INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 21, Number 5, Pages 716–738

doi: 10.4208/ijnam2024-1029

PARAMETRIC MODEL REDUCTION WITH CONVOLUTIONAL NEURAL NETWORKS

YUMENG WANG, SHIPING ZHOU, AND YANZHI ZHANG*

Abstract. Reduced order modeling (ROM) has been widely used to solve parametric PDEs. However, most existing ROM methods rely on linear projections, which face efficiency challenges when dealing with complex nonlinear problems. In this paper, we propose a convolutional neural network-based ROM method to solve parametric PDEs. Our approach consists of two components: a convolutional autoencoder (CAE) that learns a low-dimensional representation of the solutions, and a convolutional neural network (CNN) that maps the model parameters to the latent representation. For time-dependent problems, we incorporate time t into the surrogate model by treating it as an additional parameter. To reduce computational costs, we use a decoupled training strategy to train the CAE and latent CNN separately. The advantages of our method are that it does not require training data to be sampled at uniform time steps and can predict the solution at any time t within the time domain. Extensive numerical experiments have shown that our surrogate model can accurately predict solutions for both time-independent and time-dependent problems. Comparison with traditional numerical methods further demonstrates the computational effectiveness of our surrogate solver, especially for solving nonlinear parametric PDEs.

Key words. Parametric PDEs, reduced order modeling, convolutional autoencoder, convolutional neural network, decoupled training strategy.

1. Introduction

Parametric partial differential equations (PDEs) arise in various contexts, including control and design optimization [35], risk assessment [12], uncertainty quantification [5], and data assimilation [1]. The parameters describe physical and geometric constraints of PDEs and can manifest in various ways, such as model coefficients, initial conditions, boundary conditions, and even domain geometry. Solving parametric PDEs for every point in the parameter space of interest could be extremely costly and impractical, particularly in high-dimensional cases. For instance, in real-time applications with severely limited computation time, solving the PDE for even a single set of parameters can be prohibitively costly. To reduce computational costs, a common strategy is to employ reduced-order modeling (ROM) [19, 22, 29]. In this work, we propose a convolutional neural network (CNN) based ROM to solve parametric PDEs.

Let $\Omega \subset \mathbb{R}^d$ (for $d \geq 1$) denote an open bounded domain. We consider a general parametric PDE of the form:

$$\partial_t u(\mathbf{x}, t; \boldsymbol{\mu}) = \mathcal{L}u(\mathbf{x}, t; \boldsymbol{\mu}) + \mathcal{N}(\mathbf{x}, t, u; \boldsymbol{\mu}), \qquad \text{for } \mathbf{x} \in \Omega, \tag{1}$$

$$\mathcal{B}u(\mathbf{x}, t; \boldsymbol{\mu}) = g(\mathbf{x}, t), \qquad \text{for } \mathbf{x} \in \partial\Omega, \quad t \in (0, T], \tag{2}$$

where the solution u depends on space $\mathbf{x}$, time t, and parameters $\boldsymbol{\mu}$. Here, $\mathcal{L}$ denotes a linear differential operator, and $\mathcal{N}$ includes nonlinear terms of u, and $\mathcal{B}$ represents a linear boundary operator. The parameter $\boldsymbol{\mu} \in \Gamma$ could be, for example, the diffusion rate, Reynolds number, or boundary controller. We assume

Received by the editors on June 6, 2024 and, accepted on August 10, 2024.
2000 *Mathematics Subject Classification.* 65K10, 68T07, 93A15.
*Corresponding author.

that the parameter space $\Gamma \subset \mathbb{R}^p$ is compact, and the parametric map $\boldsymbol{\mu} \to u(\boldsymbol{\mu})$ is continuous [9, 14]. The general form in (1) covers both time-dependent and time-independent parametric PDEs. If a time-dependent problem is considered, the initial condition could also depend on parameters, i.e.

$$u(\mathbf{x}, 0; \boldsymbol{\mu}) = \phi(\mathbf{x}; \boldsymbol{\mu}), \qquad \text{for } \mathbf{x} \in \bar{\Omega}. \tag{3}$$

In this work, we mainly focus on problems where the parameters appear in the PDEs, initial conditions, and/or boundary conditions. We are interested in solving the PDE for a range or ensemble of parameters. Computing the PDE solution can be time-consuming, especially for nonlinear and time-dependent problems. Hence, using full order models (FOMs) to approximate the parametric map is impractical due to the constraints of computational time and resources. Consequently, ROMs have become popular for solving parameterized PDEs; see [2, 3, 6, 8, 22, 29] and references therein.

The main idea of ROMs is to find a low-dimensional representation of the original problem, such that it can be efficiently solved. In the literature, most reduced order modeling methods are based on linear projection (e.g., proper orthogonal decomposition (POD)-Galerkin method) [2, 3, 6–8, 11]. These methods are proved to be effective in solving problems that can be well approximated by a low-rank approximation. However, they encounter challenges in solving complex nonlinear problems, such as Kolmogorov n-width problems. In these cases, the projection-based methods become ineffective; see more discussion in [14, 23]. Various approaches have been proposed to address these challenges, such as combining traditional reduced basis methods with Bayesian nonlinear regression approach [32], and using a locally weighted proper orthogonal decomposition method [27]. Although these techniques enhance the performance of the original ROM methods to some extent, they also introduce substantial computational costs and complexity.

Recently, there has been an emerging trend of nonlinear, data-driven ROM approaches using neural networks [16, 18, 20, 23–25]. Neural network-based ROMs have been applied to study problems, such as cardiac electrophysiology [16], fluid dynamics [15], and water waves [23]. Instead of linear projection, these approaches compress high-dimensional data into a lower-dimensional latent space by using autoencoders. There are various types of autoencoders, including feedforward autoencoders [4, 10], recurrent autoencoders [13], graph autoencoders [28], and the widely used convolutional autoencoders [20, 23, 25]. The choice of autoencoder neural network structure typically depends on the specific tasks. The encoded representation in the latent space could be viewed as an approximation to the full-order models. For time-dependent problems, Long Short-Term Memory (LSTM) networks are typically used to learn the dynamics in the latent space [18, 23, 25]. It usually requires the input time series data sampled at uniform time steps.

In this work, we propose a CNN-based surrogate model for solving parametric PDEs. Our model consists of two components: a CAE and a latent-CNN; see the illustration in Figure 1. In the offline (training) phase, the encoder and decoder are trained to learn the low-dimensional representation in the latent space, while the latent-CNN maps parameters $\boldsymbol{\mu}$ and time t to the encoded solution in this latent space. We use a decoupled training strategy – the CAE and latent-CNN are trained separately – to enhance training efficiency. Moreover, we use a CNN structure in the latent space. Unlike LSTM, the CNN does not require uniform time steps in the training data, making it more flexible for handling multiscale time problems. It can also predict the solution at any time within the studied time frame. In the

online stage, the combination of the latent-CNN and the decoder yields our surrogate solver for parametric PDEs. Extensive numerical experiments are provided to demonstrate the effectiveness of our method in solving time-independent and time-dependent parametric PDEs. Furthermore, we compare the CAE with principal component analysis (PCA) to show its superior compression and representation capabilities. Comparisons with traditional numerical methods further suggest that our surrogate model is highly efficient for solving parametric PDEs, significantly reducing both computational and storage costs.

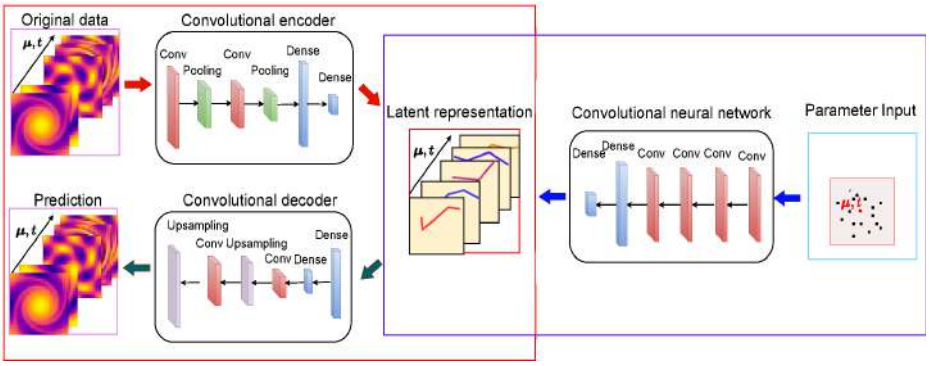


FIGURE 1. Illustration of our surrogate model, including CAE (in the red frame) and latent-CNN (in the blue frame).

The paper is organized as follows. In Section 2, we introduce our surrogate model for solving parametric PDEs. The performance of our method is examined in Section 3 through various numerical examples, including both time-dependent and time-independent problems. Finally, we summarize our work in Section 4.

2. CNN surrogate model

In this section, we introduce our CNN-based surrogate model for solving parametric PDEs. In Section 2.1, we will introduce the model reduction with CAE, followed by a CNN latent-space modeling in Section 2.2. The detailed offline training and online predicting algorithms of our method are presented in Section 2.3.

Assume that high-fidelity solution data of parametric PDEs are available for a set of representative parameters $\boldsymbol{\mu}$. If time-dependent problems are considered, solution data at multiple time points are also available. Let N, N_t, and $N_{\boldsymbol{\mu}}$ denote the number of spatial grid points, time points, and parameter points, respectively. Here, the data are acquired on a uniform spatial mesh, but not necessarily at uniform time steps. Denote the high-fidelity solution data as

$$\left\{\mathbf{u}_s^m \in \mathbb{R}^N : \ 0 \le m \le N_t,\, 1 \le s \le N_{\boldsymbol{\mu}}\right\},$$

where $\mathbf{u}_s^m$ represents snapshot data at the time $t = t_m$ for the parametric PDEs with parameters $\boldsymbol{\mu} = \boldsymbol{\mu}_s$. For time-independent problems, the data can be represented as

$$\left\{\mathbf{u}_s \in \mathbb{R}^N : \ 1 \le s \le N_{\boldsymbol{\mu}}\right\}.$$

Traditional model reductions, such as POD, assume that the high-fidelity solution $\mathbf{u}_s^m$ can be approximated by a linear combination of a set of reduced bases. However, these methods face efficiency challenges when applied to complex nonlinear problems (see more discussion in [14, 23]).

