Computer Methods in Applied Mechanics and Engineering 419 (2024) 116684

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

A nonlinear-manifold reduced-order model and operator learning for partial differential equations with sharp solution gradients

Peiyi Chen [a], Tianchen Hu [b], Johann Guilleminot [c,a,*]

[a] Department of Mechanical Engineering and Materials Science, Duke University, NC, 27710, USA
[b] Currently at Argonne National Laboratory, Lemont, IL, 60439, USA
[c] Department of Civil and Environmental Engineering, Duke University, NC, 27710, USA

ARTICLE INFO

Keywords:
Burgers equation
Machine learning
Operator inference
Reduced-order modeling
Sharp gradient

ABSTRACT

Traditional linear subspace-based reduced order models (LS-ROMs) can be used to significantly accelerate simulations in which the solution space of the discretized system has a small dimension (with a fast decaying Kolmogorov n-width). However, LS-ROMs struggle to achieve speed-ups in problems whose solution space has a large dimension, such as highly nonlinear problems whose solutions have large gradients. Such an issue can be alleviated by combining nonlinear model reduction with operator learning. Over the past decade, many nonlinear manifold-based reduced order models (NM-ROM) have been proposed. In particular, NM-ROMs based on deep neural networks (DNN) have received increasing interest. This work takes inspiration from adaptive basis methods and specifically focuses on developing an NM-ROM based on Convolutional Neural Network-based autoencoders (CNNAE) with iteration-dependent trainable kernels. Additionally, we investigate DNN-based and quadratic operator inference strategies between latent spaces. A strategy to perform vectorized implicit time integration is also proposed. We demonstrate that the proposed CNN-based NM-ROM, combined with DNN-based operator inference, generally performs better than commonly employed strategies (in terms of prediction accuracy) on a benchmark advection-dominated problem. The method also presents substantial gain in terms of training speed per epoch, with a training time about one order of magnitude smaller than the one associated with a state-of-the-art technique performing with the same level of accuracy.

1. Introduction

1.1. Background

In many decision-making applications, e.g. inverse problem, optimal control, uncertainty quantification, etc., the outer loop requires a large number of forward evaluations of the full order model (FOM). However, oftentimes a high-fidelity model, in the form of coupled partial differential equations (PDEs), is computationally prohibitive as a feasible forward simulation within the outer loop. To avoid evaluating the expensive high-fidelity model in a forward simulation, a reduced order model (ROM) can be trained as its surrogate with reasonable accuracy, hence significantly reducing the computational cost of the decision-making applications.

Nonlinear PDEs describe a wide spectrum of engineering problems. High-fidelity numerical solvers based on high-dimensional spatial discretizations are prohibitively expensive in many-query settings. Intrusive and non-intrusive LS-ROMs reduce the dimensionality of the model by projecting the solution of the high-dimensional model onto a linear subspace. Several approaches to

* Corresponding author at: Department of Civil and Environmental Engineering, Duke University, NC, 27710, USA.
E-mail address: johann.guilleminot@duke.edu (J. Guilleminot).

https://doi.org/10.1016/j.cma.2023.116684
Received 14 October 2023; Received in revised form 13 November 2023; Accepted 3 December 2023

defining the approximation subspace have been proposed (see e.g. [1] for a review). A common empirical choice is the proper orthogonal decomposition (POD) subspace, which spans the leading principal components of an available set of state data [2–4]. For PDEs with only linear and polynomial terms, the projection-based reduced model admits an efficient low-dimensional numerical representation that can be cheaply evaluated at a cost independent of the dimension of the original model [5,1,6–8]. For PDEs with more general nonlinear terms, an additional level of approximation, e.g., interpolation [9–15], is generally needed to achieve computational efficiency. Alternatively, the works in [5,7,16,17] transform PDEs with general nonlinear terms to representations with only quadratic nonlinearities via the use of structure-exposing lifting transformations, and then project the quadratic operators of the resultant lifted PDE to obtain a reduced model.

In spite of its many successes, the linear subspace solution representation suffers from the inability to represent certain physical simulation solutions with a small basis dimension, such as advection-dominated problems whose solutions possess large spatial gradients. This is because linear subspace can only retain a small dimensionality for problems with a fast decaying Kolmogorov n-width, i.e., the solution can be well-represented by a small number of basis functions. Problems with a slowly decaying Kolmogorov n-width include, but are not limited to, the hyperbolic equations with high Reynolds number, the Boltzmann transport equations, and the traffic flow simulations. As one way to alleviate such issue, there have been many attempts to replace the linear subspace solution representation with nonlinear manifolds (see also [18] for another closure strategy). In recent years, machine learning based approaches to NM-ROMs have received an increasing interest. Some treat the weights and biases of a deep neural network (DNN) as unknowns in the solution process [19–22], while others incorporate physical laws into DNN-based surrogates whose weights and biases are determined in the training phase [23–29]. Recently, a physics-informed NM-ROM was proposed in [30], where a shallow masked autoencoder and hyper-reduction techniques are used to achieve a speed-up of the NM-ROM compared to the corresponding FOM. However, results provided by standard DNN-based NM-ROM remain hard to analyze beyond accuracy evaluation, due to the lack of interpretability in deep learning models. As a result, it is difficult to determine the structure and parameters of the DNNs a priori, and a huge amount of effort has to be made to tune the NM-ROMs to achieve good performance.

In this work, we devise a novel structure for the decoder in the NM-ROM, such that the underlying nonlinear mapping closely resembles the formulation of adaptive basis methods. In essence, the non-linearity in the proposed NM-ROM is introduced through a DNN in the smoothing kernels. The nonlinear manifolds suggested by the trained NM-ROM allows one to gain additional insight into the nonlinearities of the corresponding FOM. We also propose a DNN-based reduced operator inference that learns from the latent space for full models based on time-dependent partial differential equations. Note that the issue of learning operators associated with ordinary and partial differential equations (in physical or latent spaces) has attracted much attention over the last decade. Providing an exhaustive review on – and a fair numerical comparison between – all existing approaches is outside the scope of the present work (if at all possible); see [31–34,25,26,35–46] to list a few, as well as [47,48] for comparisons between some operator learning techniques (see Section 5.1 in [47] and Section 7.2.2 in [48], in particular, for results on a one-dimensional Burgers' equation in a diffusion-dominated regime). Note that the above frameworks typically perform on fixed spatial domains, under given boundary conditions. In this context, a framework to address transfer across domains and boundary conditions was proposed in [49]. Finally, we discuss algorithmic issues for online computations and propose a strategy for vectorized implicit time integration.

This paper is organized as follows. The formulation is first presented in Section 2. We introduce the initial boundary value problem (IBVP) in a generic form. We then review commonly employed autoencoders and derive the architecture for a convolutional neural network-based autoencoder with adaptive kernels. Next, we present operator learning strategies in the latent space, including the operator inference approach recently proposed in [50] and the DNN-based approach. We also provide an algorithm for enhanced implicit time integration. Application to a 2D Burgers' equation in an advection-dominated regime is subsequently presented in Section 4. Convergence in training cost and accuracy for both the autoencoders and operator inference is specifically discussed. The performance of the vectorized implicit time integration is also assessed on various GPUs. Concluding remarks are finally provided in Section 5.

1.2. Nomenclature

We use the following nomenclature/abbreviation for various autoencoders throughout the paper:

- LAE: linear autoencoder
- POD: proper orthogonal decomposition
- NAE: nonlinear autoencoder
- DAE: nonlinear deep autoencoder
- LE-DAE: nonlinear deep autoencoder with linear encoder
- NE-DAE: nonlinear deep autoencoder with nonlinear encoder
- SAE: nonlinear sparse autoencoder
- LE-SAE: nonlinear sparse autoencoder with linear encoder
- NE-SAE: nonlinear sparse autoencoder with nonlinear decoder
- CNNAE: nonlinear convolutional autoencoder
- LE-CNNAE: nonlinear convolutional autoencoder with linear encoder
- NE-CNNAE: nonlinear convolutional autoencoder with nonlinear encoder

The above autoencoders form a hierarchy that is depicted in 1. Two non-intrusive operator inference techniques are used to solve the reduced-order formulation in this work, namely

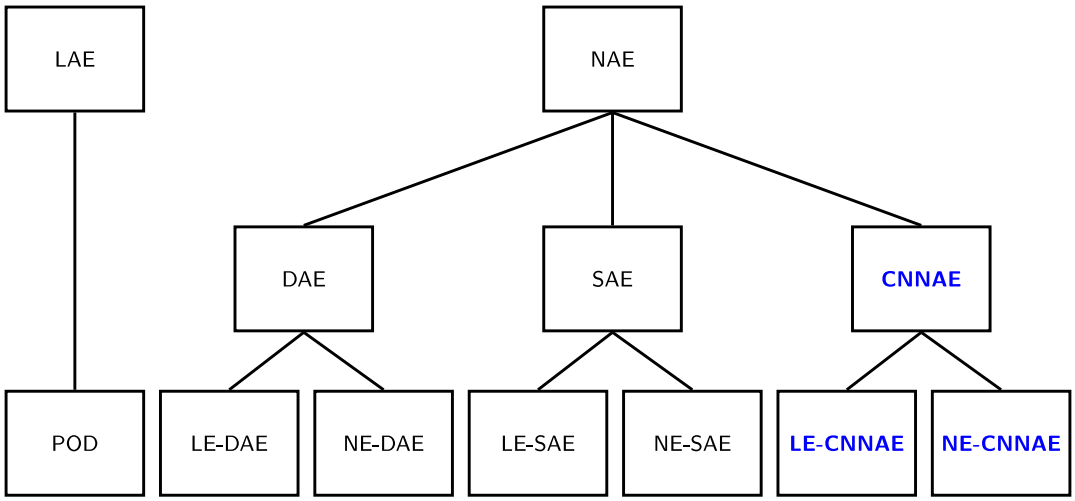


Fig. 1. The figure shows the hierarchy of several autoencoders. This paper contributes to the development of CNNAE that achieve both speedup and accuracy with the NM-ROM (following the path highlighted in blue). Throughout the paper, we will compare the performance of LAE and NAEs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
List of main symbols.

Symbol	Description
Ω	Bounded domain in $\mathbb{R}^d$ with smooth boundary $\partial\Omega$
u	d-dimensional physical state
u_0	Initial condition, $u_0(\cdot) = u(\cdot, t_0)$
f	Spatial differential operator
$\mathcal{U}$ and $\mathcal{U}_0$	Function spaces for u and u_0
$\mathcal{U}^r$ and $\mathcal{U}_0^r$	Reduced latent spaces associated with $\mathcal{U}$ and $\mathcal{U}_0$
$\mathcal{G}^\dagger$	Flow map operator from $\mathcal{U}_0$ to $\mathcal{U}$
$\mathcal{H}$	Flow map operator from $\mathcal{U}_0^r$ to $\mathcal{U}^r$
E	Encoder from $\mathcal{U}$ to $\mathcal{U}^r$
D	Decoder from $\mathcal{U}^r$ to $\mathcal{U}$
$\mathbf{u}^{\langle m\rangle}(u_0^{(\ell)})$	Discretized solution vector for initial condition $u_0^{(\ell)}$ at mth time step
$\hat{\mathbf{u}}^{\langle m\rangle}(u_0^{(\ell)})$	Latent solution vector for initial condition $u_0^{(\ell)}$ at mth time step
$\tilde{\mathbf{u}}^{\langle m\rangle}(u_0^{(\ell)})$	Approximated solution vector for initial condition $u_0^{(\ell)}$ at mth time step
$\dot{\hat{\mathbf{u}}}^{\langle m\rangle}(u_0^{(\ell)})$	Time derivative of the latent solution vector $\hat{\mathbf{u}}^{\langle m\rangle}(u_0^{(\ell)})$

- OpInf (quadratic operator inference) and
- DNNOp (deep-neural-network-based operator learning).

Autoencoders were implemented in JAX, while operator learning techniques were developed in PyTorch. The main symbols used in this paper are provided in Table 1.

2. Theory

2.1. Problem statement

We consider the initial boundary value problem

$$\frac{\partial u}{\partial t} = f(u) \quad \text{in } \Omega \times (t_0, t_f], \tag{1a}$$

$$u(x, t_0) = u_0(x) \quad \text{on } \Omega, \tag{1b}$$

where $u : \Omega \times (0, t_f] \to \mathbb{R}^d$ is the unknown d-dimensional physical state, f is a spatial differential operator, Ω is a bounded domain in $\mathbb{R}^d$ with smooth boundary $\partial\Omega$, $(0, t_f]$ denotes the time interval of interest, and $u_0 : \Omega \to \mathbb{R}^d$ is the initial condition. The solution u and initial condition u_0 are assumed to belong to *ad hoc* (separable Hilbert) function spaces, denoted by $\mathcal{U}$ and $\mathcal{U}_0$, respectively. Eq. (1) is referred to as the full-order model (FOM).

In this work, we seek to approximate the flow map operator $\mathcal{G}^\dagger : \mathcal{U}_0 \to \mathcal{U}$ mapping the initial condition to the solution at time t. To this end, we introduce *spatial* encoders for $\mathcal{U}_0$ and $\mathcal{U}$, denoted by E_0 and E, respectively. Similarly, D_0 and D are the associated

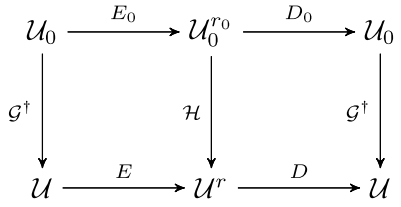


Fig. 2. Mappings used to approximate the flow map $u_0 \mapsto u(\cdot, t)$, $t > 0$ (inspired by [51]).

spatial decoders such that $D_0 \circ E_0 \approx I_0$ and $D \circ E \approx I$, where I_0 and I are the identity operators in $\mathcal{U}_0$ and $\mathcal{U}$, respectively. We further introduce the *operator* $\mathcal{H}$ approximating the flow map between the latent spaces $\mathcal{U}_0^{r_0}$ and $\mathcal{U}^r$ (in time domain). The construction of the surrogate operator then proceeds with the definition of E_0, D, and $\mathcal{H}$ such that $\mathcal{G} = D \circ \mathcal{H} \circ E_0$ is an accurate approximation to $\mathcal{G}^\dagger$; see Fig. 2. The definition of the encoder E_0 and decoder D is first addressed in Section 2.2. A *non-exhaustive* list of possible strategies for operator learning is then presented in Section 2.3.

