural operators: A data-driven nonlocal constitutive model for complex material responses, Comput. Methods Appl. Mech. Engrg. 425 (2024) 116914.
[19] H. You, Q. Zhang, C.J. Ross, C. Lee, Y. Yu, Learning deep implicit Fourier neural operators (IFNOs) with applications to heterogeneous material modeling, Comput. Methods Appl. Mech. Engrg. 398 (2022) 115296.
[20] H. You, Y. Yu, N. Trask, M. Gulian, M. D'Elia, Data-driven learning of nonlocal physics from high-fidelity synthetic data, Comput. Methods Appl. Mech. Engrg. 374 (2021).
[21] X. Zhou, J. Han, H. Xiao, Learning nonlocal constitutive models with neural networks, Comput. Methods Appl. Mech. Engrg. 384 (2021) 113927.
[22] H. Antil, Y. Chen, A. Narayan, Reduced basis methods for fractional Laplace equations via extension, SIAM J. Sci. Comput. 41 (6) (2019) A3552–A3575.
[23] A. Bonito, D. Guignard, A.R. Zhang, Reduced basis approximations of the solutions to spectral fractional diffusion problems, J. Numer. Math. 28 (3) (2020) 147–160.
[24] G. Rozza, D.B.P. Huynh, A.T. Patera, Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations, Arch. Comput. Methods Eng. 15 (3) (2008) 229–275.
[25] Z. Bai, P.M. Dewilde, R.W. Freund, Reduced-order modeling, Handb. Numer. Anal. 13 (2005) 825–895.
[26] Q. Guan, M. Gunzburger, C.G. Webster, G. Zhang, Reduced basis methods for nonlocal diffusion problems with random input data, Comput. Methods Appl. Mech. Engrg. 317 (2017) 746–770.
[27] D.R. Witman, M. Gunzburger, J. Peterson, Reduced-order modeling for nonlocal diffusion problems, Internat. J. Numer. Methods Fluids 83 (3) (2017) 307–327.
[28] T. Danczul, J. Schöberl, A reduced basis method for fractional diffusion operators ii, J. Numer. Math. 29 (4) (2021) 269–287.
[29] S. Chaturantabut, D.C. Sorensen, Nonlinear model reduction via discrete empirical interpolation, SIAM J. Sci. Comput. 32 (5) (2010) 2737–2764.
[30] Y. Li, L. Zikatanov, C. Zuo, A reduced conjugate gradient basis method for fractional diffusion, SIAM J. Sci. Comput. (2024) S68–S87.
[31] M. Ainsworth, C. Glusa, Aspects of an adaptive finite element method for the fractional Laplacian: A priori and a posteriori error estimates, efficient implementation and multigrid solver, Comput. Methods Appl. Mech. Engrg. 327 (2017) 4–35.
[32] S.O. Sahin, S.S. Kozat, Nonuniformly sampled data processing using LSTM networks, IEEE Trans. Neural Netw. Learn. Syst. 30 (5) (2018) 1452–1461.
[33] N.T. Mücke, S.M. Bohté, C.W. Oosterlee, Reduced order modeling for parameterized time-dependent PDEs using spatially and memory aware deep learning, J. Comput. Sci. 53 (2021) 101408.
[34] M. Dwarampudi, N.V. Reddy, Effects of padding on LSTMs and CNNs, 2019, arXiv preprint.
[35] S. Yoon, H. Yu, A simple distortion-free method to handle variable length sequences for recurrent neural networks in text dependent speaker verification, Appl. Sci. 10 (12) (2020) 4092.
[36] Y. Wang, S. Zhou, Y. Zhang, Parametric model reduction with convolutional neural networks, Int. J. Numer. Anal. Model. 21 (5) (2024) 716–738.
[37] F.J. Gonzalez, M. Balajewicz, Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, 2018, arXiv preprint.