2.1. Model reduction. In the following, we introduce our CAE-based model reduction method. The autoencoder, first introduced in [31], is a neural network to learn an effective low-rank representation of high-dimensional data. It usually consists of two main parts: an encoder and a decoder. The encoder, denoted as $\Psi : \mathbb{R}^N \to \mathbb{R}^n$ with $n \ll N$, maps the input $\mathbf{u} \in \mathbb{R}^N$ into a low-dimensional representation $\mathbf{v} \in \mathbb{R}^n$. Here, $\mathbf{v}$ is also known as the "latent space representation" of $\mathbf{u}$. While the decoder, denoted as $\Phi : \mathbb{R}^n \to \mathbb{R}^N$, reconstructs the high-dimensional data $\widehat{\mathbf{u}} \in \mathbb{R}^N$ from the low-dimensional representation $\mathbf{v} \in \mathbb{R}^n$. That is, with input $\mathbf{u}$, the encoder and the decoder can be expressed as

$$\mathbf{v} = \Psi(\mathbf{u}; \vartheta_e), \tag{4}$$

$$\widehat{\mathbf{u}} = \Phi(\mathbf{v}; \vartheta_d), \tag{5}$$

where ϑ_e and ϑ_d represent the neural network parameters of the encoder and decoder, respectively. Throughout this work, we denote $\widehat{\mathbf{w}}$ as the neural network reconstruction or approximation of the data $\mathbf{w}$.

We use a convolutional encoder and decoder. The encoder consists of convolutional layers, pooling layers, and fully-connected layers. The input of the encoder is the high-dimensional data with dimension N, and the output is the low-dimensional representation with dimension n. The convolutional layers use zero-padding. To reduce the data dimension, each convolutional layer is followed by a pooling layer. It downsamples the feature map from the convolutional layer, reducing dimensionality while retaining essential information. In our study, we use max pooling, which retains the maximum value from each pooling window as the output. The fully connected layers transform the data from the last pooling layer into a low-dimensional representation of the desired dimension n. Rectified Linear Unit (ReLU) activation functions are applied to all convolutional and fully-connected layers, except for the last fully-connected layer, which has no activation function.

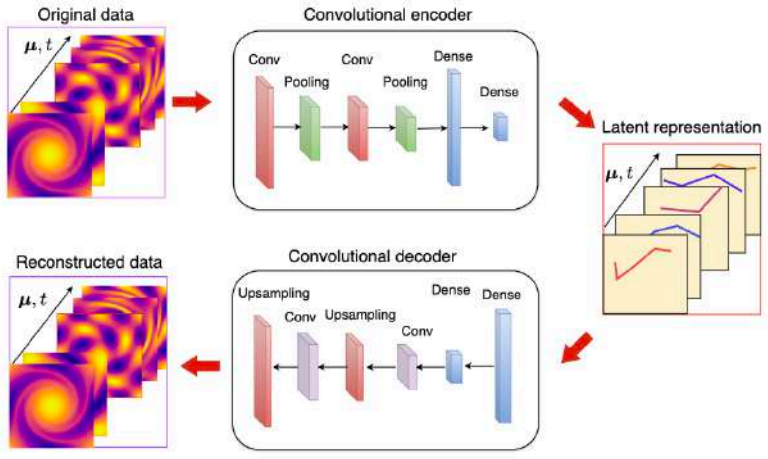


FIGURE 2. Illustration of CAE for model reduction.

The decoder consists of convolutional layers, upsampling layers, and fully connected layers. In the decoder, the latent representation with dimension n serves as the input, and the reconstructed data with dimension N is the output. Instead of pooling layers, upsampling layers are used to increase dimensionality. Here, we use the nearest neighbor interpolation method for upsampling [30]. The fully connected

layers and convolutional layers operate similarly to those in the encoder. Note that the pooling and upsampling layers do not have trainable parameters.

Figure 2 illustrates the structure of our CAE. The original data $\mathbf{u} \in \mathbb{R}^N$, representing the high-fidelity solutions for multiple representative parameters at different time steps, serve as the input to the convolutional encoder. The latent space representation, with a lower dimension, is the output of the encoder. In the decoder process, the low-dimensional representations serve as the input, while the reconstructed data $\widehat{\mathbf{u}} \in \mathbb{R}^N$ are the output. The encoder and decoder are trained together by minimizing the difference between the original data $\mathbf{u}$ and its neural network reconstruction $\widehat{\mathbf{u}}$. To this end, we define the loss function of CAE as:

$$\begin{aligned} \mathcal{L}_{\mathrm{CAE}}(\vartheta_e, \vartheta_d) &= \frac{1}{N_t^{\mathrm{train}} N_{\boldsymbol{\mu}}^{\mathrm{train}}} \sum_{s=1}^{N_{\boldsymbol{\mu}}^{\mathrm{train}}} \sum_{m=1}^{N_t^{\mathrm{train}}} \|\mathbf{u}_s^m - \widehat{\mathbf{u}}_s^m\|_{\mathrm{rms}}^2 \\ &= \frac{1}{N_t^{\mathrm{train}} N_{\boldsymbol{\mu}}^{\mathrm{train}}} \sum_{s=1}^{N_{\boldsymbol{\mu}}^{\mathrm{train}}} \sum_{m=1}^{N_t^{\mathrm{train}}} \big\|\mathbf{u}_s^m - \Phi\big(\Psi(\mathbf{u}_s^m; \vartheta_e); \vartheta_d\big)\big\|_{\mathrm{rms}}^2, \end{aligned} \tag{6}$$

where $\|\cdot\|_{\mathrm{rms}}$ is the root mean squared (rms) norm, defined as

$$\|\mathbf{w}\|_{\mathrm{rms}} = \Big(\frac{1}{N}\sum_{i=1}^{N} |w_i|^2\Big)^{1/2}, \qquad \text{for } \mathbf{w} \in \mathbb{R}^N.$$

Then the encoder and decoder are trained by

$$\min_{\vartheta_e, \vartheta_d} \mathcal{L}_{\mathrm{CAE}}(\vartheta_e, \vartheta_d),$$

and the parameters ϑ_e and ϑ_d are updated and learned simultaneously.

2.2. Latent space modeling. In this section, we introduce the latent-space modeling of our surrogate model. Here, we propose to use a CNN to map the parameters to the encoded solutions in the latent space. If a time-dependent problem is considered, we treat time t as an additional parameter. Denote the proposed CNN as $\Theta : \mathbb{R}^p \times \mathbb{R} \to \mathbb{R}^n$, i.e.,

$$\widehat{\mathbf{v}} = \Theta(\boldsymbol{\mu}, t; \vartheta_l), \tag{7}$$

where ϑ_l denotes the parameters of the CNN. The parameters $\boldsymbol{\mu} \in \mathbb{R}^p$ and time $t \in \mathbb{R}$ are the input, and $\widehat{\mathbf{v}} \in \mathbb{R}^n$ is the corresponding output that approximates $\mathbf{v}$.

For time-dependent problems, the time evolution is often addressed using LSTM networks [18, 23, 25]. However, LSTM usually requires sequential time-series data sampled at uniform time steps, which can be challenging to obtain in practical applications. Moreover, LSTM iteratively predicts solution dynamics – starting from the initial condition and making predictions step by step until the desired time is reached. To avoid these issues, we propose a CNN to learn the mapping from the parameters to the latent space solutions directly. Specifically, we treat the model parameters $\boldsymbol{\mu}$ and time t as inputs to the CNN, and thus set the input layer channel size to $p + 1$. The main advantage of our approach is that it eliminates the requirements for training data to be sampled at uniform time steps. Once trained, the CNN can predict the solution for a given parameters $\boldsymbol{\mu}$ and time t with a single evaluation, without requiring any iterations. Furthermore, compared to fully-connected neural networks, the filter-sharing feature of CNNs could help avoid overfitting.

Figure 3 illustrates the proposed CNN for mapping the model parameters $\boldsymbol{\mu}$ and time t to the low-dimensional latent space representation. Here the inputs include

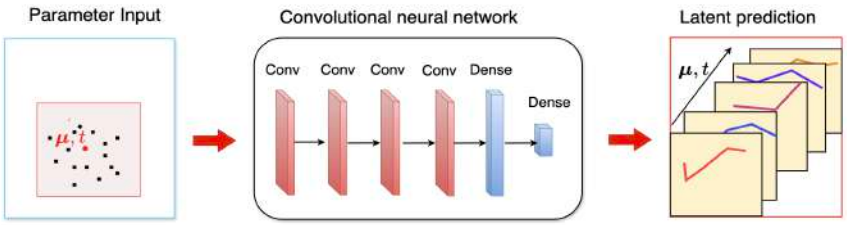


FIGURE 3. Illustration of the CNN for latent space modeling.

the model parameters $\boldsymbol{\mu}$ and time t. To learn the mapping, we minimize the loss function in the latent space, i.e.,

$$\min_{\vartheta_l} \mathcal{L}_{\text{Mapping}}(\vartheta_l),$$

where the loss function is defined as

$$\begin{aligned} \mathcal{L}_{\text{Mapping}}(\vartheta_l) &= \frac{1}{N_t^{\text{train}} N_{\boldsymbol{\mu}}^{\text{train}}} \sum_{s=1}^{N_{\boldsymbol{\mu}}^{\text{train}}} \sum_{m=1}^{N_t^{\text{train}}} \|\mathbf{v}_s^m - \widehat{\mathbf{v}}_s^m\|_{\text{rms}}^2 \\ &= \frac{1}{N_t^{\text{train}} N_{\boldsymbol{\mu}}^{\text{train}}} \sum_{s=1}^{N_{\boldsymbol{\mu}}^{\text{train}}} \sum_{m=1}^{N_t^{\text{train}}} \|\mathbf{v}_s^m - \Theta(\boldsymbol{\mu}_s, t_m; \vartheta_l)\|_{\text{rms}}^2. \end{aligned} \tag{8}$$

2.3. Surrogate model for prediction. Our data-driven surrogate model utilizes a CAE for dimensionality reduction and a CNN for latent space modeling. It consists of three neural networks:

$$\Psi : \mathbb{R}^N \to \mathbb{R}^n, \quad \Theta : \mathbb{R}^p \times \mathbb{R} \to \mathbb{R}^n, \quad \Phi : \mathbb{R}^n \to \mathbb{R}^N.$$

The encoder Ψ compresses the high-fidelity solution data with dimension N to the latent space representation with dimension n. The CNN Θ maps the parameters and time to the encoded solution and learns the relation between the parameters and the latent space representation. The decoder Φ reconstructs the high-dimensional data from the latent space representation. In the following, we summarize our surrogate model into two stages: the offline stage and the online stage.