2.2. Autoencoders

Consider a spatial discretization of Ω, defined by a collection of points $\{\mathbf{x}_i\}_{i=1}^{N_s}$, and a time discretization $\{t_m = m\Delta t\}_{m=0}^{N_t}$ of the time interval $[0, t_f]$, with Δt the step size and $t_f = N_t \Delta t$. For a given initial condition $u_0^{(\ell)}$, $1 \leqslant \ell \leqslant N_{ic}$ and $1 \leqslant m \leqslant N_t$, let

$$\mathbf{u}^{\langle 0 \rangle}(u_0^{(\ell)}) = (u_0(\mathbf{x}_1; u_0^{(\ell)})^T, \dots, u_0(\mathbf{x}_{N_s}; u_0^{(\ell)})^T)^T$$

and denote by

$$\mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) = (u(\mathbf{x}_1, t_m; u_0^{(\ell)})^T, \dots, u(\mathbf{x}_{N_s}, t_m; u_0^{(\ell)})^T)^T$$

the vector in $\mathbb{R}^N$ (with $N = d N_s$) containing the solution at all points at the mth time step. Consider the mean over all time steps and initial conditions:

$$\bar{\mathbf{u}} = \frac{1}{N_{snp}} \sum_{\ell=1}^{N_{ic}} \sum_{m=0}^{N_t} \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}), \tag{2}$$

with $N_{snp} = N_{ic}(N_t + 1)$. We then introduce the matrix of solution snapshots $\mathbf{U} \in \mathbb{R}^{N \times N_{snp}}$ as

$$\mathbf{U} = [\mathbf{u}^{\langle 0 \rangle}(u_0^{(1)}) - \bar{\mathbf{u}}, \dots, \mathbf{u}^{\langle N_t \rangle}(u_0^{(1)}) - \bar{\mathbf{u}}, \dots, \mathbf{u}^{\langle 0 \rangle}(u_0^{(N_{ic})}) - \bar{\mathbf{u}}, \dots, \mathbf{u}^{\langle N_t \rangle}(u_0^{(N_{ic})}) - \bar{\mathbf{u}}]. \tag{3}$$

It what follows, we discuss various strategies to construct E and D, and use E to encode in $\mathcal{U}_0$ (i.e., we take $E_0 = E$ and $r_0 = r$). Note that this choice is generally acceptable since the solution u typically exhibits less regularity than u_0, and becomes licit for encoding–decoding approaches involving function bases (e.g., for the POD approach in Section 2.2.1). We let $\hat{u}(t) = E(u(x,t))$ and $\tilde{u}(x,t) = D(\hat{u}(t)) = D(E(u(x,t)))$, with $\tilde{u} \approx u$, and carry this convention over all forms of representation without adjusting notation for operators, for ease of presentation (i.e., we consider $\hat{\mathbf{u}}^{\langle m \rangle} = E(\mathbf{u}^{\langle m \rangle})$ and $\tilde{\mathbf{u}}^{\langle m \rangle} = D(\hat{\mathbf{u}}^{\langle m \rangle})$ after discretization).

Classical autoencoders are reviewed below as a baseline for comparison (see Sections 2.2.1, 2.2.2, and 2.2.3), and a new decoding strategy, inspired by adaptive basis methods, is proposed (see Section 2.2.4). Note that for nonlinear reduction techniques involving neural networks, hidden layers with or without nonlinear activation functions (here, the swish activation function a defined as $a(x) = x/(1 + \exp(-x))$) are considered. Such architectures are identified with the prefixes "NE" and "LE", respectively. Activation functions are used in the hidden layers in all nonlinear decoders (except for the output layer).

2.2.1. POD autoencoder

The Proper Orthogonal Decomposition (POD) is arguably the most widely used linear approach to encode and decode in high-dimensional state spaces, due to its amenability for analysis and ease of use. In this method, the encoder $E : \mathbb{R}^N \to \mathbb{R}^r$, $r \ll N$, is given by

$$\hat{\mathbf{u}}^{\langle m \rangle} = E(\mathbf{u}^{\langle m \rangle}) = (\langle \mathbf{u}^{\langle m \rangle}, \boldsymbol{\phi}_1 \rangle_{\mathbb{R}^N}, \dots, \langle \mathbf{u}^{\langle m \rangle}, \boldsymbol{\phi}_r \rangle_{\mathbb{R}^N})^T, \tag{4}$$

where $\langle \cdot, \cdot \rangle_{\mathbb{R}^N}$ is the standard Euclidean inner product in $\mathbb{R}^N$ and $\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_r$ are the left singular vectors associated with the largest singular values in the singular value decomposition of the snapshot matrix (see Eq. (3)):

$$\mathbf{U} = \boldsymbol{\Phi} \boldsymbol{\Sigma} \mathbf{V}^T, \tag{5}$$

where $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_N] \in \mathbb{R}^{N \times N}$ and $\mathbf{V} \in \mathbb{R}^{N_{snp} \times N_{snp}}$ are real orthogonal matrices, and $\boldsymbol{\Sigma} \in \mathbb{R}^{N \times N_{snp}}$ is the matrix containing the singular values (ranked in non-increasing order) on the diagonal. The decoder $D : \mathbb{R}^r \to \mathbb{R}^N$ then defines the approximation in the span of the POD modes $\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_r$:

$$\tilde{\mathbf{u}}^{\langle m \rangle} = \sum_{i=1}^{r} \langle \mathbf{u}^{\langle m \rangle}, \boldsymbol{\phi}_i \rangle_{\mathbb{R}^N} \boldsymbol{\phi}_i. \tag{6}$$

The above formulation is optimal in the L^2 sense in terms of projection error, for a given reduced dimension r. The advantages and limitations of this method are well established. In particular, the POD-based approach can be used to significantly accelerate simulations when the solution space of the discretized system exhibits a fast decaying Kolmogorov n-width, but struggles to achieve speed-ups in highly nonlinear problems, the solutions to which have large gradients. Strategies to address the irreducibility issue can be found in [18] and the references therein, for instance.

2.2.2. Deep autoencoder

We first consider a simple Deep Autoencoder (DAE), defined through a composite mapping with fully connected layers:

$$E = E^{(n_e)} \circ E^{(n_e-1)} \circ \cdots \circ E^{(1)}, \tag{7}$$

where n_e denotes the number of layers in the encoder. As previously indicated, we denote by NE-DAE and LE-DAE the encoder with or without activation functions, respectively. In the former case, each hidden layer is defined as

$$E^{(i)}(\mathbf{x}) = a(\mathbf{W}_e^{(i)}\mathbf{x} + \mathbf{b}_e^{(i)}), \quad i < n_e, \tag{8}$$

where $\mathbf{W}_e^{(i)} \in \mathbb{R}^{p\times q}$ and $\mathbf{b}_e^{(i)} \in \mathbb{R}^p$ are the matrix of weights and vector of biases in the ith layer with input dimension q and output dimension p, respectively, and the swish activation function acts component-wise. For LE-DAE, the above equation becomes

$$E^{(i)}(\mathbf{x}) = \mathbf{W}_e^{(i)}\mathbf{x} + \mathbf{b}_e^{(i)}, \quad i < n_e, \tag{9}$$

and E can be reduced to a single-layer neural network. The decoder in both DAEs is defined as

$$D = D^{(n_d)} \circ D^{(n_d-1)} \circ \cdots \circ D^{(1)}, \tag{10}$$

where n_d is the number of layers in the decoder and

$$D^{(i)}(\mathbf{x}) = \begin{cases} \mathbf{W}_d^{(i)}x + \mathbf{b}_d^{(i)}, & i = n_d, \\ a(\mathbf{W}_d^{(i)}x + \mathbf{b}_d^{(i)}), & i < n_d. \end{cases} \tag{11}$$

The DAE is trained by minimizing the classical mean squared loss function

$$l_{\mathrm{DAE}}^{\mathrm{mse}}(\boldsymbol{\Theta}_e, \boldsymbol{\Theta}_d) = \frac{1}{N_{snp}} \sum_{\ell=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{u}^{\langle m\rangle}(u_0^{(\ell)}) - \bar{\mathbf{u}} - D(E(\mathbf{u}^{\langle m\rangle}(u_0^{(\ell)}) - \bar{\mathbf{u}}; \boldsymbol{\Theta}_e); \boldsymbol{\Theta}_d) \right\|^2, \tag{12}$$

where $\boldsymbol{\Theta}_e$ and $\boldsymbol{\Theta}_d$ contain all trainable parameters in the encoder and decoder, respectively.

2.2.3. Sparse autoencoder

One approach to reduce the computational cost and increase the efficiency of the autoencoder is to use sparse neural networks—a type of neural network that utilizes only a fraction of the available connections between neurons. By reducing the number of parameters needed to train a model, such models can lead to faster convergence rate and reduced memory requirements. In addition, sparse neural networks can improve model interpretability by highlighting the most relevant features and reducing the noise caused by irrelevant connections. These benefits make sparse neural networks an attractive option for applications with limited computational resources and/or where model interpretability is important.

In [30], a Sparse Autoencoder (SAE) with a special sparsity pattern in the nonlinear decoder was proposed. Such a sparsity pattern mimics that of a diffusion operator in numerical discretization methods, such as the finite difference method (FDM). A masked autoencoder is introduced by adding a so-called mask matrix $\mathbf{S}$ which contains either zero or one to create a sparsely connected layer for the decoder:

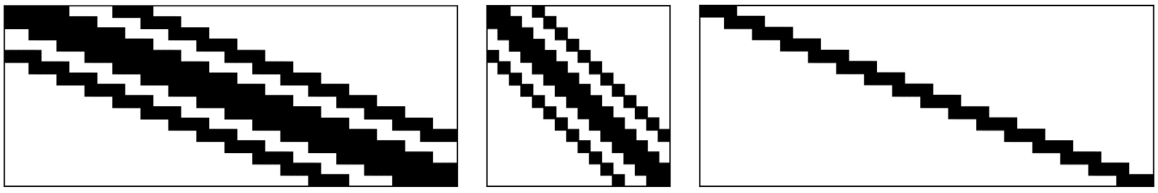
$$\mathbf{S} = \mathrm{heaviside}(\mathbf{CB}), \quad \mathbf{B} = \sum_i \sum_j b_{ij}\mathbf{e}_i\mathbf{e}_j, \quad b_{ij} = \begin{cases} 1, & i\delta \leqslant j \leqslant i\delta + b, \\ 0, & \text{otherwise}, \end{cases} \tag{13}$$

where $\mathbf{C} \in \mathbb{R}^{N\times N}$ is the nodal connectivity matrix (e.g., in the FDM), $\{\mathbf{e}_i\}_{i=1}^N$ are one-hot vectors in $\mathbb{R}^N$, b is the bandwidth of the sparsity, and $\delta = \lfloor (M-b)/(N-1) \rfloor$ is the average shift per neuron, where M is the size of the last hidden layer in the sparse decoder. A sparse weight matrix is then obtained using the Hadamard (element-wise) product:

$$\mathbf{W}^{(n_d)} = \mathbf{W}_{\mathrm{dense}}^{(n_d)} \odot \mathbf{S}, \tag{14}$$

so that the mask is only applied at the last hidden layer of the decoder. Fig. 3 shows an example of the mask matrix $\mathbf{S}$, the nodal connectivity matrix $\mathbf{C}$, and the base matrix $\mathbf{B}$ for a 2D problem with $N = 16$, $M = 81$, $b = 6$, and $\delta b = 5$.

Following our convention, sparse autoencoders with or without activation functions in the encoder are denoted by NE-SAE and LE-SAE, respectively.



(a) **S** (b) **C** (c) **B**

Fig. 3. Mask matrix **S**, nodal connectivity matrix **C**, and base matrix **B** for $N = 16$, $M = 81$, $b = 6$, and $\delta b = 5$.

2.2.4. Convolutional autoencoder with enhanced adaptivity

In this section, we propose to utilize Convolutional Neural Networks (CNNs) in the decoder to promote adaptivity. This choice is motivated by the fact that for PDEs with sharp solution gradients, the localization induced by convolutional filters helps extract local information from neighboring points, hence enabling high-quality reconstruction from the low-dimensional latent space $\mathcal{U}^r$. Our aim is to enhance adaptivity by considering trainable kernels in the convolution operator.

We define the decoder as

$$D(\hat{\mathbf{u}}^{\langle m\rangle}) = \sum_{i=1}^{r} D_i(\hat{u}_i^{\langle m\rangle}; \hat{\mathbf{u}}^{\langle m\rangle}), \tag{15}$$

where $D_i : \mathbb{R} \to \mathbb{R}^N$ is a lifting function defined as

$$D_i(\hat{u}_i^{\langle m\rangle}; \hat{\mathbf{u}}^{\langle m\rangle}) = (P \circ C_{\kappa_i}(\cdot\,; \hat{\mathbf{u}}^{\langle m\rangle}) \circ R \circ S_i)(\hat{u}_i^{\langle m\rangle}), \tag{16}$$

where each term in the right-hand side is defined below.