[38] B. Lusch, J.N. Kutz, S.L. Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, Nat. Commun. 9 (1) (2018) 4950.
[39] S. Fresca, A. Manzoni, POD-DL-ROM: Enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition, Comput. Methods Appl. Mech. Engrg. 388 (2022) 114181.
[40] F. Pichi, B. Moya, J.S. Hesthaven, A graph convolutional autoencoder approach to model order reduction for parametrized PDEs, J. Comput. Phys. 501 (2024) 112762.
[41] S. Duo, Y. Zhang, Accurate numerical methods for two and three dimensional integral fractional Laplacian with applications, Comput. Methods Appl. Mech. Engrg. 355 (2019) 639–662.
[42] M. D'Elia, M. Gunzburger, Identification of the diffusion parameter in nonlocal steady diffusion problems, Appl. Math. Optim. 73 (2) (2016) 227–249.
[43] S.G. Samko, A.A. Kilbas, O.I. Marichev, Fractional Integrals and Derivatives: Theory and Applications, Gordon and Breach Science Publishers, 1993.
[44] S. Duo, Y. Zhang, Numerical approximations for the tempered fractional Laplacian: Error analysis and applications, J. Sci. Comput. 81 (1) (2019) 569–593.
[45] H. Fu, H. Wang, Z. Wang, POD/DEIM reduced-order modeling of time-fractional partial differential equations with applications in parameter identification, J. Sci. Comput. 74 (2018) 220–243.
[46] L. Chen, K. Painter, C. Surulescu, A. Zhigun, Mathematical models for cell migration: A non-local perspective, Philos. Trans. R. Soc. B: Biol. Sci. 375 (1807) (2020) 20190379.
[47] D.G. Kendall, Mathematical models of the spread of infection, Math. Comput. Sci. Biol. Med. (1965) 213–225.
[48] N.I. Kavallaris, E. Latos, T. Suzuki, Diffusion-driven blow-up for a nonlocal Fisher–KPP type model, SIAM J. Math. Anal. 55 (3) (2023) 2411–2433.
[49] N. Sugimoto, Burgers equation with a fractional derivative; hereditary effects on nonlinear acoustic waves, J. Fluid Mech. 225 (1991) 631–653.
[50] T. Kakutani, K. Matsuuchi, Effect of viscosity on long gravity waves, J. Phys. Soc. Japan 39 (1) (1975) 237–246.
[51] M.J. Miksis, L. Ting, Effective equations for multiphase flows—waves in a bubbly liquid, in: J.W. Hutchinson, T.Y. Wu (Eds.), Advances in Applied Mechanics, Vol. 28, Elsevier, 1991, pp. 141–260.
[52] Y. Lin, C. Xu, Finite difference/spectral approximations for the time-fractional diffusion equation, J. Comput. Phys. 225 (2) (2007) 1533–1552.
[53] P.W. Bates, S. Brown, J. Han, Numerical analysis for a nonlocal Allen-Cahn equation, Int. J. Numer. Anal. Model. 6 (1) (2009) 33–49.
[54] C.E. Heaney, Y. Li, O.K. Matar, C.C. Pain, Applying convolutional neural networks to data on unstructured meshes with space-filling curves, Neural Netw. 175 (C) (2024).
[55] J. Mack, R. Arcucci, M. Molina-Solana, Y.K. Guo, Attention-based convolutional autoencoders for 3D-variational data assimilation, Comput. Methods Appl. Mech. Engrg. 372 (2020) 113291.
[56] H. Gao, L. Sun, J. Wang, PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain, J. Comput. Phys. 428 (2021) 110079.
[57] S. Fresca, L. Dedè, A. Manzoni, A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs, Math. Comp. 87 (2021) 1–36.
[58] S. Hanifi, A. Cammarono, H. Zare-Behtash, Advanced hyperparameter optimization of deep learning models for wind power prediction, Renew. Energy 221 (2024) 119700.