In the offline (training) stage, we adopt a decoupled training strategy. Specifically, we first train the CAE (i.e., Ψ and Φ) with the high-fidelity solution data to obtain the low-dimensional representation. Subsequently, we train the CNN (i.e., Θ) to learn the mapping between the parameters (including time t) and the latent space representation. The detailed steps are summarized in Algorithm 1. Here, we denote ϑ_e^* and ϑ_d^* as the optimal neural network parameters that minimize the loss function $\mathcal{L}_{\text{CAE}}$, while ϑ_l^* represents the optimal parameters minimizing $\mathcal{L}_{\text{Mapping}}$. In the literature [18, 21], the joint training approach is used where dimensionality reduction and the latent-space modeling are trained simultaneously. To that end, a weighted loss function is introduced by combining (6) and (8) with proper weights [18, 21]. Compared to the joint training approach, our decoupled strategy offers more flexibility and enhances training efficiency. First, it has fewer trainable parameters for each step. Secondly, there is no need to balance the weights of the two loss functions, as required in the joint training approach.

In the online (predicting) stage, our surrogate solver for parametric PDEs consists of the trained CNN $\Theta(\cdot, \cdot; \vartheta_l^*)$ and decoder $\Phi(\cdot; \vartheta_d^*)$; see the illustration in Figure 4 and the detailed algorithms in Algorithm 2. Given new parameters as

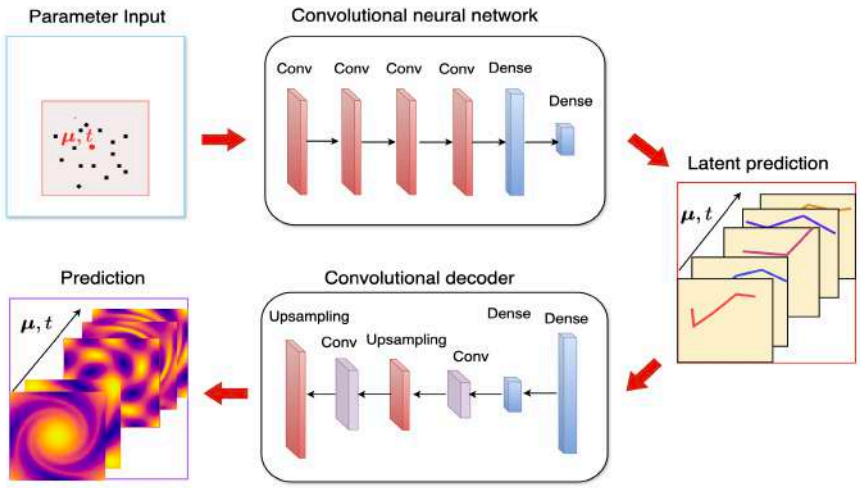


FIGURE 4. Illustration of the online stage of our model.

input ($\boldsymbol{\mu}$ and time t if applicable), the CNN predicts the corresponding reduced solution in the latent space and then decoder outputs the predicted solution in the original high-dimensional space. The process can be expressed as

$$(\boldsymbol{\mu}, t) \xrightarrow{\Theta:\mathbb{R}^p\times\mathbb{R}\to\mathbb{R}^n} \widehat{\mathbf{v}} \xrightarrow{\Phi:\mathbb{R}^n\to\mathbb{R}^N} \widehat{\mathbf{u}}.$$

In contrast to the LSTM-based model (e.g. in [18, 23, 25]), our surrogate solver requires only the inputs of the parameters $\boldsymbol{\mu}$ and time t to predict the solution. In contrast, LSTM-based method starts from an encoded initial condition and iteratively evaluate the solution step by step until the prediction time t.

Algorithm 1 Offline stage

Step 1: CAE training
Input: High-fidelity solution data $\{\mathbf{u}_s^m\}$

1: Initialize the autoencoder randomly;
2: Optimize the loss function $\mathcal{L}_{\text{CAE}}(\vartheta_e, \vartheta_d)$ in (6) with input $\{\mathbf{u}_s^m\}$;
3: Return the trained encoder $\Psi(\cdot; \vartheta_e^*)$, decoder $\Phi(\cdot; \vartheta_d^*)$;
Output the latent space representation $\{\mathbf{v}_s^m\}$ of $\{\mathbf{u}_s^m\}$.

Step 2: CNN training
Input: Parameters $(\boldsymbol{\mu}, t)$ and latent space representation $\{\mathbf{v}_s^m\}$ from Step 1

4: Initialize the CNN randomly;
5: Optimize the loss function $\mathcal{L}_{\text{Mapping}}(\vartheta_l)$ in (8) with inputs $(\boldsymbol{\mu}, t)$ and $\{\mathbf{v}_s^m\}$;
6: Return the trained CNN model $\Theta(\cdot, \cdot; \vartheta_l^*)$.

Our method is designed to handle both time-independent and time-dependent parametric PDEs. For time-independent problems, the proposed CNN can effectively learn the mapping between the model parameters and the latent space representation. For time-dependent problems, we further treat time t as an additional parameter. Thus, our surrogate model provides a time-continuous solver, which needs only a single evaluation of the neural network to predict the solution at any

Algorithm 2 Online stage

Input: Model parameters $\boldsymbol{\mu}$ and time t

Output: Predicted solution $\widehat{\mathbf{u}} \in \mathbb{R}^N$ at time t for parameters $\boldsymbol{\mu}$

1: Feed $(\boldsymbol{\mu}, t)$ into the trained CNN and output the latent space prediction $\widehat{\mathbf{v}} = \Theta\left(\boldsymbol{\mu}, t; \vartheta_l^*\right)$;
2: Feed $\widehat{\mathbf{v}}$ into the trained decoder and output the predicted solution $\widehat{\mathbf{u}} = \Phi\left(\widehat{\mathbf{v}}; \vartheta_d^*\right)$.

time within the studied time frame. Specific details of the CAE and latent-space CNN configurations for our test cases will be provided in Section 3.

3. Numerical experiments

In this section, we conduct numerical experiments to assess the performance of our surrogate solver for parametric PDEs. Furthermore, we will compare the compression capability of CAE with PCA, especially in handling nonlinear problems. In Sections 3.1–3.4, we test the performance of our method in solving both time-independent and time-dependent PDEs. A brief comparison of our surrogate solver and traditional numerical methods is provided in Section 3.5.

Denote $\mathcal{S}_{\text{train}}$ and $\mathcal{S}_{\text{test}}$ as the sets of parameters for training and testing data, respectively, and they satisfy $\mathcal{S}_{\text{train}} \cap \mathcal{S}_{\text{test}} = \varnothing$. In our study, both training and testing data are generated by numerically solving parametric PDEs. Note that our method is a purely data-driven approach, and its training does not require PDE information. We adopt the adaptive moment estimation (Adam) algorithm to train both the CAE and the latent-CNN. In the Adam algorithm, the two hyperparameters that control the decay rates of the moment estimates are set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$, respectively. Additionally, we use an adaptive learning rate defined as $\eta_{n+1} = \eta_0/(1 + n\delta_\eta)$, where n denotes the number of iterations, with $\eta_0 = 0.0003$ and $\delta_\eta = 10^{-5}$. In both Step 1 and Step 2, we set the number of training epochs to 5000 and the batch size to 32. During the training process, we adopt an early stopping strategy to prevent overfitting.

3.1. Parametric Poisson equations. Consider a two-dimensional (2D) parametric Poisson problem on the domain $\Omega = (-1, 1)^2$:

$$\begin{aligned} \Delta u(\mathbf{x}; \boldsymbol{\mu}) &= f(\mathbf{x}; \boldsymbol{\mu}), \qquad \text{for } \mathbf{x} \in \Omega, \\ u(\mathbf{x}; \boldsymbol{\mu}) &= 0, \qquad\quad\ \ \text{for } \mathbf{x} \in \partial\Omega, \end{aligned} \tag{9}$$

where the forcing term f depends on parameters $\boldsymbol{\mu} = (\mu_1, \mu_2)$, given by

$$f(\mathbf{x}; \boldsymbol{\mu}) = -\mathrm{e}^{-2[(x-\mu_1)^2 + (y-\mu_2)^2]}$$

with $\mu_1, \mu_2 \in [-1, 1]$. The training and testing data are generated by solving the Poisson problem (9) with the central difference method. Specifically, we use a uniform mesh with $N = 256^2$ total grid points, i.e., 256 grid points along each of x- and y-directions. The training data include numerical solutions for 121 pairs of parameters μ_1 and μ_2, with each sampled uniformly with 11 values in the range $[-1, 1]$. The testing data include numerical solutions for 417 parameter sets (μ_1, μ_2), which are randomly sampled from $(-1, 1)^2$.

The detailed information of the three neural networks in our surrogate solver – an encoder, a decoder, and a latent-space CNN – is listed below. The convolutional filter size in the CAE is 3×3, while the filter size in the latent space is 2.

- **Encoder:** The encoder consists of six 2D convolutional layers, with 4, 8, 16, 32, 64, and 128 filters used in each layer, respectively. Each convolutional layer is followed by a 2D max-pooling layer with a pooling size of 2×2. The output of the last convolutional and pooling layers is then flattened and fed into the fully-connected layers. There are three fully-connected layers with 50, 25, and 4 neurons, respectively.
- **Decoder:** In order to reconstruct the data from the latent space representation to the original high dimensions, a reverse structure is used as in the encoder. An additional fully-connected layer with 16 neurons is added before the first convolutional layer. Instead of pooling layers, the 2D upsampling layers are used with nearest neighbor interpolation method [30].
- **CNN mapping:** We treat the two parameters (μ_1, μ_2) as input. The CNN includes five 1D convolutional layers, each with 128 filters. They are followed by two fully connected layers with 128 and 4 neurons, respectively. ReLU activation functions are applied to all convolutional and fully connected layers, except for the last fully connected layer, which has no activation function.