- The function S_i is lifting $\hat{u}_i^{\langle m\rangle}$ to a vector in $\mathbb{R}^N$ through the learnable mapping

$$S_i(\hat{u}_i^{\langle m\rangle}) = \mathbf{W}_i \hat{u}_i^{\langle m\rangle} + \mathbf{b}_i, \tag{17}$$

  where $\mathbf{W}_i \in \mathbb{R}^N$ and $\mathbf{b}_i \in \mathbb{R}^N$ are weights and bias of the fully connected layer.
- R reshapes a vector in $\mathbb{R}^N$ into a matrix in $\mathbb{R}^{n_1 \times \cdots \times n_d}$, where n_i specifies the number of degrees of freedom along the ith direction in the physical domain.
- The convolution function $C_{\kappa_i} : \mathbb{R}^{n_1 \times \cdots \times n_d} \to \mathbb{R}^{n_1 \times \cdots \times n_d}$ introduces adaptive spatial smoothing to capture localization. To define C_{κ_i}, we first introduce the kernel $\kappa_i : \mathbb{R}^r \to \mathbb{R}^{\mu^d}$ through the composite mapping

$$\kappa_i = R_\kappa \circ (K_i^{(n_k)} \circ K_i^{(n_k-1)} \circ \cdots \circ K_i^{(1)}), \tag{18}$$

  where $K_i^{(n_k)} \circ K_i^{(n_k-1)} \circ \cdots \circ K_i^{(1)}$ is a neural network with $(n_k - 1)$ hidden layers (with activation functions), i.e.,

$$K_i^{(\ell)}(\mathbf{x}) = \begin{cases} \mathbf{W}_i^{(\ell)}\mathbf{x} + \mathbf{b}_i^{(\ell)}, & \mathbf{W}_i^{(\ell)} \in \mathbb{R}^{\mu^d \times q_{\ell-1}}, \quad \mathbf{b}_i^{(\ell)} \in \mathbb{R}^{\mu^d}, \quad \ell = n_k, \\ a(\mathbf{W}_i^{(\ell)}\mathbf{x} + \mathbf{b}_i^{(\ell)}), & \mathbf{W}_i^{(\ell)} \in \mathbb{R}^{p_\ell \times q_\ell}, \quad \mathbf{b}_i^{(\ell)} \in \mathbb{R}^{p_\ell}, \quad \ell < n_k, \end{cases} \tag{19}$$

  where $q_1 = r$ and the sets $\{p_\ell\}_{\ell=1}^{n_k-1}$ and $\{q_\ell\}_{\ell=2}^{n_k-1}$ are user-specified. The function $R_\kappa \in \mathbb{R}^{\mu^d} \mapsto \mathbb{R}^{\bigtimes_{j=1}^{d} \mu}$ reshapes a vector into a matrix of size $\mu \times \cdots \times \mu$ (d times). The function C_{κ_i} then performs the discrete linear convolution of $(R \circ S_i)(\hat{u}_i^{\langle m\rangle})$ with $\kappa_i(\hat{\mathbf{u}}^{\langle m\rangle})$.
- P projects back from a matrix representation in $\mathbb{R}^{n_1 \times \cdots \times n_d}$ into a vector in $\mathbb{R}^N$.

In the above, and in contrast with strategies where the CNN is trained once for all, the trainable parameters in the kernel of the CNN are defined as the output of an auxiliary neural network and are, by construction, iteration-dependent: they evolve as a function of the latent solution vector to capture non-smoothness, hence promoting adaptivity.

The structure of the decoder D is depicted in Fig. 4. This convolutional autoencoder (denoted by CNNAE, with the LE-CNNAE and NE-CNNAE variations depending on whether activation functions are used in the encoder) is also trained by minimizing the mean squared loss given by Eq. (12).

2.3. Strategies for operator learning between latent spaces

We now turn to the construction of the surrogate operator $\mathcal{H} : \mathcal{U}_0^r \to \mathcal{U}^r$ mapping $\hat{\mathbf{u}}_0 = E(u_0)$ to $\hat{\mathbf{u}} = E(\mathbf{u})$. To this end, we consider the reduced-order model

$$\frac{\partial \hat{\mathbf{u}}(t)}{\partial t} = F(\hat{\mathbf{u}}(t)), \quad \hat{\mathbf{u}}(0) = \hat{\mathbf{u}}_0, \tag{20}$$

where the operator F in the right-hand side is learned through an *ad hoc* technique. We restrict the discussion below to two classes of techniques. The first class involves so-called quadratic operator inference [50], which is an efficient operator learning method

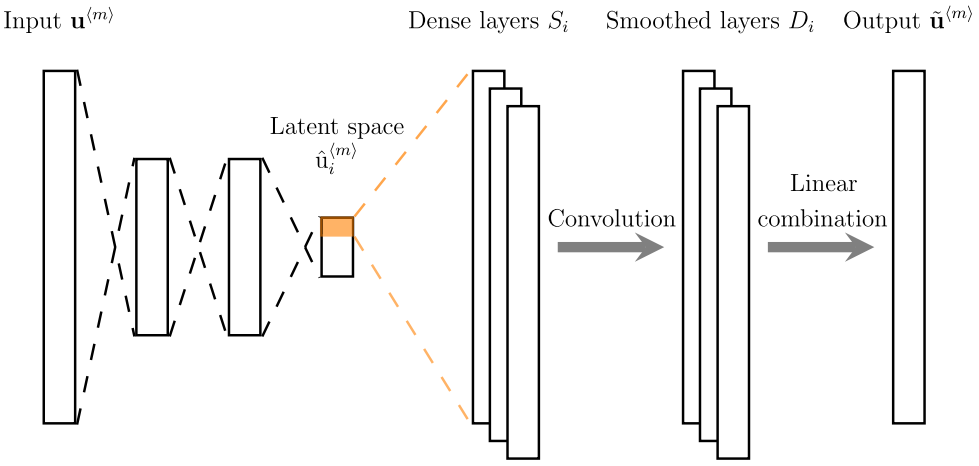


Fig. 4. General description of the CNN-based autoencoder: $\mathbf{u}^{\langle m\rangle}$ is encoded into a latent vector $\hat{\mathbf{u}}^{\langle m\rangle}$ by the encoder. The latent vector is then decoded by the decoder to get $\tilde{\mathbf{u}}^{\langle m\rangle}$. A smoothing convolutional layer is added after the single dense layer at the beginning of the decoder to introduce nonlinearity into the decoder.

that does not necessitate the definition and training of a deep learning model. This approach is briefly introduced in Section 2.3.1. The second approach involves a simple feedforward neural network model, detailed in Section 2.3.2. Regarding time integration, we rely on the implicit backward Euler method, combined with a standard Newton–Raphson solver.

2.3.1. Quadratic operator inference

Following [50], the operator inference (OpInf) approach approximates a reduced operator of the polynomial form (with linear and quadratic reduced operators). It then casts PDEs with more general nonlinear terms through the use of lifting variable transformations which expose quadratic structure in the PDE.

OpInf seeks a linear operator $\mathbf{A}$ and a quadratic operator $\mathbf{H}$ by minimizing the approximation error in the least-squares sense

$$\min_{\mathbf{A}\in\mathbb{R}^{r\times r},\ \mathbf{H}\in\mathbb{R}^{r\times r^2}} \frac{1}{N_{snp}} \sum_{\ell=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{A}\hat{\mathbf{u}}^{\langle m\rangle}(u_0^{(\ell)}) + \mathbf{H}\left(\hat{\mathbf{u}}^{\langle m\rangle}(u_0^{(\ell)}) \otimes \hat{\mathbf{u}}^{\langle m\rangle}(u_0^{(\ell)})\right) - \dot{\hat{\mathbf{u}}}^{\langle m\rangle}(u_0^{(\ell)}) \right\|^2, \tag{21}$$

with the corresponding matrix algebraic form

$$\mathbf{D}\begin{bmatrix}\mathbf{A}^T\\ \mathbf{H}^T\end{bmatrix} = \dot{\hat{\mathbf{U}}}, \tag{22}$$

where the least-squares data matrix $D \in \mathbb{R}^{r\times(r+r^2)}$ and the right-hand side $\dot{\hat{\mathbf{U}}} \in \mathbb{R}^{r\times N_{snp}}$ are given by

$$\mathbf{D} = \begin{bmatrix} \hat{\mathbf{u}}^{\langle 0\rangle}(u_0^{(1)}) & \dots & \hat{\mathbf{u}}^{\langle N_t\rangle}(u_0^{(1)}) & \dots & \hat{\mathbf{u}}^{\langle N_t\rangle}(u_0^{(N_{ic})}) \\ \hat{\mathbf{u}}^{\langle 0\rangle}(u_0^{(1)}) \otimes \hat{\mathbf{u}}^{\langle 0\rangle}(u_0^{(1)}) & \dots & \hat{\mathbf{u}}^{\langle N_t\rangle}(u_0^{(1)}) \otimes \hat{\mathbf{u}}^{\langle N_t\rangle}(u_0^{(1)}) & \dots & \hat{\mathbf{u}}^{\langle N_t\rangle}(u_0^{(N_{ic})}) \otimes \hat{\mathbf{u}}^{\langle N_t\rangle}(u_0^{(N_{ic})}) \end{bmatrix}^T \tag{23}$$

and

$$\dot{\hat{\mathbf{U}}} = \left[\dot{\hat{\mathbf{u}}}^{\langle 0\rangle}(u_0^{(1)}) \quad \dots \quad \dot{\hat{\mathbf{u}}}^{\langle N_t\rangle}(u_0^{(1)}) \quad \dots \quad \dot{\hat{\mathbf{u}}}^{\langle N_t\rangle}(u_0^{(N_{ic})})\right], \tag{24}$$

respectively. The reduced operator is then given by

$$F(\hat{\mathbf{u}}^{\langle m\rangle}) = \mathbf{A}\hat{\mathbf{u}}^{\langle m\rangle} + \mathbf{H}\left(\hat{\mathbf{u}}^{\langle m\rangle} \otimes \hat{\mathbf{u}}^{\langle m\rangle}\right). \tag{25}$$

The combination of the above operator with the time integration scheme defines the surrogate operator $\mathcal{H}$.

2.3.2. Deep neural network-based operator

In addition to the above OpInf framework, we also explore the use of a simple deep neural network to approximate the reduced operator F. The DNN-based operator (DNNOp) is defined as

$$F(\hat{\mathbf{u}}^{\langle m\rangle}) = O(\hat{\mathbf{u}}^{\langle m\rangle}), \tag{26}$$

where $O : \mathbb{R}^r \to \mathbb{R}^r$ is a deep neural network

$$O = O^{(n_o)} \circ O^{(n_o-1)} \circ \cdots \circ O^{(1)}, \tag{27}$$

with $(n_o - 1)$ hidden layers (with activation functions), i.e.,

$$O^{(\ell)}(\mathbf{x}) = \begin{cases} \mathbf{W}^{(\ell)}\mathbf{x} + \mathbf{b}^{(\ell)}, & \mathbf{W}^{(\ell)} \in \mathbb{R}^{r \times q_{\ell-1}}, \quad \mathbf{b}^{(\ell)} \in \mathbb{R}^{r}, \quad \ell = n_o, \\ a(\mathbf{W}^{(\ell)}\mathbf{x} + \mathbf{b}^{(\ell)}), & \mathbf{W}^{(\ell)} \in \mathbb{R}^{p_\ell \times q_\ell}, \quad \mathbf{b}_i^{(\ell)} \in \mathbb{R}^{p_\ell}, \quad \ell < n_o, \end{cases} \tag{28}$$

with $q_1 = r$. The dimensions in the hidden layers $\{p_\ell\}_{\ell=1}^{n_k-1}$ and $\{q_\ell\}_{\ell=2}^{n_k-1}$ are chosen while devising the architecture.

The operator is trained by minimizing the projection error defined as

$$l^{\text{proj}}_{\text{DNNOp}}(\hat{\mathbf{U}}; \boldsymbol{\Theta}_o) = \frac{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \dot{\hat{\mathbf{u}}}^{\langle m \rangle}(u_0^{(\ell)}) - \mathrm{O}(\hat{\mathbf{u}}^{\langle m \rangle}(u_0^{(\ell)}); \boldsymbol{\Theta}_o) \right\|^2}}{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \dot{\hat{\mathbf{u}}}^{\langle m \rangle}(u_0^{(\ell)}) \right\|^2}}, \tag{29}$$

where $\boldsymbol{\Theta}_o$ contains all trainable parameters in the reduced operator. The use of the approximation in Eq. (26) with the time integrator defines the DNNOp variant for $\mathcal{H}$.

3. Vectorized implicit time integration

Ultimately, the surrogate of the flow map operator is applied to find solutions to the FOM at various times with different initial conditions. To that end, numerical time integration methods are used. Many of the physical systems with sharp gradients (such as the Burger's equation considered in this work) are "stiff" and numerically unstable. If an explicit time integration method is used, a prohibitively small time step size is often required to obtain an accurate solution. Furthermore, due to intrinsic dependence on time step size (as illustrated in Section 4.4), the same time step sizes used in the offline training phase are preferred. Given these considerations, all online evaluations in this work are performed using the implicit backward Euler time integration.

On the other hand, since General-Purpose Graphics Processing Unit (GPGPU)-enabled computing devices are used in the offline training process, it is preferable to take advantage of the hardware acceleration also in the online evaluation phase. However, a conundrum arises:

- The discretized surrogate of the map operator is by definition *small*, i.e. the solution vector in the latent space is of size $r \ll N$, and the linearized system is of size $r \times r$, and the solution at time step m depends on the solution from $m - 1$.
- Modern GPGPU-enabled computing devices rely on being able to handle a *large* amount of workload in a vectorized fashion.

In other words, the effective and accurate surrogate operator will unfortunately starve the computing device for work. To address this issue, we propose a new technique to effectively "vectorize" the time integration in parallel, in order to increase the workload, i.e. from a system of size $r \times r$ to a system of size $m_c \times r \times r$, where $m_c \leqslant m$ is the number of "chunked" time steps to be explained momentarily. The basic idea/algorithm is to assemble m_c linearized systems in a vectorized fashion, form an implicit system of size $m_c n \times m_c n$, and then solve the system of equations again in a vectorized fashion by exploiting its special block-bidiagonal structure. Each step of the algorithm is described in the following paragraphs.

Increasing the size of the linearized system. Consider a "chunk" of m_c time steps integrating the surrogate flow map from time t_i to time t_{i+m_c}. Recast the solution at each of these time steps in an incremental form as

$$\hat{\mathbf{u}}_{i+j} = \hat{\mathbf{u}}_i + \Delta\hat{\mathbf{u}}_j. \tag{30}$$

That is, the solution $\hat{\mathbf{u}}_{i+j}$ at a given time is equal to the solution $\hat{\mathbf{u}}_i$ at the start of the chunk of time steps plus an increment $\Delta\hat{\mathbf{u}}_j$. With this rearrangement, we can write the implicit time integration equations for the entire chunk of m_c steps as

$$\begin{bmatrix} F\left(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_i + \Delta\hat{\mathbf{u}}_1, t_{i+1}, t_i\right) \\ F\left(\hat{\mathbf{u}}_i + \Delta\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_i + \Delta\hat{\mathbf{u}}_2, t_{i+2}, t_{i+1}\right) \\ \vdots \\ F\left(\hat{\mathbf{u}}_i + \Delta\hat{\mathbf{u}}_j, \hat{\mathbf{u}}_i + \Delta\hat{\mathbf{u}}_{j-1}, t_{i+j}, t_{i+j-1}\right) \end{bmatrix} = \mathbf{0}. \tag{31}$$

Or, specifically for the backward Euler scheme[1]:

$$\begin{bmatrix} \Delta\hat{\mathbf{u}}_1 - h\left(\hat{\mathbf{u}}_i + \Delta\hat{\mathbf{u}}_1, t_{i+1}; p\right) \Delta t_{i+1} \\ \Delta\hat{\mathbf{u}}_2 - \Delta\hat{\mathbf{u}}_1 - h\left(\hat{\mathbf{u}}_i + \Delta\hat{\mathbf{u}}_2, t_{i+2}; p\right) \Delta t_{i+2} \\ \vdots \\ \Delta\hat{\mathbf{u}}_j - \Delta\hat{\mathbf{u}}_{j-1} - h\left(\hat{\mathbf{u}}_i + \Delta\hat{\mathbf{u}}_j, t_{i+j}; p\right) \Delta t_{i+j} \end{bmatrix} = \mathbf{0}. \tag{32}$$

In the simple, serial time integration scheme the nonlinear system has dimension r. The nonlinear system for this vectorized time integration scheme (Eq. (32)) has dimension $m_c \times r$. Clearly this approach can fully consume the available bandwidth of the computing

[1] The algorithm is derived based on the backward Euler scheme but can be trivially generalized to other (higher order) variants.

device with a sufficiently large m_c. Eq. (32) is still nonlinear. Applying the Newton–Raphson method provides the update algorithm

$$\begin{bmatrix} {}_{k+1}\Delta\hat{\mathbf{u}}_1 \\ {}_{k+1}\Delta\hat{\mathbf{u}}_2 \\ \vdots \\ {}_{k+1}\Delta\hat{\mathbf{u}}_j \end{bmatrix} = \begin{bmatrix} {}_{k}\Delta\hat{\mathbf{u}}_1 \\ {}_{k}\Delta\hat{\mathbf{u}}_2 \\ \vdots \\ {}_{k}\Delta\hat{\mathbf{u}}_j \end{bmatrix} - {}_{k}J^{-1}_{i+1} \begin{bmatrix} {}_{k}\Delta\hat{\mathbf{u}}_1 - h\left(\hat{\mathbf{u}}_i + {}_{k}\Delta\hat{\mathbf{u}}_1, t_{i+1}; p\right)\Delta t_{i+1} \\ {}_{k}\Delta\hat{\mathbf{u}}_2 - {}_{k}\Delta\hat{\mathbf{u}}_1 - h\left(\hat{\mathbf{u}}_i + {}_{k}\Delta\hat{\mathbf{u}}_2, t_{i+2}; p\right)\Delta t_{i+2} \\ \vdots \\ {}_{k}\Delta\hat{\mathbf{u}}_j - {}_{k}\Delta\hat{\mathbf{u}}_{j-1} - h\left(\hat{\mathbf{u}}_i + {}_{k}\Delta\hat{\mathbf{u}}_j, t_{i+j}; p\right)\Delta t_{i+j} \end{bmatrix} \tag{33}$$

where the chunked discrete Jacobian has a bidiagonal form

$${}_{k}J_{i+1} = \begin{bmatrix} I - {}_{k}J^{\dagger}_{i+1} & 0 & \cdots & 0 & 0 \\ -I & I - {}_{k}J^{\dagger}_{i+2} & 0 & \cdots & 0 \\ 0 & -I & I - {}_{k}J^{\dagger}_{i+3} & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & -I & I - {}_{k}J^{\dagger}_{i+j} \end{bmatrix} \tag{34}$$

where ${}_{k}J^{\dagger}_{i+j}$ is the ODE Jacobian evaluated at the current values of the state $\hat{\mathbf{u}}_i + {}_{k}\Delta\hat{\mathbf{u}}_j$, and I is the identity matrix. The key linear algebra kernel for the vectorized time integration is solving potentially batched, block-bidiagonal[2] systems of this type (Eq. (35)) to update the current Newton–Raphson iterate for the incremental solution. The following two paragraphs discuss two viable algorithms, namely Thomas's algorithm and parallel cyclic reduction (PCR), to efficiently solve systems of this type.

$$\begin{bmatrix} A_1 & & & & \\ B_1 & A_2 & & & \\ & B_2 & A_3 & & \\ & & \ddots & \ddots & \\ & & & B_{n-1} & A_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \tag{35}$$

Thomas algorithm. The Thomas algorithm sweeps down the diagonal blocks of the matrix, solving for each block of unknown solution $\hat{\mathbf{u}}_i$ one-by-one. Algorithm 1 summarizes the process. All the matrix operations in this algorithm can be batched over an additional batch dimension, e.g. for different initial conditions.

Algorithm 1 Thomas algorithm for solving batched, block-bidiagonal matrix systems.

$x_1 = A_1^{-1} y_1$
$i \leftarrow 1$
while $i < m_c$ **do**
  $x_{i+1} \leftarrow A_{i+1}^{-1}\left(y_{i+1} - B_i x_i\right)$
  $i \leftarrow i + 1$
end while

Parallel cyclic reduction (PCR). Parallel cyclic reduction is a divide-and-conquer algorithm that recursively splits a (batched, blocked) bidiagonal matrix into two independent bidiagonal linear systems. The following equations give the basic update, splitting a single bidiagonal system

$$\begin{bmatrix} A_1 & 0 & 0 & 0 \\ B_1 & A_2 & 0 & 0 \\ 0 & B_2 & A_3 & 0 \\ 0 & 0 & B_3 & A_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \tag{36}$$

into two independent bidiagonal systems

$$\begin{bmatrix} A_1 & 0 & 0 & 0 \\ 0 & A_2 & 0 & 0 \\ -B_2 A_2^{-1} B_1 & 0 & A_3 & 0 \\ 0 & -B_3 A_3^{-1} B_2 & 0 & A_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 - B_1 A_1^{-1} y_1 \\ y_3 - B_2 A_2^{-1} y_2 \\ y_4 - B_3 A_3^{-1} y_3 \end{bmatrix} \tag{37}$$

where the first subsystem corresponds to (x_1, x_3), and the second subsystem corresponds to (x_2, x_4). PCR then recursively applies this splitting formula to further subdivide the matrices until it reaches a block diagonal form. Algorithm 2 describes the complete process. The key point in PCR is that the new matrices produced by each application of the splitting formula can be factored independently, in parallel.

[2] We note that the system is lower-block-bidiagonal. However, in the case of integrating the surrogate flow operator "backward" in time for solving an adjoint problem, the system is upper-block-bidiagonal. Both Thomas's algorithm and PCR work in these two cases.

Algorithm 2 Parallel cyclic reduction algorithm for solving batched, block-bidiagonal matrix systems.

$i \leftarrow 1$
while $i < \log_2 m_c$ **do**
  Form 2^i submatrices by taking every 2^ith row
  for submatrix $j = 1$ to $j = 2^i$ **do** ▷ This loop can be vectorized
    for index $k \in$ submatrix j **do**
      $B_k \leftarrow -B_k D_k^{-1} B_{k-1}$
      $y_k \leftarrow y_k - B_k D_{k-1}^{-1} y_{k-1}$
    end for
  end for
  $i \leftarrow i + 1$
end while
return $x \leftarrow y$

Remark. Naturally, a third option could be to start solving the system with PCR, but halt the recursive subdivision before reducing the system to a series of diagonal blocks and instead solve the remaining, unreduced sets of independent equations using Thomas algorithm. This approach has an extra parameter controlling the heuristic, the number of iterations after which to switch from PCR to Thomas, $\bar{m}$.

Time complexities of vectorized time integration algorithms. Without considering parallelization and vectorization over the batched matrix operations, the Thomas algorithm is serial with time complexity $O(m_c r^3)$, including the cost of back propagation loop and the cost of factorizing the diagonal blocks. Note this is considerably cheaper than solving the full dense matrix, which would be $O(m_c^3 r^3)$. PCR, on the other hand, has time complexity $O(\log_2(m_c) m_c r^3 + m_c r^3)$. However, if the computing device can accommodate the full amount of available parallel work (m_c submatrices on the final iteration), which is a reasonable assumption for a GPU and the size of the discretized flow map surrogate, the time complexity becomes $O(m_c r^3)$. The two algorithms therefore have the same asymptotic performance dominated by the cost of factorizing the diagonal blocks, assuming perfect vectorization of PCR. However, in practice, as explored in the timing studies, the two algorithms are competitive depending on the size of the system (defined by m_c and r, as suggested by the differing complexity of the reduction/back-propagation part of the analysis).

4. Application to 2D Burgers' equation

4.1. Description of the problem

Consider the parameterized 2D viscous Burgers' equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{38a}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right), \tag{38b}$$

$$(x, y) \in \Omega = [0, 1] \times [0, 1], \quad t \in [0, 2], \tag{38c}$$

where $u, v : \Omega \times T \mapsto \mathbb{R}$ are scalar-valued time-dependent velocities in the x and y directions, respectively, and Re is the Reynolds number. In this work, we consider $Re = 10000$ to enforce sharp gradients in the solution. We note that standard values used to benchmark learning methods are in the range $Re \in [10, 5000]$, with a larger representation of low Reynolds numbers (e.g., $Re = 100$ in [47,48]). The initial conditions are given as

$$u(x, y, 0; \mu) = \begin{cases} \mu \sin(2\pi x)\sin(2\pi y), & \text{if } (x, y) \in [0, 0.5] \times [0, 0.5], \\ 0, & \text{otherwise}, \end{cases} \tag{39a}$$

$$v(x, y, 0; \mu) = \begin{cases} \mu \sin(2\pi x)\sin(2\pi y), & \text{if } (x, y) \in [0, 0.5] \times [0, 0.5], \\ 0, & \text{otherwise}, \end{cases} \tag{39b}$$

with zero boundary conditions on the edges. The spatial domain Ω is discretized into $(n_x - 1)$ and $(n_y - 1)$ uniform meshes with linear quadrilateral elements in x and y directions, respectively, with $n_x = 60$, $n_y = 60$. The number of uniform time steps $n_t = 1500$.

To enable fair comparison and reproducibility, the solution snapshots used by the authors in [30] were used for $\mu = \{0.9, 0.95, 1.05, 1.1\}$. A separate dataset obtained with $\mu = 1.0$ was used for testing the generalization ability of the models (test dataset).

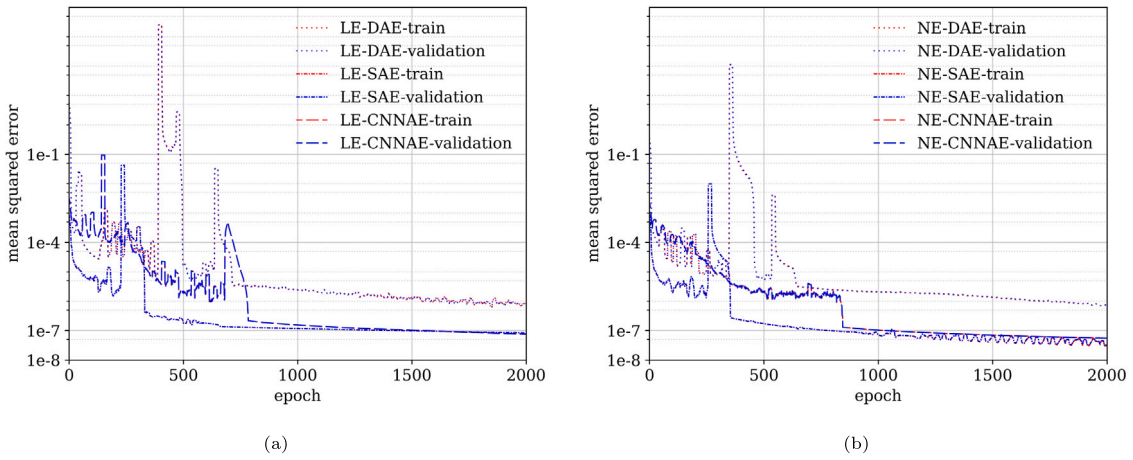


Fig. 5. Loss (training and validation) history ($r = 20$) for the deep autoencoder (DAE), the sparse autoencoder (SAE), and the proposed convolutional autoencoder (CNNAE) with linear (a) and nonlinear encoder (b). Savitzky–Golay filter is applied to the loss history for better visualization, with filter window length equals to 15, the order of the polynomial used to fit the loss is 1, and the extension contains the nearest input value.