Figure 5 compares the predicted solution with the ground truth solution for different parameters in $\mathcal{S}_{\text{test}}$, where the latent dimension $n = 4$. Each column

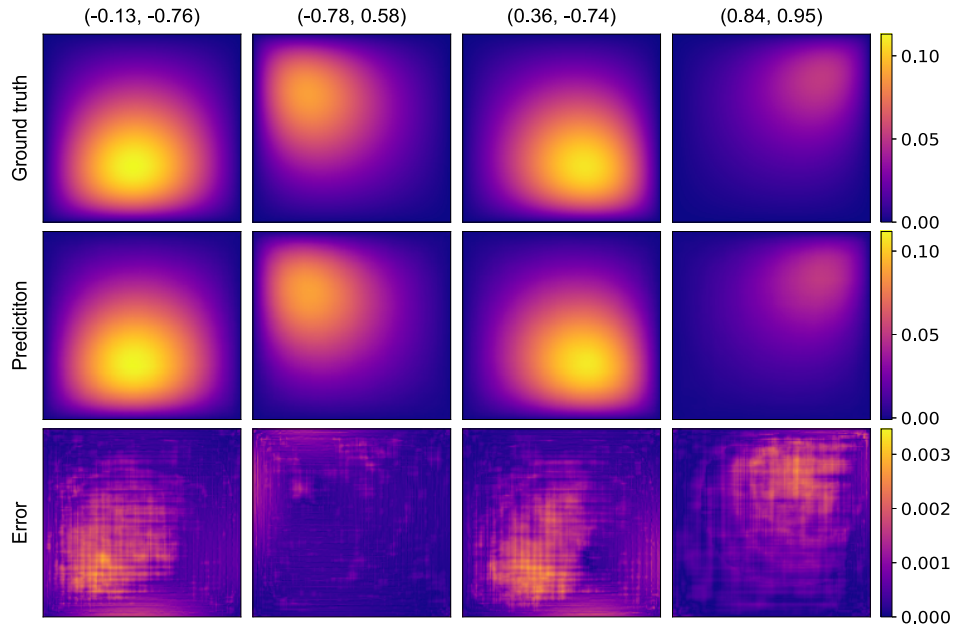
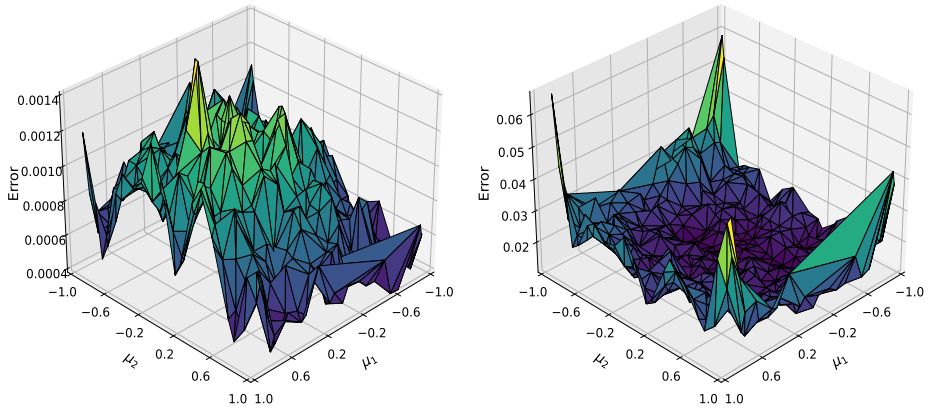


FIGURE 5. Comparison of the ground truth solution ($\mathbf{u}$) and predicted solution ($\widehat{\mathbf{u}}$) of the Poisson problem for different parameters (μ_1, μ_2), where the error is computed as $|\mathbf{u} - \widehat{\mathbf{u}}|$. For easy comparison, the results in each row use the same colorbar scale.

represents the results for one parameter set (μ_1, μ_2). The first and second rows are the ground truth solution $\mathbf{u}$ and the predicted solution $\widehat{\mathbf{u}}$, respectively. The third row shows their difference $|\mathbf{u} - \widehat{\mathbf{u}}|$. Figure 5 and our extensive studies show that our surrogate solver provides accurate solutions for parameters that are not included in the training data. The difference between predicted and ground truth

solutions is small, around 10^{-3}. The solution profile of the parametric Poisson problem depends on the distribution of the source term, which is determined by the parameters $\boldsymbol{\mu} = (\mu_1, \mu_2)$. Figure 6 further presents the absolute and relative errors of our surrogate solver in predicting the solution for 417 test parameter sets. It shows that the errors are small for all testing data. The results in Figures 5 and 6 suggest that our method is effective in solving the parametric Poisson problems.



(a) (b)

FIGURE 6. (a) Absolute errors $\|\mathbf{u} - \widehat{\mathbf{u}}\|_{\text{rms}}$ and (b) relative errors $\|\mathbf{u} - \widehat{\mathbf{u}}\|_{\text{rms}}/\|\mathbf{u}\|_{\text{rms}}$ in the predicted solutions of the parametric Poisson problem for different parameters (μ_1, μ_2).

Moreover, the results in Figures 5–6 indicate that CAE learns an effective latent space representation of the given high-dimensional data, even with a small latent dimension $n = 4$. Here, the data dimension reduces from $N = 65536$ to $n = 4$, and the compression ratio exceeds 99.97%. To further study its compression capability, we compare the CAE with PCA, one of the most popular projection-based dimensionality reduction methods. Here, we focus only on their compression capability, so latent space modeling is not involved. In other words, the solution $\widehat{\mathbf{u}}$ is obtained as $\widehat{\mathbf{u}} = \Phi(\Psi(\mathbf{u}; \vartheta_e^*); \vartheta_d^*)$. Figure 7 compares the average reconstruction errors of CAE and PCA across all 417 test data. The results show that the reconstruction error of PCA increases monotonically as the latent dimension (the number of principal components) decreases. While the reconstruction errors of CAE remain low for different latent dimensions, and they are much smaller than those of PCA when latent dimension $n \leq 8$. Moreover, CAE has a narrow range of maximum and minimum errors, indicating greater stability across all parameters. Compared to PCA, the CAE can achieve much smaller reconstruction errors for small latent space dimensions. In contrast, PCA requires larger latent dimensions to ensure smaller reconstruction errors, which leads to higher computational costs. More discussion about CAE compression capability can be found in [14, 15, 18, 25, 26].

3.2. Viscous Burgers equation. Consider the viscous Burgers equation of the form:

$$\begin{aligned} \frac{\partial u(x,t;\mu,\nu)}{\partial t} + \mu u \frac{\partial u}{\partial x} &= \nu \frac{\partial^2 u}{\partial x^2}, && \text{for} \quad x \in \Omega, \qquad t \in (0,T], \\ u(x,t;\mu,\nu) &= 0, && \text{for} \quad x \in \partial\Omega, \end{aligned} \tag{10}$$

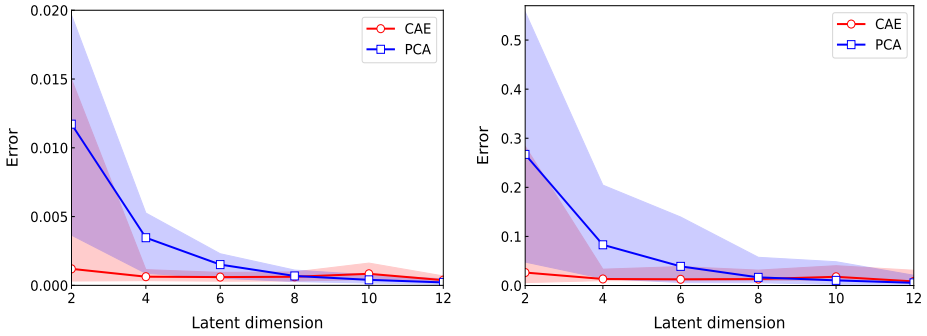


FIGURE 7. (a) Absolute errors $\|\mathbf{u} - \widehat{\mathbf{u}}\|_{\text{rms}}$ and (b) relative errors $\|\mathbf{u} - \widehat{\mathbf{u}}\|_{\text{rms}}/\|\mathbf{u}\|_{\text{rms}}$ of CAE and PCA for the parametric Poisson problems, with no latent space modeling involved. The solid line represents the average errors across 417 test data, while the shaded area indicates the range of maximum and minimum errors.

where parameters $\mu \in [0.1, 1]$ and $\nu \in [0.01, 0.2]$. The initial condition is given by

$$u(x, 0) = -\sin(\pi x), \quad \text{for } x \in \bar{\Omega}. \tag{11}$$

The Burgers equation (10) arises in various fields, including fluid mechanics, nonlinear acoustics, and traffic flow. The viscosity parameter ν plays an important role in its solution behavior. For small values of ν, Burgers equation can lead to shock formation.

Take the computational domain $\Omega = (-1, 1)$ and end time $T = 1$. To obtain high-fidelity solutions, we solve (10) using the finite difference method in space and a fourth-order Runge–Kutta (RK4) scheme in time, with $N = 512$ grid points and a time step of 10^{-5}. The training dataset consists of numerical solutions for 30 parameter sets, selected from combinations of $\mu \in \{0.1, 0.3, 0.5, 0.7, 1\}$ and $\nu \in \{0.01, 0.04, 0.08, 0.12, 0.16, 0.2\}$. For each parameter set, 11 snapshots of the solutions are taken at time $t = 0, 0.1, 0.2, \ldots, 1$. Note that our surrogate solver can also handle nonuniform time-step data, which is a key advantage over other existing methods [18, 23, 25]. The testing dataset include numerical solutions for 39 pairs of parameters (μ, ν) randomly chosen on the parameter space $(0.1, 1) \times (0.01, 0.2)$.

In our surrogate solver, the encoder includes eight 1D convolutional layers, each with 2, 4, 8, 16, 32, 64, 128, and 256 filters respectively. Each convolutional layer is followed by a max-pooling layer with a pooling size of 2. For this 1D problem, no fully-connected layers are used. Instead, an additional convolutional layer with 1 filter is used as the output layer in both the encoder and decoder to aggregate information across the channels. The same CNN structure used for the parametric Poisson problem is applied to the latent-CNN. We set the number of channels to match the number of parameters. In this case, since time t is treated as a parameter, there are three channels.

Figure 8 compares the predicted solutions with the ground truth solutions for three randomly selected parameter sets in $\mathcal{S}_{\text{test}}$, where the latent dimension is $n = 2$. For a larger value of ν, the solution tends to smooth out over time. Conversely, for smaller values of ν, a shock wave emerges over time, which is usually challenging to

capture. The results show that in both cases, our surrogate solver provides accurate predictions compared to the ground truth solutions.