4.2. Training of autoencoders

Autoencoders are trained using 90% of the reference solution from the training dataset, while the remaining 10% data points are considered as the validation dataset. The test dataset is not used during the offline training process and is used for performance comparison between all autoencoders and reduced operators.

For the deep autoencoders (LE-DAE and NE-DAE), we use a hidden layer of size $M_1 = 6728$ in the encoder, and a hidden layer of size $M_2 = 33730$ in the decoder. For the sparse autoencoders (LE-SAE and NE-SAE), we use the same neural network structure with the shallow mask built as shown in (13), with bandwidth $b = 70$, and shift $\delta = \lfloor (M - b)/(N - 1) \rfloor = 10$. For the convolutional autoencoders (LE-CNNAE and NE-CNNAE), we use one hidden layer of size $M_e = 200$ in the encoder, and the nonlinear mapping for the smoothing kernels are obtained by a deep neural network with one hidden layer of size $M_\mu = 200$, with the size of the kernel $\mu = 5$. We use the whole training dataset to learn the bases of the POD, and evaluate it on the test dataset as well.

For the training strategies, we employ the Adam optimizer [52], which is a variant of stochastic gradient descent (SGD), with an initial learning rate set to 0.001. For the deep and sparse autoencoders, we split the training dataset into $n_{\text{batch}} = 22$ batches, and use maximum patience of 100 epochs such that the learning rate will decrease by a factor of 10 when a training loss stagnates for 100 successive training epochs. For the convolutional autoencoders, we split the training dataset into $n_{\text{batch}} = 240$ batches and use maximum patience of 100 epochs as well. We experimented model training with different batch sizes, and choose different batch sizes for the deep/sparse autoencoder and the convolutional based autoencoder based on best performances. We consider four different latent space dimensions: $r \in \{5, 10, 15, 20\}$. The train and validation loss (mean-squared error) histories for all autoencoders with $r = 20$ are shown in Fig. 5. The training and validation losses are fairly close to each other showing good balance between accuracy and overfitting.

We consider the *projection errors* of the autoencoders defined as

$$I^{\text{proj}}_{\text{LAE}}(\mathbf{U};\boldsymbol{\Phi}) = \frac{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) - \bar{\mathbf{u}} - \sum_{i=1}^{r} \langle \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) - \bar{\mathbf{u}}, \boldsymbol{\phi}_i \rangle_{\mathbb{R}^N} \boldsymbol{\phi}_i \right\|^2}}{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) \right\|^2}} \tag{40}$$

and

$$I^{\text{proj}}_{\text{NAE}}(\mathbf{U};\boldsymbol{\Theta}_e,\boldsymbol{\Theta}_d) = \frac{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) - \bar{\mathbf{u}} - D(E(\mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) - \bar{\mathbf{u}}; \boldsymbol{\Theta}_e); \boldsymbol{\Theta}_d) \right\|^2}}{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) \right\|^2}}, \tag{41}$$

for the linear and nonlinear autoencoders, respectively. The projection error for all autoencoders on the test dataset ($\mu = 1.0$) over different latent space dimensions $r \in \{5, 10, 15, 20\}$ is summarized in Fig. 6(b).

4.3. Training of reduced operators

We consider two strategies for operator learning between latent spaces. For OPInf, we solve Eq. (22) for the linear and quadratic reduced operators. For DNNOp, we use 5 hidden layers with sizes of $M_o = 50$, and the swish function as the activation function. The

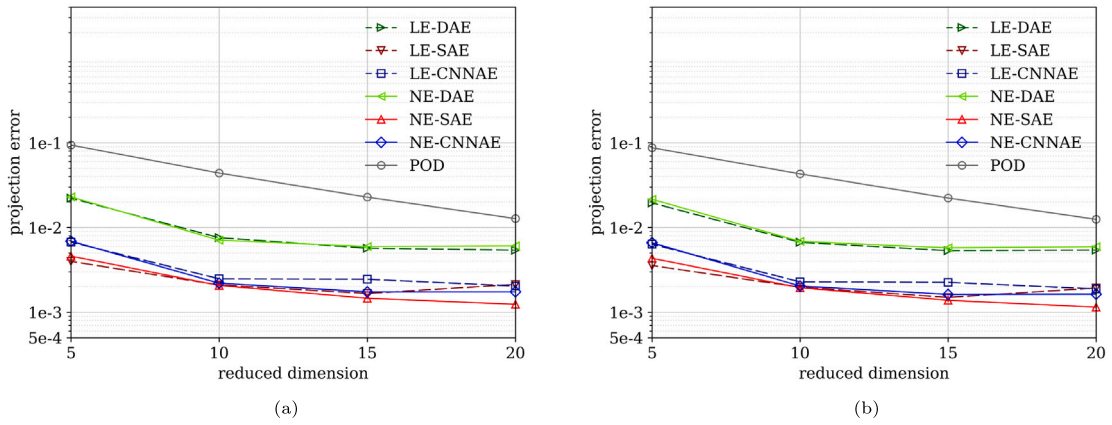


Fig. 6. (a) Projection error over different latent space dimensions on training data with $\mu = \{0.9, 0.95, 1.05, 1.1\}$; (b) Projection error over different latent space dimensions on test data with $\mu = 1.0$.

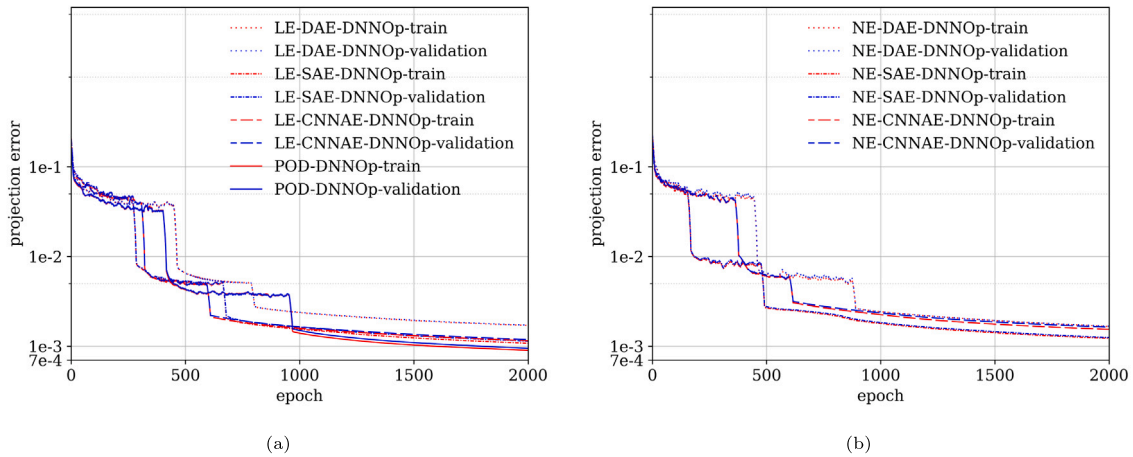


Fig. 7. Loss (training and validation) history ($r = 20$) of DNNOp paired with linear (a) and nonlinear encoders (b) as well as POD (a). Savitzky–Golay filter is applied to the loss history for better visualization, with filter window length equals to 15, the order of the polynomial used to fit the loss is 1, and the extension contains the nearest input value.

model is trained for 2000 epochs, using the Adam optimizer and an initial learning rate set to 0.01. We split the training dataset into $n_{\text{batch}} = 100$ batches, and use maximum patience of 100 epochs such that the learning rate will decrease by a factor of 10 when the maximum patience is reached. The reduced operators are trained and paired with the autoencoders given different latent space dimensions, and the training and validation loss (projection error on the rate between latent spaces) histories with r = 20 for all autoencoders paired with the reduced operator are shown in Fig. 7. Again, it is seen that the training and validation losses exhibit very similar trends and magnitudes, with a good balance between accuracy and overfitting. In Fig. 8, we summarize the projection error of the trained reduced operator given different latent space dimensions. The projection error between latent spaces of the reduced operator is defined as

$$l^{\text{proj}}_{\text{LQ}}(\hat{\mathbf{U}}; \mathbf{A}, \mathbf{H}) = \frac{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \dot{\hat{\mathbf{u}}}^{\langle m \rangle}(u_0^{(\ell)}) - \mathbf{A}\hat{\mathbf{u}}^{\langle m \rangle} - \mathbf{H}\left(\hat{\mathbf{u}}^{\langle m \rangle} \otimes \hat{\mathbf{u}}^{\langle m \rangle}\right) \right\|^2}}{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \dot{\hat{\mathbf{u}}}^{\langle m \rangle}(u_0^{(\ell)}) \right\|^2}}, \tag{42}$$

and

$$l^{\text{proj}}_{\text{DNNOp}}(\hat{\mathbf{U}}; \boldsymbol{\Theta}_o) = \frac{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \dot{\hat{\mathbf{u}}}^{\langle m \rangle}(u_0^{(\ell)}) - \mathrm{O}(\hat{\mathbf{u}}^{\langle m \rangle}(u_0^{(\ell)}); \boldsymbol{\Theta}_o) \right\|^2}}{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \dot{\hat{\mathbf{u}}}^{\langle m \rangle}(u_0^{(\ell)}) \right\|^2}}, \tag{43}$$

for the quadratic operator inference and DNN-based operator, respectively.

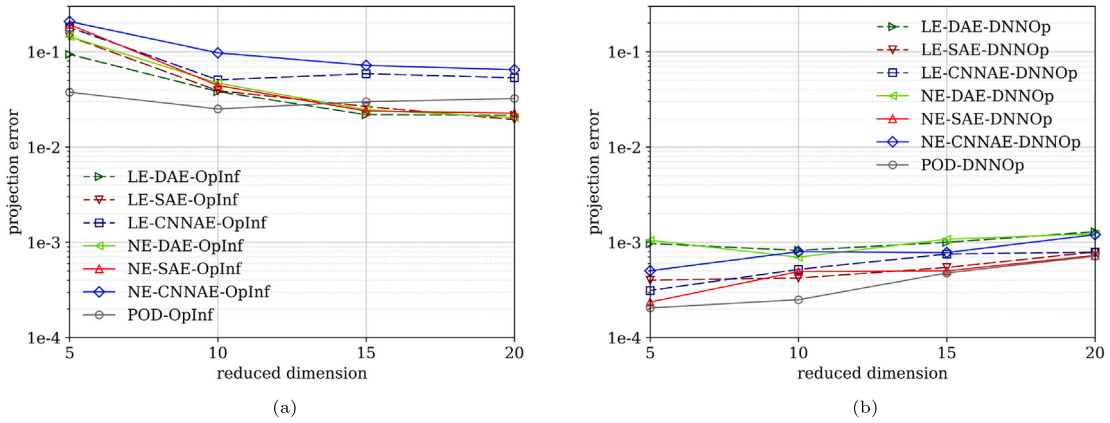


Fig. 8. Projection error of the trained reduced operator F in the latent spaces for (a) quadratic operator inference; and (b) DNN-based operator on the training dataset with $\mu = \{0.9, 0.95, 1.01, 1.1\}$.

We evaluate the reduced operators' performances between the latent spaces (i.e., the flow map $u_0 \mapsto u(\cdot, t)$, $t > 0$) on the test dataset using the errors given by

$$I_{rom}^{\text{proj}}(\hat{\mathbf{U}}; \hat{\mathbf{u}}_0, \boldsymbol{\Theta}_o) = \frac{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \hat{\mathbf{u}}^{\langle m \rangle}(u_0^{(\ell)}) - \hat{\mathbf{u}}_{rom}^{\langle m \rangle}(u_0^{(\ell)}) \right\|^2}}{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \hat{\mathbf{u}}^{\langle m \rangle}(u_0^{(\ell)}) \right\|^2}}, \tag{44}$$

where $\hat{\mathbf{u}}_{rom}^{\langle m \rangle}(u_0^{(\ell)}) = \mathcal{H}(\hat{\mathbf{u}}_0(u_0^{(\ell)}); \boldsymbol{\Theta}_o, t_m)$ is the approximated latent solution at a given time step m. We solve the reduced order model using the Newton–Raphson method. We evaluate the reduced-order model solution obtained by combining the surrogate operator $\mathcal{H}$ with any of the autoencoders with a projection error in the original high dimensional space, using the error defined as

$$I_{\text{sol}}^{\text{proj}}(\mathbf{U}; \boldsymbol{\Phi}, \mathcal{H}(\boldsymbol{\Theta}_o)) = \frac{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) - \bar{\mathbf{u}} - \sum_{i=1}^{r} \mathcal{H}(\hat{\mathbf{u}}_0(u_0^{(\ell)}); \boldsymbol{\Theta}_o, t^m) \boldsymbol{\phi}_i \right\|^2}}{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) \right\|^2}} \tag{45}$$

and

$$I_{\text{sol}}^{\text{proj}}(\mathbf{U}; \boldsymbol{\Theta}_e, \boldsymbol{\Theta}_d, \mathcal{H}(\boldsymbol{\Theta}_o)) = \frac{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) - \bar{\mathbf{u}} - D(\mathcal{H}(\hat{\mathbf{u}}_0(u_0^{(\ell)}); \boldsymbol{\Theta}_o, t^m); \boldsymbol{\Theta}_d) \right\|^2}}{\sqrt{\sum_{l=1}^{N_{ic}} \sum_{m=0}^{N_t} \left\| \mathbf{u}^{\langle m \rangle}(u_0^{(\ell)}) \right\|^2}}, \tag{46}$$

for the linear and nonlinear autoencoders, respectively, with $\hat{\mathbf{u}}_0(u_0^{(\ell)}) = E(\mathbf{u}_0(u_0^{(\ell)}); \boldsymbol{\Theta}_e)$.

In Fig. 9, we show the projection error for the reduced-order model solution over all latent dimensions with different pairs of autoencoders and operators. Note that quadratic operator inference fails to solve this 2D Burger's problem with an error blowing up for $m > 200$ approximately.

The projection error of the reduced-order model for the nonlinear autoencoders is lower than for the linear autoencoders. We summarized ROM solution and their absolute errors of the POD method in Fig. 10 for four latent space dimensions.

Both NE-SAE and NE-CNNAE have a projection error around 2×10^{-3} and deliver the most accurate predictions. Their ROM solutions and associated absolute errors are shown in Figs. 11 and 12.

The absolute errors decrease together when latent dimension increases. However, oscillations in the ROM solution of the POD method can still be observed for $r = 20$. These oscillations can also be observed from the ROM solution with the LE-DAE and NE-DAE, but are not present for NE-CNNAE and NE-SAE when $r \in \{15, 20\}$.

The adaptivity in the bases promoted in the CNNAE approach is illustrated in Fig. 13, which shows how one particular basis (here, D_4) evolves as a function of time (similar responses are obtained for the other bases). This adaptivity mechanism helps capture the features exhibited by the solution as the simulation progresses, hence facilitating interpretation.

The performance of vectorized implicit time integration was assessed by measuring the computational cost of flow map estimation for 1500 time steps, using wall-clock time and multiple GPUs—as speed up also varies based on the hardware itself. The timing is obtained by performing calculations on NVIDIA RTX A4500 (64 bits, 33 MHz, 20 GB), and NVIDIA RTX A6000 (64 bits, 33 MHz, 48 GB), NVIDIA GeForce 4080 (64 bits, 33 MHz, 16 GB). The wall-clock times are provided in Table 2. The algorithm improves performance for $r \geqslant 15$, regardless of chunk size, with a maximum speed-up of 25.48 when solving the reduced order model in the

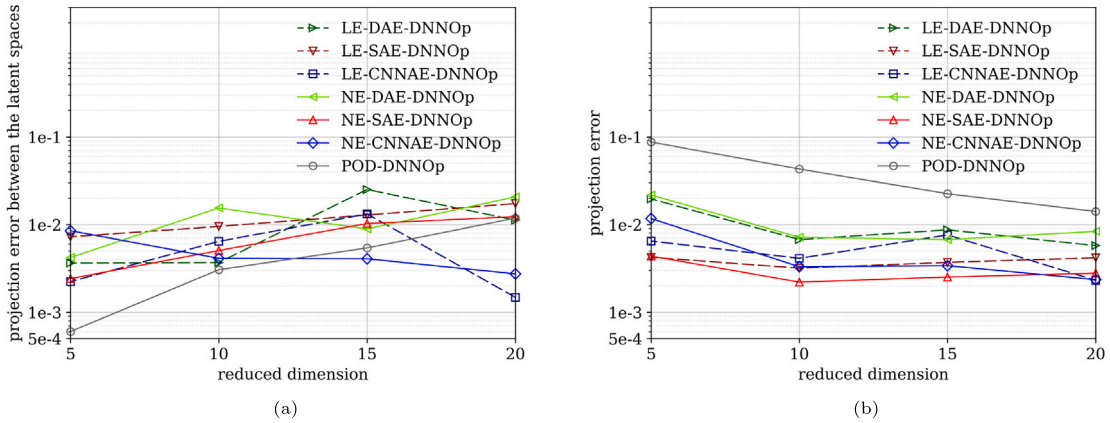


Fig. 9. (a) Projection error between the latent spaces obtained from the surrogate operator $\mathcal{H}$; and (b) projection error of the reduced order model solution on the test dataset with $\mu = 1.0$ over different latent space dimension.

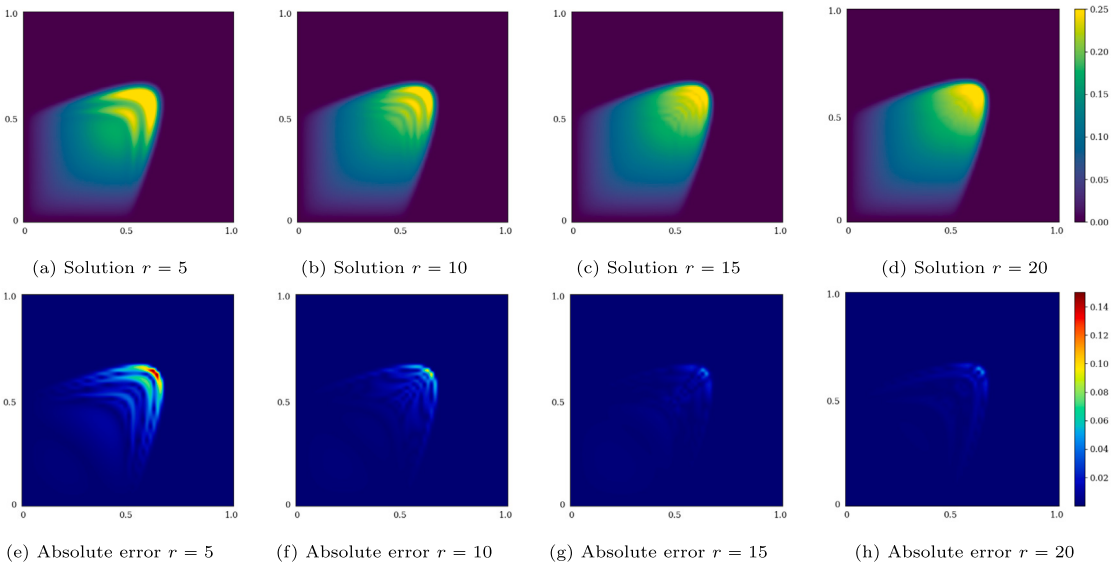
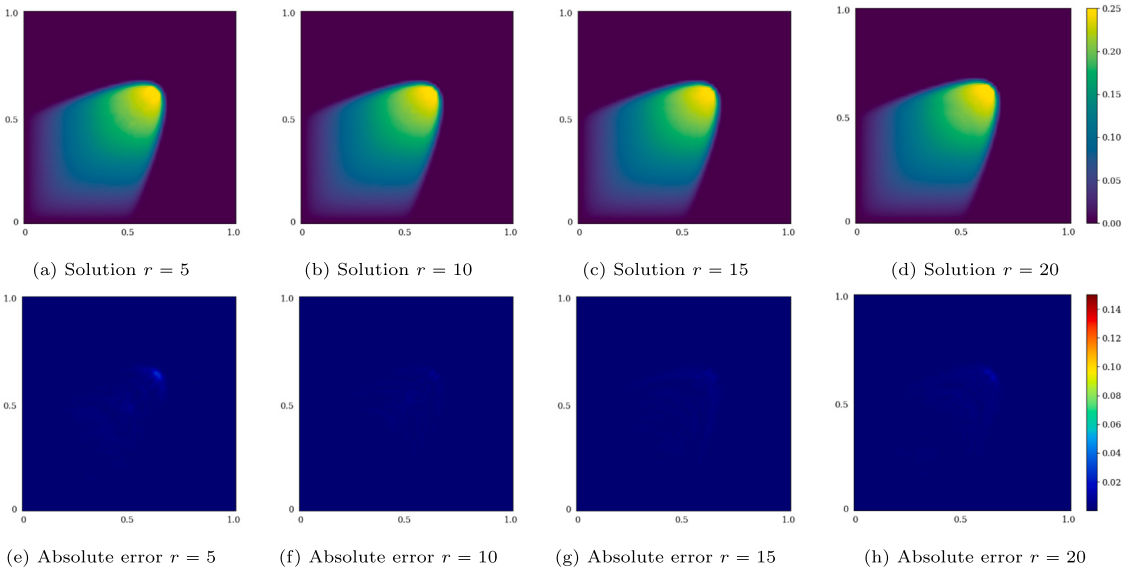


Fig. 10. Solutions (a, b, c, d) and pointwise errors (e, f, g, h) for POD-DNNOp with different latent space dimensions.

latent space (as compared to serial solving for the ROM). Observe that $r \geqslant 15$, speed-ups do not decrease as the chunk size reaches 100. This implies that memory limitation (hardware limitation) was not reached for any of the GPUs, which indicates that the tested chunk sizes may be suboptimal. For $r = 10$ and a chunk size equal to 100, speed-ups decrease for all configurations, which suggests that for this specific case, the required number of nonlinear iterations is larger than for the other cases. For this choice of parameters, the solver eventually fails, hence resulting in infinite projection errors (as shown in Table 2). To further support this explanation, the maximum number of Newton–Raphson (N–R) iterations required for each chunk when solving the adjoint integration problems is also reported. It is seen that when $r = 10$ and for a chunk size equal to 100, the number of nonlinear iterations reaches its maximum allowed value (set to 100), for all GPUs. This behavior is a manifestation of the interplay between the chunk size, the initial guess in the Newton–Raphson solver, and the latent space dimension. When the dimension r are large enough, the solution can be obtained even under poor initialization: the number of solver iterations then increases together with the chunk size. This can be seen in Table 2: when the chunk size equals 1, around 2 iterations are necessary to reach convergence, while around 10 iterations are needed for a chunk size set to 100.

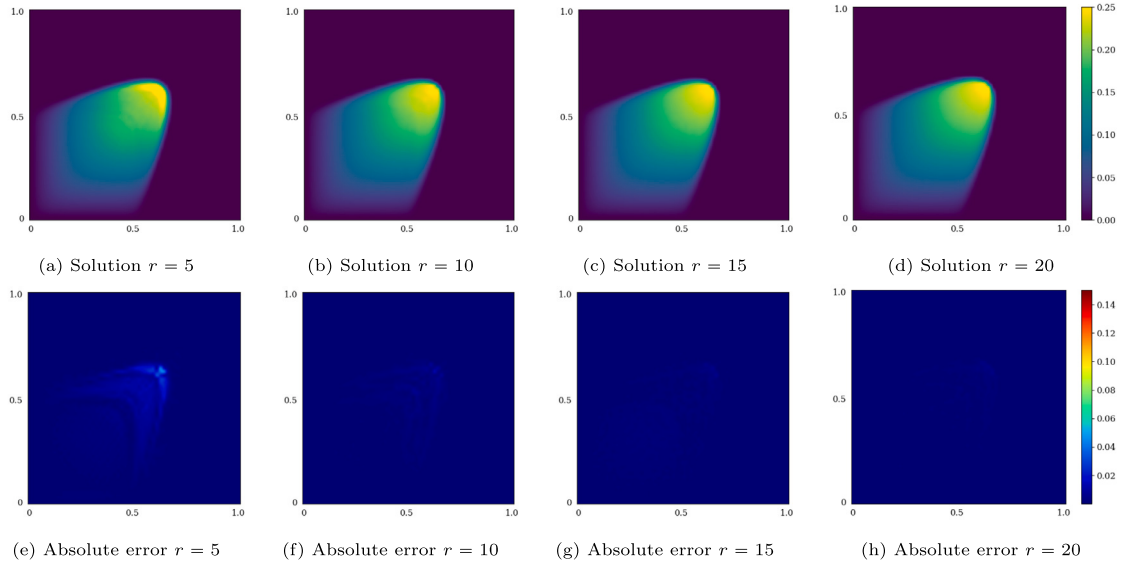
We also evaluated the performance of NE-CNNAE on other datasets beyond the range of the training dataset, for $\mu = 1.11$, 1.12, 1.13, 1.14, 1.15. The projection errors of the reduced-order model and the projection error between the latent spaces of NE-CNNAE with $n = 20$ are shown in Fig. 14. This figure illustrates how the accuracy decreases in the extrapolation regime, with an increase of almost one order of magnitude between $\mu = 1.1$ (included in the training dataset) and $\mu = 1.15$.



(a) Solution $r = 5$ (b) Solution $r = 10$ (c) Solution $r = 15$ (d) Solution $r = 20$

(e) Absolute error $r = 5$ (f) Absolute error $r = 10$ (g) Absolute error $r = 15$ (h) Absolute error $r = 20$

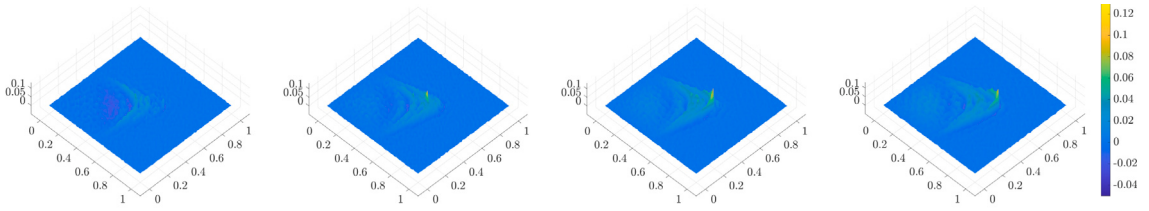
Fig. 11. Solutions (a, b, c, d) and pointwise errors (e, f, g, h) for NE-SAE-DNNOp with different latent space dimensions.



(a) Solution $r = 5$ (b) Solution $r = 10$ (c) Solution $r = 15$ (d) Solution $r = 20$

(e) Absolute error $r = 5$ (f) Absolute error $r = 10$ (g) Absolute error $r = 15$ (h) Absolute error $r = 20$

Fig. 12. Solutions (a, b, c, d) and pointwise errors (e, f, g, h) for NE-CNNAE-DNNOp with different latent space dimensions.