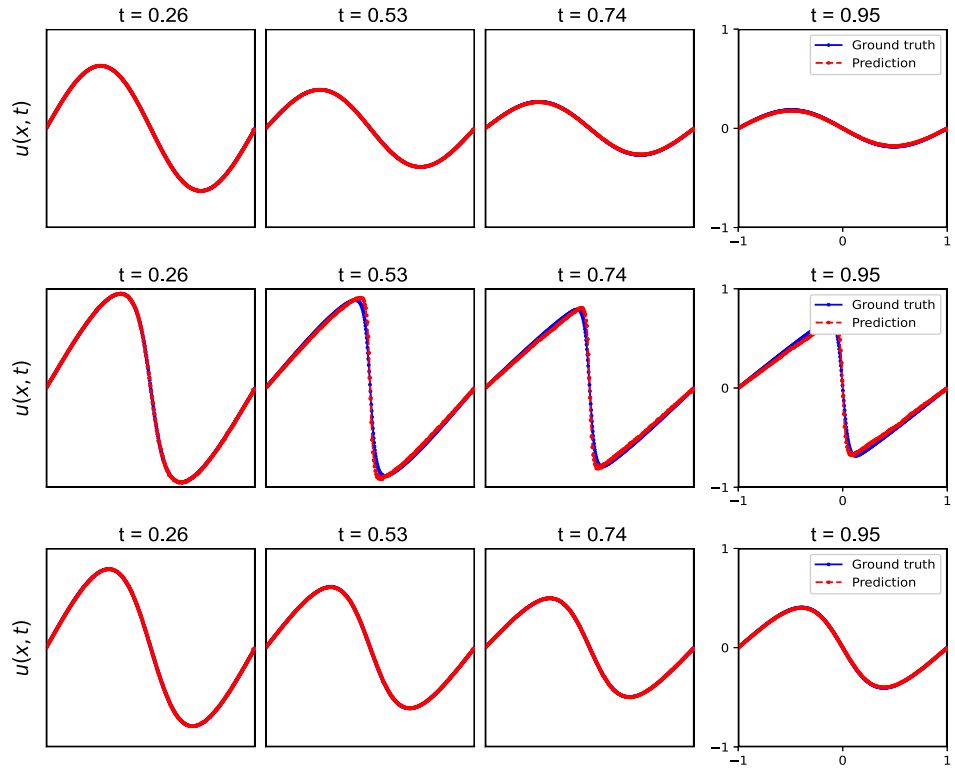


FIGURE 8. Comparison of the ground truth and predicted solutions of the viscous Burgers equation. Each row shows the solution dynamics for a randomly chosen parameter set (μ, ν). From top to bottom: $(\mu, \nu) = (0.2, 0.18), (0.9, 0.02)$, and $(0.6, 0.09)$.

It is important to note that the time points displayed in Figure 8 are different from those used in the training data. Compared to the time evolution approach, such as LSTM [23, 25], our surrogate model excels at providing predictions for times not included in the training data. Moreover, our model does not require the training data to be given at uniform time steps. Our CNN learns a continuous relationship between time and the encoded solution within the latent space. Hence, it can predict the encoded solution for any time t within the studied time interval. Extensive numerical studies show that our proposed neural network solver can effectively predict solutions across the parameter space and time $t \in [0, T]$.

3.3. Buckley–Leverett equation. Consider the 2D Buckley–Leverett equation of the form [33, 34]:

$$\begin{aligned} \frac{\partial u(\mathbf{x}, t; \mu)}{\partial t} + \frac{\partial f_1(u)}{\partial x} + \frac{\partial f_2(u)}{\partial y} - \mu \Delta u = 0, \text{ for } (x, y) \in \Omega, \\ u(x, y, t) = 0, \text{ for } (x, y) \in \partial\Omega, \end{aligned} \quad t \in (0, T], \tag{12}$$

where the parameter $\mu \in [0.01, 0.1]$. The nonlinear flux functions $f_1(u)$ and $f_2(u)$ are defined as

$$f_1(u) = \frac{u^2}{u^2 + (1-u)^2}, \qquad f_2(u) = f_1(u)\left(1 - 5(1-u)^2\right).$$

The Buckley–Leverett equation is often used to describe two-phase flow in porous media [33]. The diffusion term introduces a smoothing effect and models the dispersion and capillary pressure effects. The parameter μ usually depends on the porous media and the fluid involved. Numerical challenges in solving (12) come from its highly nonlinear flux terms, which often results in sharp fronts or discontinuities in the saturation profile.

In our study, we take $\Omega = (-1.5, 1.5)^2$ and $T = 0.5$. The initial condition is chosen as

$$u(x, y, 0) = \begin{cases} 1, & \text{for } x^2 + y^2 < 0.5, \\ 0, & \text{otherwise.} \end{cases}$$

To prepare the training and testing data, we solve the problem using the finite difference method with $N = 256^2$ grid points and the RK4 temporal discretization with a time step of 0.0001. The training data include numerical solutions for 10 parameters that are uniformly sampled on $[0.01, 0.1]$. For each training data, all spatial points are included (i.e., $N = 256^2$), but only 11 time snapshots (i.e., $N_t = 11$) of solutions are considered, sampled at uniformly distributed time points within the interval $[0, 0.5]$. The testing data consist of numerical solutions for 57 randomly sampled parameters in the parameter interval $(0.01, 0.1)$.

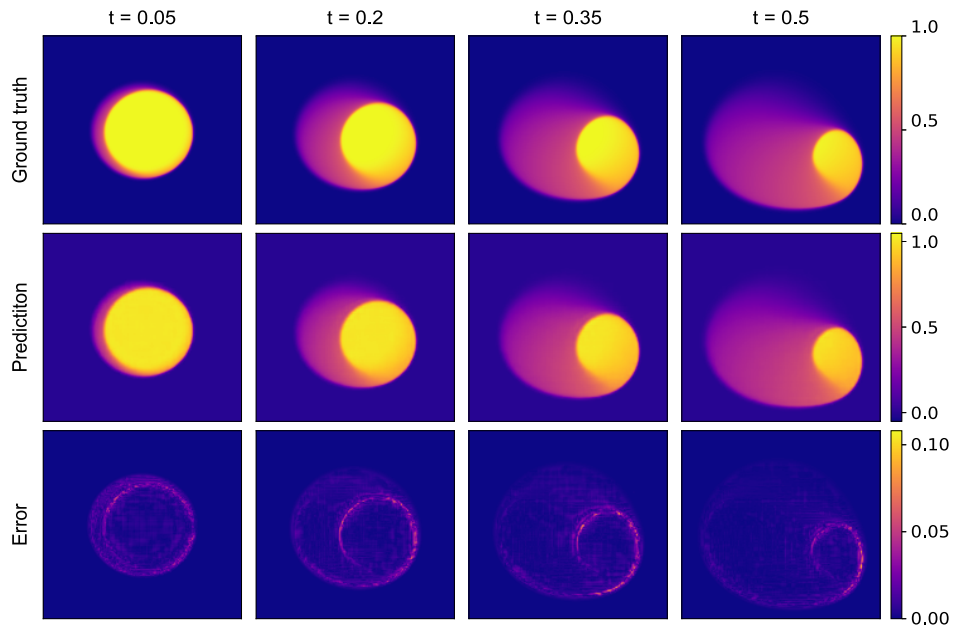


FIGURE 9. Comparison of the ground truth solution ($\mathbf{u}$) and predicted solution ($\widehat{\mathbf{u}}$) of the parametric Buckley–Leverett equation with $\mu = 0.017$, where the error is computed as $|\mathbf{u} - \widehat{\mathbf{u}}|$. For easy comparison, the results in each row use the same colorbar scale.

The neural network structure in this example is similar to that used in Section 3.1 for the parametric Poisson problem. Specifically, we consider the encoder with five 2D convolutional layers, and each has 8, 16, 32, 64, and 256 filters, respectively. Each convolutional layer is followed by a 2D max-pooling layer with pooling size of 2×2. Two fully-connected layers with 50 and 6 neurons, respectively, are added after the last pooling layer. The decoder mirrors the structure of the encoder, with the addition of a fully-connected layer containing 512 neurons placed before the first 2D convolutional layers.

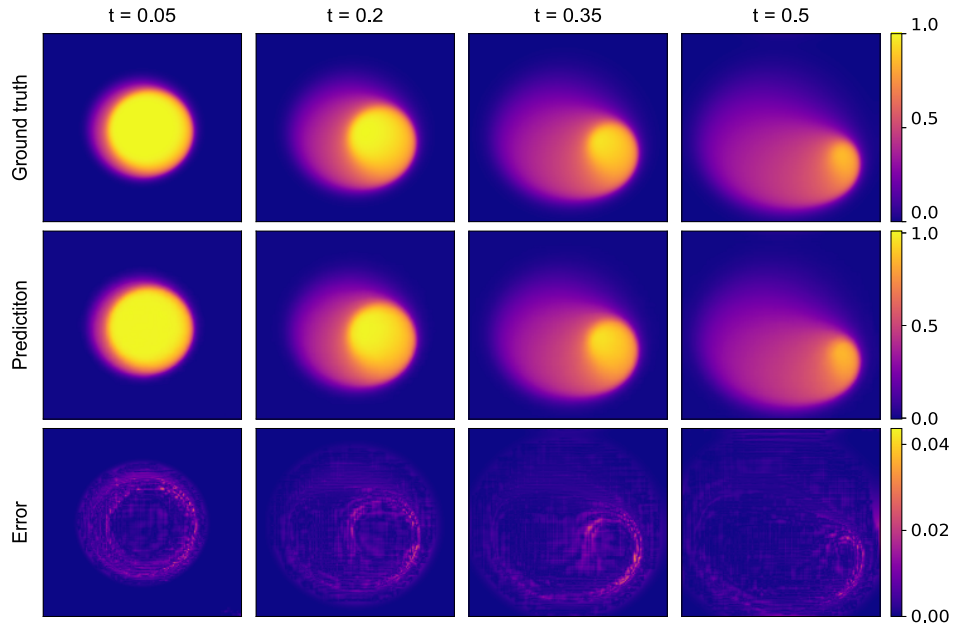


FIGURE 10. Comparison of the ground truth solution ($\mathbf{u}$) and predicted solution ($\widehat{\mathbf{u}}$) of the parametric Buckley–Leverett equation with $\mu = 0.095$, where the error is computed as $|\mathbf{u} - \widehat{\mathbf{u}}|$. For easy comparison, the results in each row use the same colorbar scale.

Figures 9 and 10 compare the ground truth and predicted solutions at different times for randomly chosen parameter $\mu = 0.017$ and $\mu = 0.095$, respectively, where the latent dimension $n = 6$. It shows that the predicted solutions agree well with the ground truth solutions. Our surrogate solver can predict the solution for unseen parameters effectively. Moreover, it is much faster than traditional numerical methods. See the comparison of their computational times in Table 1. To obtain the solution of a new parameter μ, traditional numerical methods require a significant amount of time. In contrast, once the CAE and CNN are trained, our surrogate solver can quickly predict the solution for a given parameter μ – first inputting the parameter μ and time t into the CNN to obtain the encoded solution, which is then decoded to produce the predicted solution in the original high-dimensional space.