(a) $M = 201$ (b) $M = 601$ (c) $M = 1001$ (d) $M = 1401$

Fig. 13. This figure illustrates adaptivity by showing the evolution of the fourth basis in Eq. (15). Here, we take the basis at the 101st time step as reference, and plot the evolution of the difference $\Delta D_4(M) = D_4(\hat{u}_4^{\langle M\rangle}; \hat{\mathbf{u}}^{\langle M\rangle}) - D_4(\hat{u}_4^{\langle 101\rangle}; \hat{\mathbf{u}}^{\langle 101\rangle})$ at selected time steps (namely, 201, 601, 1001, and 1401). Note that the basis is shown in physical space to enable proper visualization.

Table 2
Wall-clock times at different latent space dimensions and chunk size on NVIDIA RTX A4500, NVIDIA RTX A6000, and NVIDIA GeForce 4080 for 1500 time steps on the test dataset ($\mu = 1.0$). The shortest wall-clock time is obtained when $r = 15$ using NVIDIA RTX A6000 with 1.32 s.

NVIDIA RTX A4500												
Reduced dimension r	10				15				20			
Chunk size	1	10	50	100	1	10	50	100	1	10	50	100
Projection error (%)	0.33	0.33	0.33	INF	0.34	0.34	0.34	0.34	0.23	0.23	0.23	0.23
N–R iterations #	2	3	6	100	3	3	5	7	3	4	7	12
Wall time (s)	34.08	5.10	1.61	6.94	34.65	5.06	1.62	1.36	34.46	5.07	1.78	1.48
Speed-up	1	6.68	21.17	4.91	1	6.85	21.39	**25.48**	1	6.80	19.36	23.28
NVIDIA RTX A6000												
Reduced dimension r	10				15				20			
Chunk size	1	10	50	100	1	10	50	100	1	10	50	100
Projection error (%)	0.33	0.33	0.33	INF	0.34	0.34	0.34	0.34	0.23	0.23	0.23	0.23
N–R iterations #	2	3	6	100	3	3	5	6	2	3	6	8
Wall time (s)	10.49	2.4	1.2	5.87	10.72	2.34	1.42	**1.32**	14.19	2.26	2.08	1.38
Speed-up	1	4.37	8.74	1.79	1	4.58	7.55	8.12	1	6.59	7.16	10.80
NVIDIA GeForce 4080												
Reduced dimension r	10				15				20			
Chunk size	1	10	50	100	1	10	50	100	1	10	50	100
Projection error (%)	0.33	0.33	0.33	INF	0.34	0.34	0.34	0.34	0.23	0.23	0.23	0.23
N–R iterations #	2	3	6	100	3	3	5	7	2	3	5	7
Wall-clock time (s)	14.74	1.37	0.73	8.69	13.02	2.59	1.50	1.51	13.52	2.62	1.64	1.64
Speed-up	1	10.76	20.19	1.67	1	5.03	8.67	8.62	1	5.16	8.24	8.24

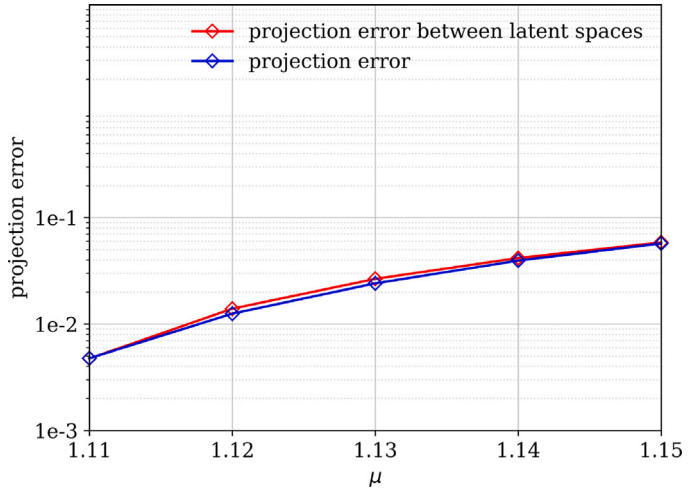


Fig. 14. Projection error between the latent spaces obtained from the surrogate operator $\mathcal{H}$ and projection error of the reduced-order model solution on the test dataset with $\mu = 1.11, 1.12, 1.13, 1.14, 1.15$ when $n = 20$ of NE-CNNAE.

4.4. Remarks on complexity and time step size dependency

For nonlinear autoencoders, the number of trainable parameters can be considered as a measure of model complexity. For the above numerical example, with a latent dimension equal to 20, DAE has around 140 millions of trainable parameters, SAE has around 25 millions of trainable parameters, and the proposed CNNAE (which achieves best performance, together with SAE) has around 1 million trainable parameters. When training the models for 22 batches per epoch on NVIDIA RTX A4500 (64 bits, 33 MHz, 20 GB), with each batch containing approximately 246 time steps of solutions with a latent dimension set to 20, LE-DAE requires 14.70 seconds, LE-SAE 14.96 seconds, LE-CNNAE 1.56 seconds, NE-DAE 14.72 seconds, NE-SAE 14.98 seconds, and NE-CNNAE requires 1.58 seconds per epoch (average wall-clock time over 100 epochs).

It should be noticed that the considered operator learning strategies are, by construction, dependent on the time step size used for the training dataset. To quantify this effect, projection errors (in latent space and for the reduced-order model) obtained with NE-SAE and NE-CNNAE, and refined step sizes, are shown in Fig. 15. Here, the surrogates were trained with a time step size denoted by Δt, and the projection errors are evaluated by running both the FOM and ROMs with decreasing time step sizes: $\{\Delta t, \Delta t/2, \Delta t/4, \Delta t/8\}$. It is seen that both errors substantially increase as the time resolution is refined, which is consistent with results reported elsewhere

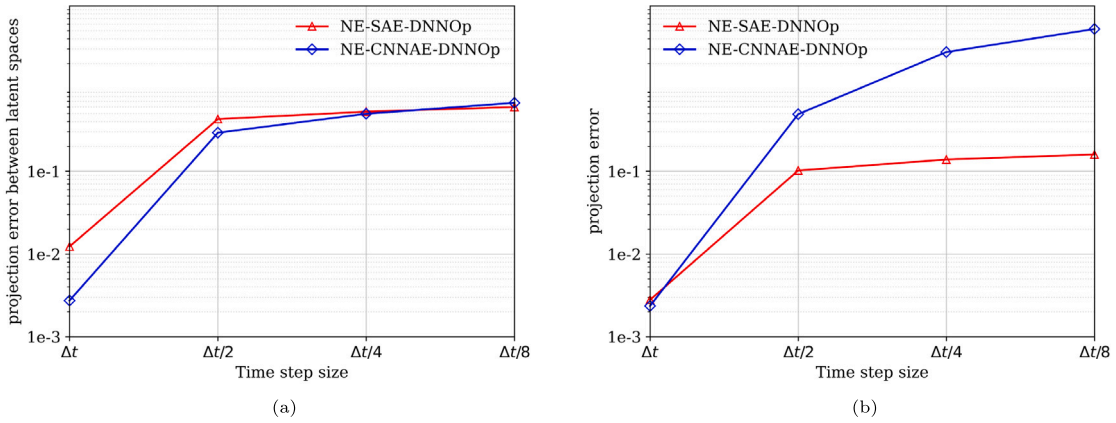


Fig. 15. (a) Projection error of the solution in the latent space; and (b) projection error of the reduced-order model with $r = 20$ on the test dataset over different time step sizes. Here, Δt is the time step size used for training the autoencoders and the reduced operators.

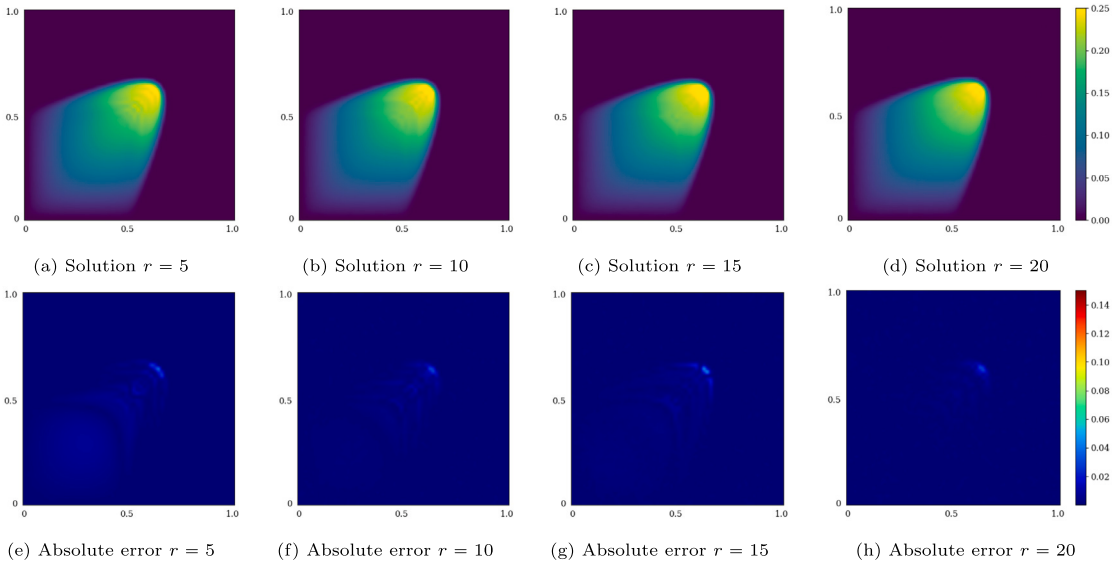
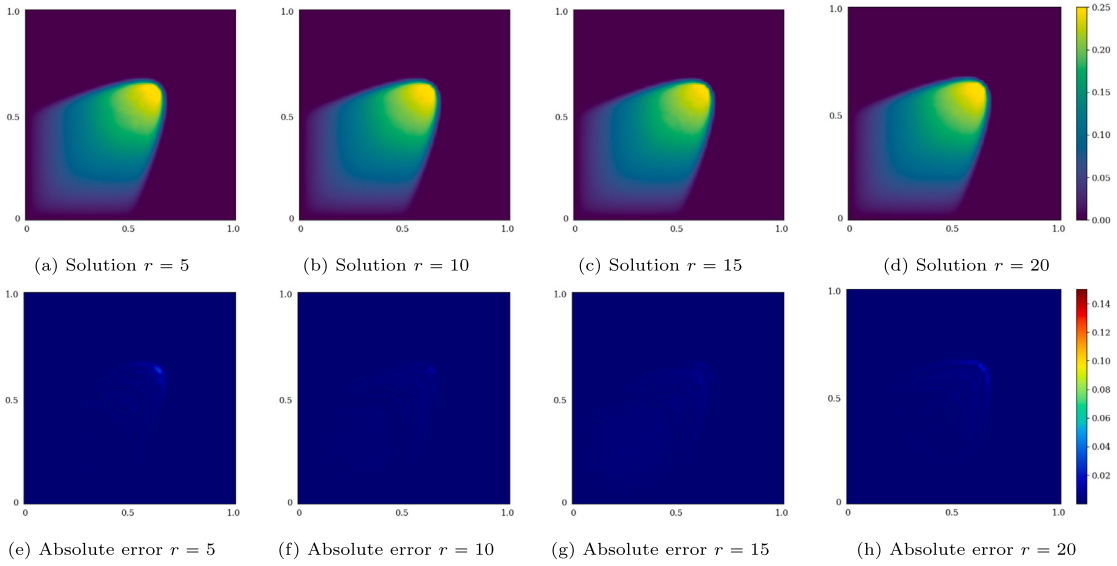


Fig. A.16. Solutions (a, b, c, d) and pointwise errors (e, f, g, h) for LE-DAE-DNNOp with different latent space dimensions.

(see, e.g., [53] for similar behavior with space discretization). The increase in the ROM projection error is larger for CNNAE than for SAE (see the right panel in Fig. 15), which may be attributed to the fact that the adaptive basis is constructed for a given time step size.

5. Conclusion

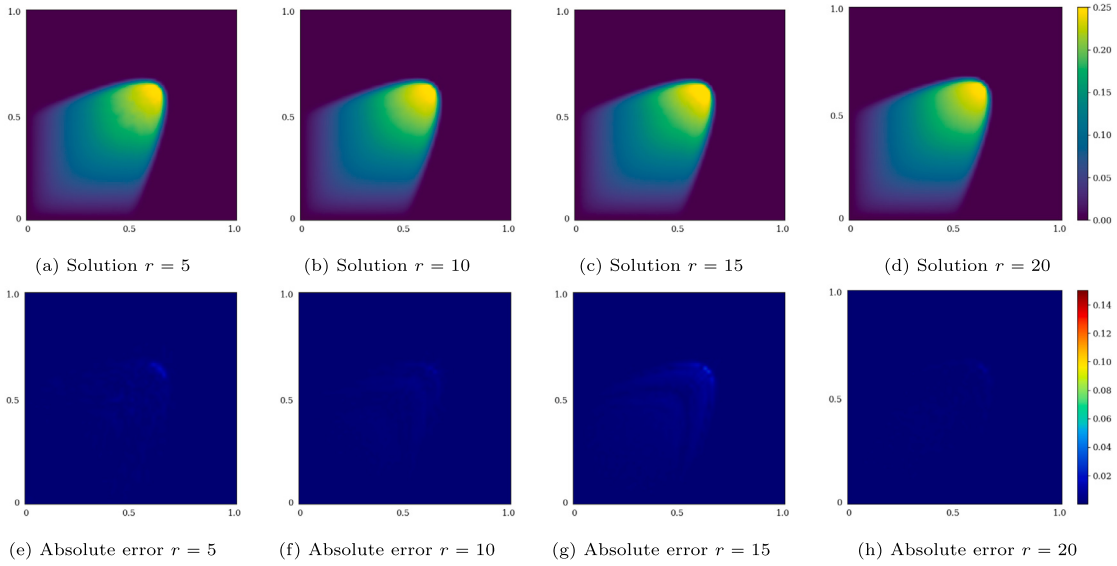
An operator learning strategy for flow maps exhibiting sharp gradients was proposed. State-of-the-art linear and nonlinear reduction techniques were first reviewed. We then devised a new Convolutional Neural Network (CNN)-based autoencoder. Taking inspiration from adaptive basis methods, the proposed architecture involves iteration-dependent trainable kernels at the decoding stage. Quadratic operator inference and a Deep Neural Network (DNN)-based operator inference model were next investigated to learn the flow map in latent space. We also proposed a strategy to accelerate reduced-order model solving through vectorized implicit time integration. The set of operator learning methods and algorithms thus obtained was benchmarked on an advection-dominated problem (here, a 2D Burgers' equation). It was found that the sparse autoencoder and the proposed CNN-based autoencoder generally perform better in terms of prediction accuracy. It is noticeable however that the latter strategy achieves such performance with much fewer trainable parameters (typically, one order of magnitude less). Regarding operator learning between the latent spaces, quadratic operator inference performed fairly well at the training stage, but was found unstable on the test case. On the contrary, the DNN-based strategy delivered accurate predictions during both training and testing. Finally, it was shown that the vectorized



(a) Solution $r = 5$ (b) Solution $r = 10$ (c) Solution $r = 15$ (d) Solution $r = 20$

(e) Absolute error $r = 5$ (f) Absolute error $r = 10$ (g) Absolute error $r = 15$ (h) Absolute error $r = 20$

Fig. A.17. Solutions (a, b, c, d) and pointwise errors (e, f, g, h) for LE-SAE-DNNOp with different latent space dimensions.



(a) Solution $r = 5$ (b) Solution $r = 10$ (c) Solution $r = 15$ (d) Solution $r = 20$

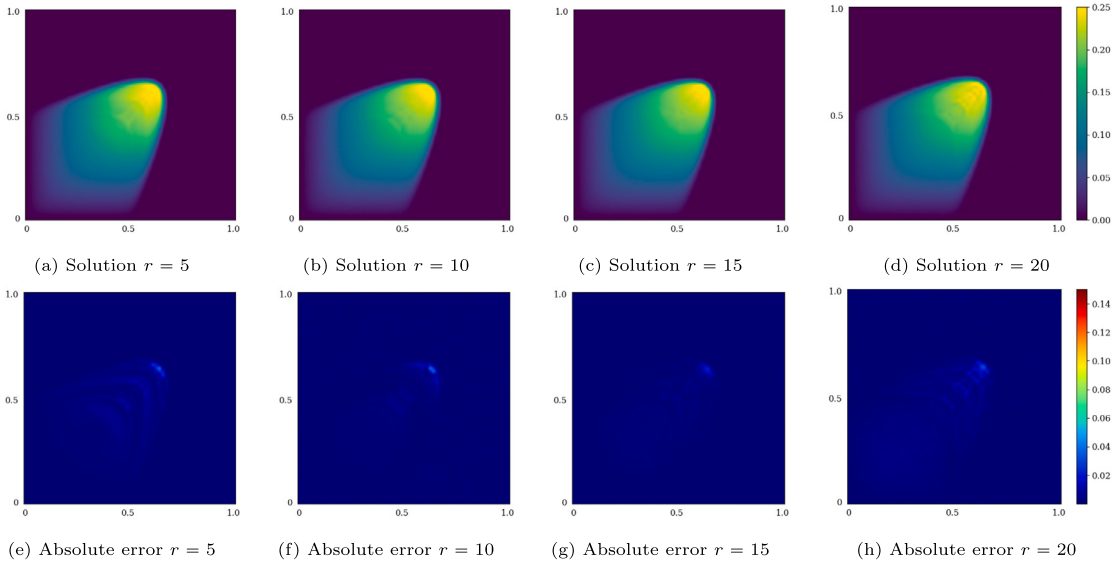
(e) Absolute error $r = 5$ (f) Absolute error $r = 10$ (g) Absolute error $r = 15$ (h) Absolute error $r = 20$

Fig. A.18. Solutions (a, b, c, d) and pointwise errors (e, f, g, h) for LE-CNNAE-DNNOp with different latent space dimensions.

implicit time integration enables substantial speed-ups as the latent space dimension increases. The main limitation of the proposed method lies in time-resolution dependency. A possible way to circumvent this issue is to finite-dimensionalize in time domain, in the spirit of the PCA-Net framework introduced by Stuart and coworkers. Such developments, together with application to other problems presenting non-smooth solution fields, are left for future work.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Johann Guilleminot reports financial support was provided by National science Foundation. If there are other authors they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.



(a) Solution $r = 5$ (b) Solution $r = 10$ (c) Solution $r = 15$ (d) Solution $r = 20$

(e) Absolute error $r = 5$ (f) Absolute error $r = 10$ (g) Absolute error $r = 15$ (h) Absolute error $r = 20$

Fig. A.19. Solutions (a, b, c, d) and pointwise errors (e, f, g, h) for NE-DAE-DNNOp with different latent space dimensions.

Data availability

Data will be made available on request.

Acknowledgments

The work of the P.C. and J.G. was supported by the National Science Foundation, United States under awards DGE-2022040 and CMMI-1942928. The authors would like to thank Dr. Youngsoo Choi for sharing the reference data used in the application.

Appendix. Reduced-order model solutions

In this Appendix, additional solution snapshots are provided, together with absolute errors, to assess the accuracy of the presented frameworks (see Figs. A.16–A.19).

References

[1] Peter Benner, Serkan Gugercin, Karen Willcox, A survey of projection-based model reduction methods for parametric dynamical systems, SIAM Rev. 57 (4) (2015) 483–531.
[2] Gal Berkooz, Philip Holmes, John L. Lumley, The proper orthogonal decomposition in the analysis of turbulent flows, Annu. Rev. Fluid Mech. 25 (1) (1993) 539–575.
[3] John Leask Lumley, The structure of inhomogeneous turbulent flows, Atmos. Turbul. Radio Wave Propag. (1967).
[4] Lawrence Sirovich, Turbulence and the dynamics of coherent structures. I. Coherent structures, Q. Appl. Math. 45 (3) (1987) 561–571.
[5] Peter Benner, Tobias Breiten, Two-sided projection methods for nonlinear model order reduction, SIAM J. Sci. Comput. 37 (2) (2015) B239–B260.
[6] Nguyen Ngoc Cuong, Karen Veroy, Anthony T. Patera, Certified real-time solution of parametrized partial differential equations, in: Handbook of Materials Modeling, Springer, 2005, pp. 1529–1564.
[7] Pawan Kumar Goyal, Peter Benner, Algebraic gramians for quadratic-bilinear systems and their application in model order reduction, in: 22nd International Symposium on Mathematical Theory of Networks and Systems, 2016.
[8] Jan S. Hesthaven, Gianluigi Rozza, Benjamin Stamm, et al., Certified Reduced Basis Methods for Parametrized Partial Differential Equations, vol. 590, Springer, 2016.
[9] Patricia Astrid, Siep Weiland, Karen Willcox, Ton Backx, Missing point estimation in models described by proper orthogonal decomposition, IEEE Trans. Automat. Control 53 (10) (2008) 2237–2251.
[10] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, Anthony T Patera, An 'empirical interpolation' method: Application to efficient reduced-basis discretization of partial differential equations, C. R. Math. 339 (9) (2004) 667–672.
[11] Kevin Carlberg, Charbel Farhat, Julien Cortial, David Amsallem, The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows, J. Comput. Phys. 242 (2013) 623–647.
[12] Saifon Chaturantabut, Danny C. Sorensen, Nonlinear model reduction via discrete empirical interpolation, SIAM J. Sci. Comput. 32 (5) (2010) 2737–2764.
[13] Zlatko Drmac, Serkan Gugercin, A new selection operator for the discrete empirical interpolation method—improved a priori error bound and extensions, SIAM J. Sci. Comput. 38 (2) (2016) A631–A648.
[14] Martin A. Grepl, Yvon Maday, Ngoc C. Nguyen, Anthony T. Patera, Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations, ESAIM Math. Model. Numer. Anal. 41 (3) (2007) 575–605.

[15] Ngoc C. Nguyen, Anthony T. Patera, Jaime Peraire, A 'best points' interpolation method for efficient approximation of parametrized functions, Internat. J. Numer. Methods Engrg. 73 (4) (2008) 521–543.
[16] Boris Kramer, Karen E. Willcox, Nonlinear model order reduction via lifting transformations and proper orthogonal decomposition, AIAA J. 57 (6) (2019) 2297–2307.
[17] Boris Kramer, Karen E. Willcox, Balanced truncation model reduction for lifted nonlinear systems, 2019, arXiv preprint arXiv:1907.12084.
[18] Joshua Barnett, Charbel Farhat, Yvon Maday, Neural-network-augmented projection-based model order reduction for mitigating the Kolmogorov barrier to reducibility, J. Comput. Phys. 492 (2023) 112420.
[19] M.W.M.G. Dissanayake, Nhan Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201.
[20] Isaac E. Lagaris, Aristidis Likas, Dimitrios I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[21] Andrew J. Meade Jr., Alvaro A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, Math. Comput. Modelling 19 (12) (1994) 1–25.
[22] B. Ph van Milligen, V. Tribaldos, J.A. Jiménez, Neural network differential equation and plasma equilibrium solver, Phys. Rev. Lett. 75 (20) (1995) 3594.
[23] Jiequn Han, Arnulf Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510.
[24] Lu Lu, Pengzhan Jin, George Em Karniadakis, Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators, 2019, arXiv preprint arXiv:1910.03193.
[25] Lu Lu, Xuhui Meng, Zhiping Mao, George Em Karniadakis, DeepXDE: A deep learning library for solving differential equations, SIAM Rev. 63 (1) (2021) 208–228.
[26] Guofei Pang, Lu Lu, George Em Karniadakis, fPINNs: Fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626.
[27] Maziar Raissi, Paris Perdikaris, George E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[28] Kookjin Lee, Kevin T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, J. Comput. Phys. 404 (2020) 108973.
[29] Kookjin Lee, Kevin Carlberg, Deep conservation: A latent dynamics model for exact satisfaction of physical conservation laws, 2019, arXiv preprint arXiv:1909.09754.
[30] Youngkyu Kim, Youngsoo Choi, David Widemann, Tarek Zohdi, A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder, J. Comput. Phys. 451 (2022) 110841.
[31] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, George Em Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218–229.
[32] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Fourier neural operator for parametric partial differential equations, 2020, arXiv preprint arXiv:2010.08895.
[33] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, Anima Anandkumar, Fourier neural operator with learned deformations for pdes on general geometries, 2022, arXiv preprint arXiv:2207.05209.
[34] Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, Sally M. Benson, U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow, Adv. Water Resour. 163 (2022) 104180.
[35] Dongkun Zhang, Lu Lu, Ling Guo, George Em Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, J. Comput. Phys. 397 (2019) 108850.
[36] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, Anima Anandkumar, Multipole graph neural operator for parametric partial differential equations, Adv. Neural Inf. Process. Syst. 33 (2020) 6755–6766.
[37] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Neural operator: Graph kernel network for partial differential equations, 2020, arXiv preprint arXiv:2003.03485.
[38] Alasdair Tran, Alexander Mathews, Lexing Xie, Cheng Soon Ong, Factorized fourier neural operators, 2021, arXiv preprint arXiv:2111.13802.
[39] John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, Bryan Catanzaro, Adaptive fourier neural operators: Efficient token mixers for transformers, 2021, arXiv preprint arXiv:2111.13587.
[40] Tapas Tripura, Souvik Chakraborty, Wavelet neural operator: A neural operator for parametric partial differential equations, 2022, arXiv preprint arXiv:2205.02191.
[41] Gaurav Gupta, Xiongye Xiao, Paul Bogdan, Multiwavelet-based operator learning for differential equations, Adv. Neural Inf. Process. Syst. 34 (2021) 24048–24062.
[42] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Neural operator: Learning maps between function spaces, 2021, arXiv preprint arXiv:2108.08481.
[43] Pengzhan Jin, Shuai Meng, Lu Lu, MIONet: Learning multiple-input operators via tensor product, SIAM J. Sci. Comput. 44 (6) (2022) A3490–A3514.
[44] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, Anima Anandkumar, Physics-informed neural operator for learning partial differential equations, 2021, arXiv preprint arXiv:2111.03794.
[45] Bogdan Raonic, Roberto Molinaro, Tobias Rohner, Siddhartha Mishra, Emmanuel de Bezenac, Convolutional neural operators, in: ICLR 2023 Workshop on Physics for Machine Learning, 2023.
[46] Pau Batlle, Matthieu Darcy, Bamdad Hosseini, Houman Owhadi, Kernel methods are competitive for operator learning, 2023, arXiv:2304.13202.
[47] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, George Em Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data, Comput. Methods Appl. Mech. Engrg. 393 (2022) 114778.
[48] Nikola B. Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M. Stuart, Anima Anandkumar, Neural operator: Learning maps between function spaces with applications to PDEs, J. Mach. Learn. Res. 24 (89) (2023) 1–97.
[49] Hengjie Wang, Robert Planas, Aparna Chandramowlishwaran, Ramin Bostanabad, Mosaic flows: A transferable deep learning framework for solving PDEs on unseen domains, Comput. Methods Appl. Mech. Engrg. 389 (2022) 114424.
[50] Elizabeth Qian, Ionut-Gabriel Farcas, Karen Willcox, Reduced operator inference for nonlinear partial differential equations, SIAM J. Sci. Comput. 44 (4) (2022) A1934–A1959.
[51] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, Andrew M. Stuart, Model reduction and neural networks for parametric PDEs, SMAI J. Comput. Math. 7 (2021) 121–157.
[52] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[53] Yinhao Zhu, Nicholas Zabaras, Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447.