Figure 11 further compares the compression and reconstruction abilities of CAE and PCA, where the latent dimension $n = 6$. Note that the latent-space modeling is not involved for this purpose. Figure 11 shows that with a latent dimension $n = 6$, it is challenging for PCA to capture sharp fronts, resulting in significantly larger errors compared to CAE. Generally, CAE can achieve better accuracy in

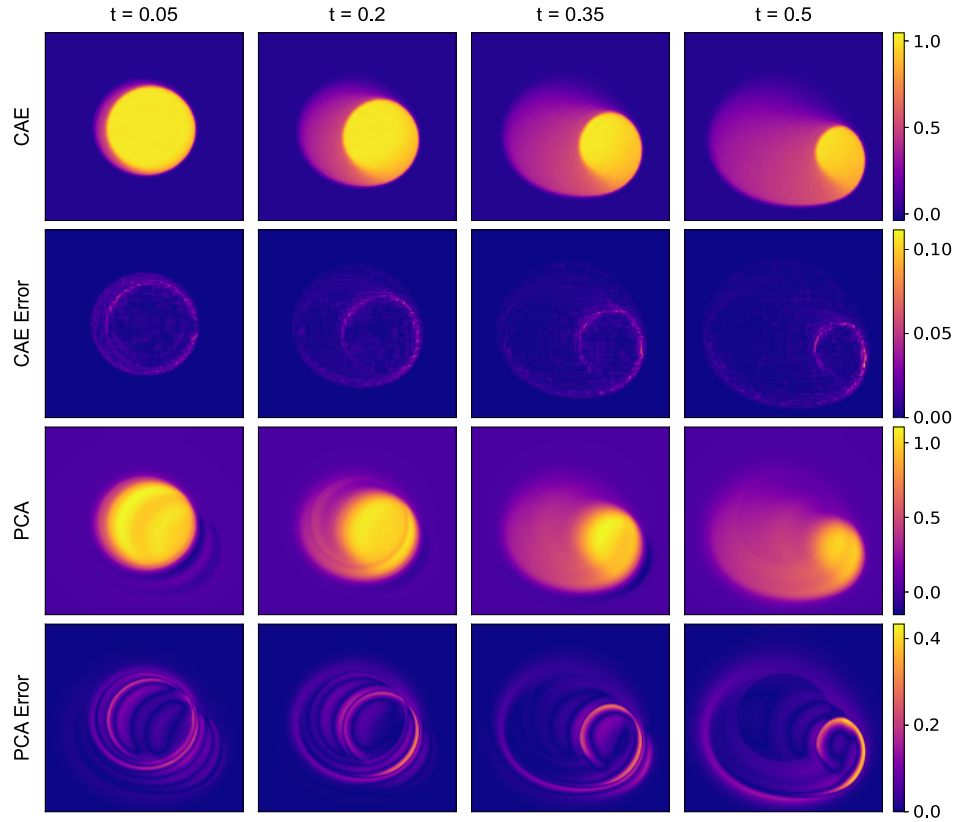


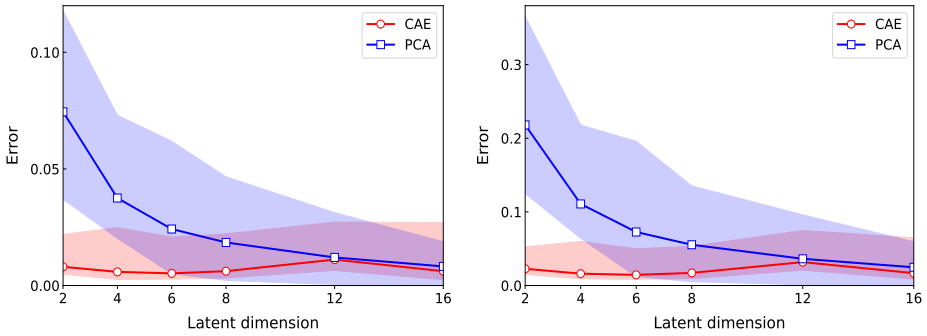
FIGURE 11. Comparison of CAE and PCA for the Buckley–Leverett equation with $\mu = 0.017$ and latent dimension $n = 6$. For easy comparison, the results in each row use the same colorbar scale.

compressing and reconstructing data when the latent dimension is small. Similar to our observations in Section 3.1, the accuracy of PCA improves as the latent dimension n increases. However, even with a dimension of $n = 32$, the CAE still outperforms PCA in this case.

Figure 12 further illustrates the aggregate errors across all test parameters and time. For a given parameter $\mu = \mu_s$, the absolute and relative errors are defined as:

$$\text{(13)} \qquad \mathrm{e}_{\mathrm{abs}}^{\mu_s} = \frac{1}{N_t^{\mathrm{test}}} \sum_{m=1}^{N_t^{\mathrm{test}}} \|\mathbf{u}_s^m - \widehat{\mathbf{u}}_s^m\|_{\mathrm{rms}}, \qquad \mathrm{e}_{\mathrm{rel}}^{\mu_s} = \frac{1}{N_t^{\mathrm{test}}} \sum_{m=1}^{N_t^{\mathrm{test}}} \frac{\|\mathbf{u}_s^m - \widehat{\mathbf{u}}_s^m\|_{\mathrm{rms}}}{\|\mathbf{u}_s^m\|_{\mathrm{rms}}},$$

respectively. Furthermore, we define $\bar{\mathrm{e}}_{\mathrm{abs}}$, $\mathrm{e}_{\mathrm{abs}}^{\max}$, and $\mathrm{e}_{\mathrm{abs}}^{\min}$ as the average, maximum, and minimum of the absolute errors $\mathrm{e}_{\mathrm{abs}}^{\mu_s}$, respectively, across all test parameters μ_s for $1 \le s \le N_\mu^{\mathrm{test}}$. Similar notations apply to the relative errors $\mathrm{e}_{\mathrm{rel}}^{\mu_s}$. It shows that the reconstruction errors of CAE remain within a narrow range even with small latent dimensions, whereas PCA has much larger errors. This indicates that CAE maintains stable performance across various test data. Compared to the results in Section 3.1, the advantage of CAE over PCA is more significant in this case.



(a) (b)

FIGURE 12. (a) Absolute errors e^{μ}_{abs} of CAE and PCA across all test data at different latent dimensions, with no latent space modeling involved. The solid line represents the average errors $\bar{e}_{abs}$, while the shaded area indicates the range between maximum e^{max}_{abs} and minimum e^{min}_{abs} errors. (b) Corresponding results for relative errors e^{μ}_{rel}.

3.4. Radial advection equation. Consider the 2D radial advection equation [17]:

$$\frac{\partial u(\mathbf{x},t;\mu)}{\partial t} = -v(\mathbf{x})\cdot\nabla u, \qquad \text{for } \mathbf{x}\in\Omega, \quad t\in(0,T] \tag{14}$$

with periodic boundary conditions, where the coefficient function is given by

$$v(\mathbf{x}) = \frac{\pi}{2}(1-x^2)^2(1-y^2)^2\begin{bmatrix} x \\ -y \end{bmatrix}.$$

The parametrized initial condition takes the form

$$u(\mathbf{x},0;\mu) = \cos(\pi\mu x)\cos(\pi\mu y), \qquad \text{for } \mathbf{x}\in\bar{\Omega} \tag{15}$$

with the parameter $\mu\in[0.6,2]$. Here, we take domain $\Omega=(-1,1)^2$ and end time $T=3$.

To prepare the training and testing data, we solve the problem (14)–(15) using finite difference methods with $N=256^2$ grid points (i.e., 256 points in each of the x- and y-directions) and the RK4 temporal discretization with a time step of 0.005. Although a small time step is used to generate data, the training data of each parameter μ include only 16 time snapshots, i.e., $N_t^{train}=16$, uniformly sampled at $t=0,0.2,0.4,\ldots,3$. The training data include numerical solutions for 15 parameters which are uniformly sampled on the parameter interval $[0.6,2]$. The testing data consist of numerical solutions for 64 randomly chosen parameters μ in the interval $(0.6,2)$. The neural network structures in this example are similar to those used in solving the Buckley–Leverett equation.

Figures 13–14 compare the ground truth and predicted solutions for parameters $\mu=0.75$ and 1.63, respectively, where the latent dimension $n=4$. It shows that the predicted solutions agree very well with the ground truth solutions. For large values of μ, the initial condition becomes more oscillatory, leading to increasingly oscillatory solution over time. Moreover, Figure 15 presents both the absolute and relative errors for different parameters and times. Figure 15 (b) and (d) show that errors remain low for all test parameters μ, even though only 15 parameters are

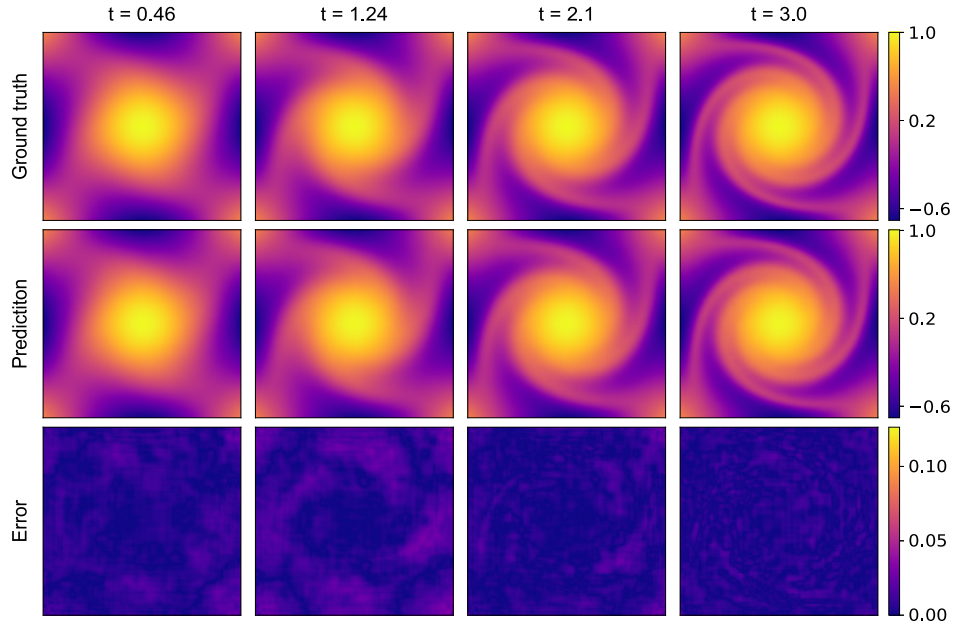


FIGURE 13. Comparison of the ground truth solution ($\mathbf{u}$) and predicted solution ($\widehat{\mathbf{u}}$) of the radial advection equation with $\mu = 0.75$, where the latent dimension $n = 4$. The error is computed as $|\mathbf{u}-\widehat{\mathbf{u}}|$. For easy comparison, the results in each row use the same colorbar scale.

used in the training data. Figure 15 (a) and (c) demonstrate that errors remain small and bounded over time t, in contrast to traditional numerical methods where numerical errors generally increase over time. For $\mu = 1.63$, numerical errors spike between $t = 2.8$ and $t = 3$ due to the rapid changes in the solution within this time interval, although the errors at $t = 2.8$ and $t = 3$ are relatively small. To enhance prediction accuracy, including more time snapshots in this interval could be beneficial. As seen in Figure 15 (a) and (c), our method can predict solutions at any time $t \in [0, 3]$, even though only 16 time points are used in the training data.

Similarly, we compare the compression and reconstruction capabilities of CAE and PCA for solving the 2D radial advection problem. Again, the latent-space modeling is not involved for this purpose. Figure 16 clearly shows that with a latent dimension of $n = 4$, CAE achieves a reconstruction error less than 0.05 across all time points. However, the reconstruction errors of PCA are larger than 0.8. The results in Figure 17 further show that the CAE outperforms PCA for latent dimensions $n \leq 12$. As the latent dimension increases, the reconstruction errors of PCA continue to reduce and eventually become smaller than those of the CAE. Thus, CAE is preferable due to its superior compression and reconstruction capabilities, especially when using a small latent dimension.

3.5. Comparison to traditional numerical methods. To further assess its efficiency, we provide a brief comparison between our surrogate solver and traditional numerical methods for solving each time-dependent problem in Sections 3.1–3.4. Table 1 presents their computational times (in seconds), with details on both traditional numerical methods and our surrogate model as discussed in Sections 3.1–3.4.

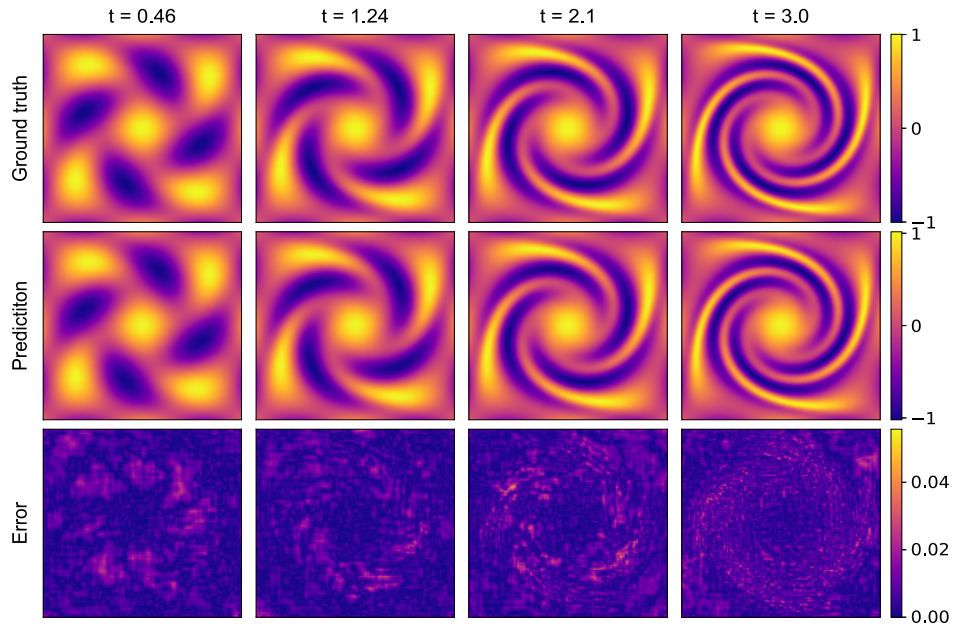


FIGURE 14. Comparison of the ground truth solution ($\mathbf{u}$) and predicted solution ($\widehat{\mathbf{u}}$) of the radial advection equation with $\mu = 1.63$, where the latent dimension $n = 4$. The error is computed as $|\mathbf{u}-\widehat{\mathbf{u}}|$. For easy comparison, the results in each row use the same colorbar scale.

The computations are performed on a macOS Monterey system equipped with a 10-core 3.6 GHz Intel i9 CPU and 128 GB of 2667 MHz DDR4 memory.

For traditional numerical methods, explicit time-stepping methods are used for each problem, where small time steps are used to ensure stability conditions. In contrast, implicit methods allow for larger time steps. However, they require numerical iterations at each step, resulting in longer computational time per step, despite potentially reducing the total number of steps. Further discussion of different numerical methods is beyond the scope of this paper. For our surrogate solver, we provide the times 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.

Table 1 shows that: (i) The time of training the CAE dominates the offline stage time, mainly depending on the data dimension and volume. Generally, higher-dimensional data, such as the $N = 65536$ dimensions in radial advection problems compared to $N = 512$ in the Burgers' equation, results in longer training times. Moreover, more training data and time snapshots are used in the radial advection equation compared to the Buckley–Leverett equation, leading to longer training times. (ii) The training time for the CNN is relatively insignificant compared to the CAE.

To compare our surrogate solver with traditional numerical methods, we present the total computational time required to predict the solution for $N_{\boldsymbol{\mu}}^{\text{pred}} = 1000$ parameter sets $\boldsymbol{\mu}$ at different time t. In other words, the times listed under "*Online prediction*" and "*Numerical methods*" columns represent the total computational

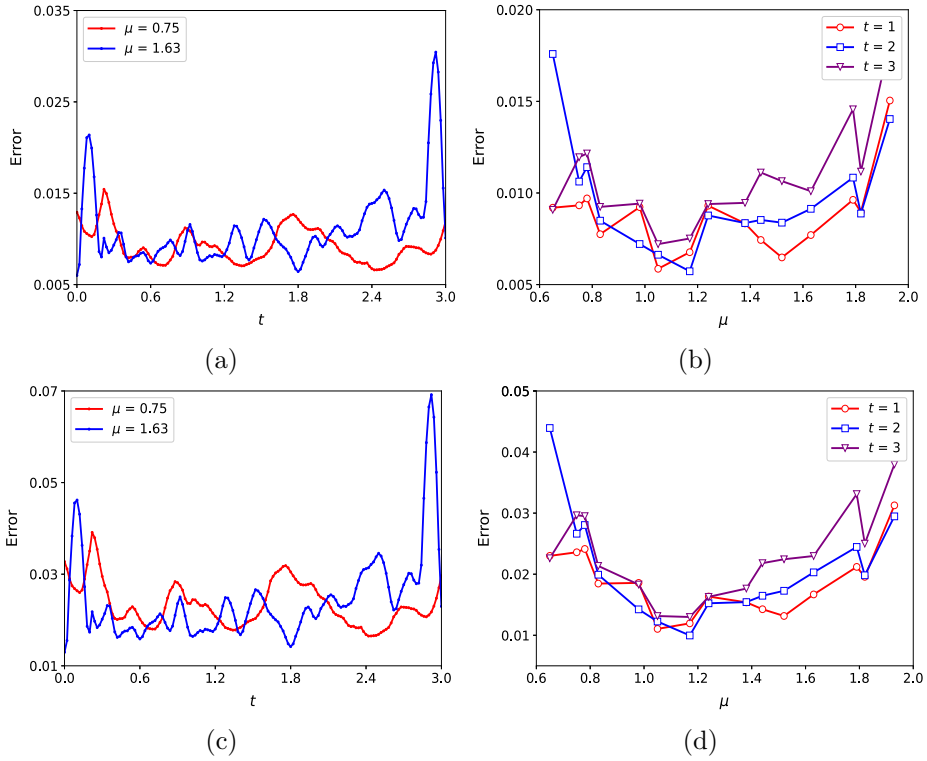


FIGURE 15. (a) Absolute errors $\|\mathbf{u} - \widehat{\mathbf{u}}\|_{\text{rms}}$ over time t and (b) across different parameters μ. Panels (c) and (d) show the corresponding results of relative errors $\|\mathbf{u} - \widehat{\mathbf{u}}\|_{\text{rms}}/\|\mathbf{u}\|_{\text{rms}}$.

TABLE 1. Comparison of computational time of our surrogate solver and traditional numerical methods for solving problems in Sections 3.2–3.4. The times listed under "*Online prediction*" and "*Numerical methods*" columns are the total computational time spent to compute the solution $u(\cdot, t; \boldsymbol{\mu})$ for 1000 parameter sets $\boldsymbol{\mu}$.

Problems	T	Our surrogate solver				Numerical methods	
		Offline training		Online prediction			
		CAE	CNN	$t = T/2$	$t = T$	$t = T/2$	$t = T$
Section 3.2	1.0	0.909e+3	139	0.113	0.108	1.077e+3	2.164e+3
Section 3.3	0.5	1.122e+4	111	1.582	1.606	1.380e+6	2.792e+6
Section 3.4	3.0	2.386e+4	209	1.683	1.659	8.468e+4	1.706e+5

time spent to compute the solution for 1000 parameter sets $\boldsymbol{\mu}$. 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 $t = T/2$ and $t = T$ in Table 1) In contrast, our surrogate model predicts the solution directly as $\widehat{\mathbf{u}} = \Phi(\Theta(\boldsymbol{\mu}, t; \vartheta_l^*); \vartheta_d^*)$, involving only the trained CNN and decoder. This significantly reduces the computational

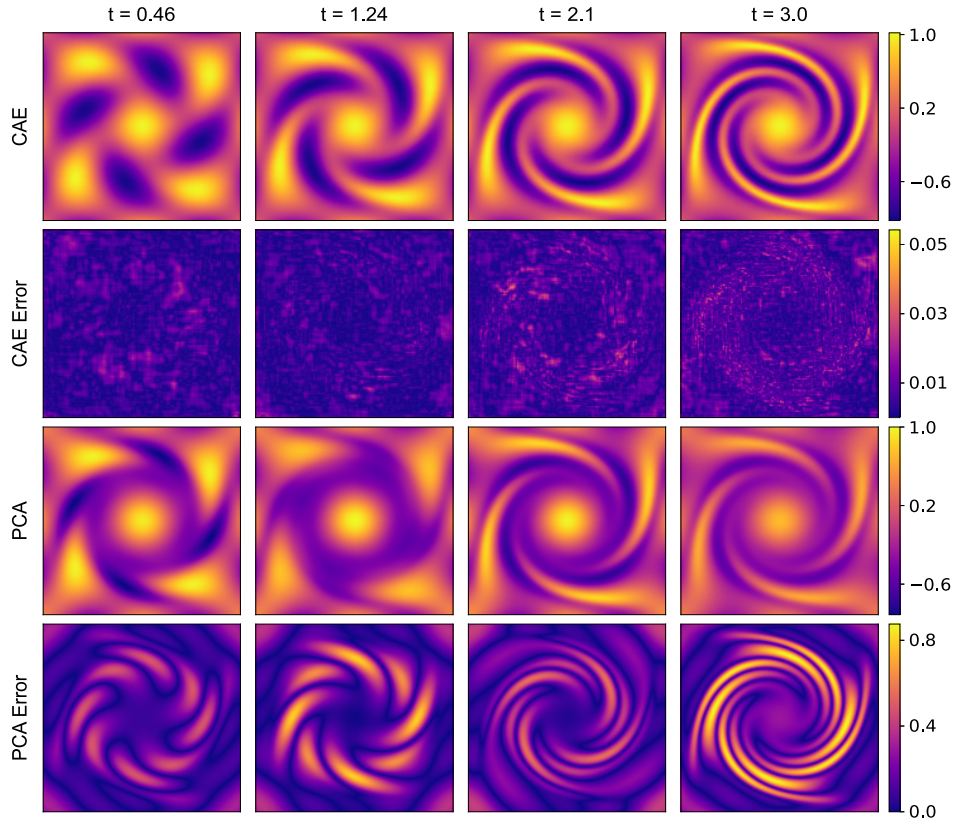


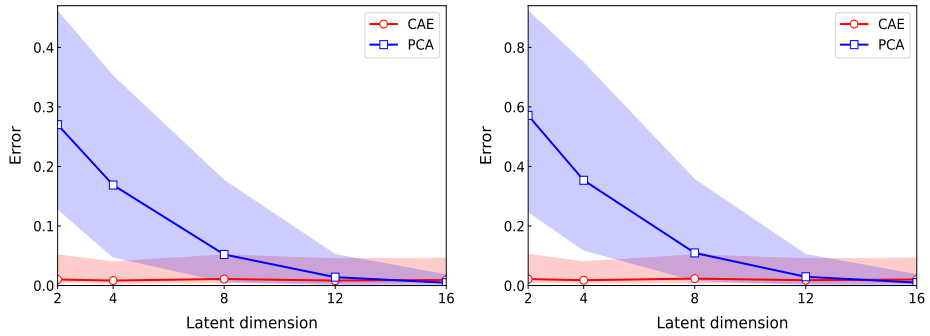
FIGURE 16. Comparison of CAE and PCA for the radial advection problem with $\mu = 1.63$ and latent dimension $n = 4$. For easy comparison, the results in each row use the same colorbar scale.

time. Therefore, our method is more efficient for computing solutions of parametric PDEs across different parameters (e.g., in optimal control problems).

4. Conclusion

We proposed a CNN surrogate model for solving parametric PDEs, including time-independent and time-dependent problems. Our surrogate model consists of two parts: a CAE that learns a low-dimensional latent space representation of the high-fidelity data, and a latent CNN that maps the model parameters to the latent space representation, with the model parameters serving as the input to the CNN. To address the time-dependent problems, we embedded time t into the surrogate model by treating it as an additional model parameter, i.e., as another input of the CNN. To enhance computational efficiency, we adopted a decoupled training strategy – training the CAE and CNN mapping separately. In the online stage, our surrogate solver is constructed by combining the trained latent CNN with the decoder.

Our surrogate model offers the following advantages: First, it does not require the training data to be sampled at uniform time steps. Second, it can predict the solution at an arbitrary time t within the time domain, and the prediction only needs to evaluate the model (only the latent CNN and the decoder) once.



(a) (b)

FIGURE 17. (a) Absolute errors e^{μ}_{abs} of CAE and PCA across all test data at different latent dimension, with no latent space modeling involved. The solid line represents the average errors $\bar{e}_{abs}$, while the shaded area indicates the range between maximum e^{max}_{abs} and minimum e^{min}_{abs} errors. (b) Corresponding results for relative errors e^{μ}_{rel}.

In contrast, existing LSTM-based ROM methods in the literature require training data to be sampled at uniform time steps. In the online stage, these methods start from encoded initial conditions and compute the solution step by step until reaching the prediction time t. Extensive numerical experiments showed that our surrogate model can accurately predict solutions for both time-dependent and time-independent problems. Comparisons with PCA further demonstrate its superior compression and reconstruction capabilities. Specifically, for the Burgers equation and the BuckleyCLeverett equation, our surrogate model accurately captures the sharp fronts of the solutions.

Acknowledgments

The authors thank the anonymous reviewers for their valuable feedback in improving the manuscript. This work was partially supported by the US National Science Foundation under grant number DMS–1953177.

References

[1] M. Asch, M. Bocquet, and M. Nodet. Data Assimilation: Methods, Algorithms, and Applications. SIAM, Philadelphia, PA, 2016.

[2] C. Audouze, F. De Vuyst, and P. B. Nair. Reduced-order modeling of parameterized PDEs using time–space-parameter principal component analysis. Int. J. Numer. Methods Eng., 80(8):1025–1057, 2009.

[3] F. Ballarin, A. Manzoni, A. Quarteroni, and G. Rozza. Supremizer stabilization of POD–Galerkin approximation of parametrized steady incompressible Navier–Stokes equations. Int. J. Numer. Methods Eng., 102(5):1136–1161, 2015.

[4] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart. Model reduction and neural networks for parametric PDEs. SMAI J. Comput. Math., 7:121–157, 2021.

[5] H. Bijl, D. Lucor, S. Mishra, and C. Schwab. Uncertainty Quantification in Computational Fluid Dynamics, volume 92 of Lect. Notes Comput. Sci. Eng. Springer, 2013.

[6] G. Carere, M. Strazzullo, F. Ballarin, G. Rozza, and R. Stevenson. A weighted POD-reduction approach for parametrized PDE-constrained optimal control problems with random inputs and applications to environmental sciences. Comput. Math. Appl., 102:261–276, 2021.

[7] K. T. Carlberg, M. Barone, and H. Antil. Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction. J. Comput. Phys., 330:693–734, 2017.

[8] D. Chapelle, A. Gariah, P. Moireau, and J. Sainte-Marie. A Galerkin strategy with proper orthogonal decomposition for parameter-dependent problems – Analysis, assessments and applications to parameter estimation. ESAIM: Math. Model. Numer. Anal., 47(6):1821–1843, 2013.

[9] A. Cohen and R. DeVore. Approximation of high-dimensional parametric PDEs. Acta Numer., 24:1–159, 2015.

[10] N. Dal Santo, S. Deparis, and L. Pegolotti. Data driven approximation of parametrized PDEs by reduced basis and neural networks. J. Comput. Phys., 416:109550, 2020.

[11] L. Dede. Reduced basis method for parametrized elliptic advection-reaction problems. J. Comput. Math., 28(1):122–148, 2010.

[12] D. Draper, A. Pereira, P. Prado, A. Saltelli, R. Cheal, S. Eguilior, B. Mendes, and S. Tarantola. Scenario and parametric uncertainty in GESAMAC:A methodological study in nuclear waste disposal risk assessment. Comput. Phys. Commun., 117(1-2):142–155, 2011.

[13] F. Fatone, S. Fresca, and A. Manzoni. Long-time prediction of nonlinear parametrized dynamical systems by deep learning-based reduced order models. Math. Eng., 5:1–26, 2023.

[14] N. Franco, A. Manzoni, and P. Zunino. A deep learning approach to reduced order modelling of parameter dependent partial differential equations. Math. Comput., 92(340):483–524, 2023.

[15] S. Fresca, L. Dedè, and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. Math. Comput., 87:1–36, 2021.

[16] S. Fresca, A. Manzoni, L. Dedè, and A. Quarteroni. Deep learning-based reduced order models in cardiac electrophysiology. PLoS ONE, 15:e0239416, 2020.

[17] W. D. Fries, X. He, and Y. Choi. LaSDI: Parametric latent space dynamics identification. Comput. Methods Appl. Mech. Eng., 399:115436, 2022.

[18] F. J. Gonzalez and M. Balajewicz. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. arXiv:1808.01346, 2018.

[19] K. Ito and S. S. Ravindran. A reduced-order method for simulation and control of fluid flows. J. Comput. Phys., 143(2):403–425, 1998.

[20] K. Lee and K. T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. J. Comput. Phys., 404:108973, 2020.

[21] B. Lusch, J. N. Kutz, and S. L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. Nat. Commun., 9(1):4950, 2018.

[22] A. Manzoni, A. Quarteroni, and G. Rozza. Computational reduction for parametrized PDEs: strategies and applications. Milan J. Math., 80:283–309, 2012.

[23] R. Maulik, B. Lusch, and P. Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. Phys. Fluids, 33(3):037106, 2021.

[24] N. T. Mücke, S. M. Bohté, and C. W. Oosterlee. Reduced order modeling for parameterized time-dependent pdes using spatially and memory aware deep learning. J. Comput. Sci., 53:101408, 2021.

[25] T. Nakamura, K. Fukami, K. Hasegawa, Y. Nabae, and K. Fukagata. Convolutional neural network and long short-term memory based reduced order surrogate for minimal turbulent channel flow. Phys. Fluids, 33(2), 2021.

[26] S. Nikolopoulos, I. Kalogeris, and V. Papadopoulos. Non-intrusive surrogate modeling for parametrized time-dependent partial differential equations using convolutional autoencoders. Eng. Appl. Artif. Intell., 109:104652, 2022.

[27] L. Peng and K. Mohseni. Nonlinear model reduction via a locally weighted POD method. Int. J. Numer. Methods Eng., 106(5):372–396, 2016.

[28] F. Pichi, B. Moya, and J. S. Hesthaven. A graph convolutional autoencoder approach to model order reduction for parametrized PDEs. J. Comput. Phys., 501:112762, 2024.

[29] A. Quarteroni, A. Manzoni, and F. Negri. Reduced Basis Methods for Partial Differential Equations, volume 92. Springer, 2016.

[30] O. Rukundo and H. Cao. Nearest neighbor value interpolation. Int. J. Adv. Comput. Sci. Appl., 3(4), 2012.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations, pages 318–362. MIT Press, 1986.

[32] A. A. Shah, W. Xing, and V. Triantafyllidis. Reduced-order modelling of parameter-dependent, linear and nonlinear dynamic partial differential equation models. Proc. R. Soc. A: Math. Phys. Eng. Sci., 473(2200):20160809, 2017.
[33] P. Yang, T. Xiong, J. M. Qiu, and Z. Xu. High order maximum principle preserving finite volume method for convection dominated problems. J. Sci. Comput., 67:795–820, 2016.
[34] K. Yu, J. Cheng, Y. Liu, and C. W. Shu. High-order implicit maximum-principle-preserving local discontinuous Galerkin methods for convection-diffusion equations. preprint, 2024.
[35] Y. Zhang, J. Peterson, and M. Gunzburger. Maximizing critical currents in superconductors by optimization of normal inclusion properties. Phys. D: Nonlinear Phenom., 240(21):1701–1713, 2011.

Department of Mathematics and Statistics, Missouri University of Science and Technology, Rolla, MO 65409, USA

E-mail: yw2bc@umsystem.edu, szb5g@umsystem.edu, zhangyanz@umsystem.edu