Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 388 (2022) 114181
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

POD-DL-ROM: Enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition

Stefania Fresca, Andrea Manzoni*

MOX - Dipartimento di Matematica, Politecnico di Milano, P.zza Leonardo da Vinci 32, 20133 Milano, Italy

Received 4 February 2021; received in revised form 23 August 2021; accepted 6 September 2021
Available online 13 October 2021

Abstract

Deep learning-based reduced order models (DL-ROMs) have been recently proposed to overcome common limitations shared by conventional reduced order models (ROMs) – built, e.g., through proper orthogonal decomposition (POD) – when applied to nonlinear time-dependent parametrized partial differential equations (PDEs). These might be related to *(i)* the need to deal with projections onto high dimensional linear approximating trial manifolds, *(ii)* expensive hyper-reduction strategies, or *(iii)* the intrinsic difficulty to handle physical complexity with a linear superimposition of modes. All these aspects are avoided when employing DL-ROMs, which learn in a non-intrusive way both the nonlinear trial manifold and the reduced dynamics, by relying on deep (e.g., feedforward, convolutional, autoencoder) neural networks. Although extremely efficient at testing time, when evaluating the PDE solution for any new testing-parameter instance, DL-ROMs require an expensive training stage, because of the extremely large number of network parameters to be estimated. In this paper we propose a possible way to avoid an expensive training stage of DL-ROMs, by *(i)* performing a prior dimensionality reduction through POD, and *(ii)* relying on a multi-fidelity pretraining stage, where different physical models can be efficiently combined. The proposed POD-DL-ROM is tested on several (both scalar and vector, linear and nonlinear) time-dependent parametrized PDEs (such as, e.g., linear advection–diffusion–reaction, nonlinear diffusion–reaction, nonlinear elastodynamics, and Navier–Stokes equations) to show the generality of this approach and its remarkable computational savings.

Keywords: Reduced order modeling; Deep learning; Proper orthogonal decomposition; Dimensionality reduction; Parametrized PDEs

1. Introduction

Performing the numerical approximation of parametrized partial differential equations (PDEs) for multiple parameter values, or solving them in real-time, is unaffordable by means of full order, traditional high-fidelity techniques, such as the Galerkin-finite element method [1]. Replacing a full order model (FOM) by a reduced order model (ROM), featuring a much lower dimension, yet capable to express the physical features of the problem at hand, is the main goal of reduced order modeling techniques, among which the reduced basis (RB) method

* Corresponding author.
E-mail addresses: stefania.fresca@polimi.com (S. Fresca), andrea1.manzoni@polimi.com (A. Manzoni).

https://doi.org/10.1016/j.cma.2021.114181

represents one of the most popular options [2]. The basic assumption underlying the RB method is that the solution of a parametrized PDE lies on a low-dimensional manifold, which can be approximated by a linear trial subspace spanned by a set of basis functions [3,4], built from a set of FOM snapshots employing, e.g., proper orthogonal decomposition (POD). In this case, the ROM approximation is given by the linear superimposition of POD modes, whose degrees of freedom (depending both on time and parameters) result from the solution of a low-dimensional (nonlinear, dynamical) system, obtained through a (Petrov-)Galerkin projection onto a linear test subspace, which might coincide with the trial subspace. Being able to assemble such a ROM efficiently, for any new parameter instance, is granted at the price of a further *hyper-reduction* stage on the FOM arrays [5].

ROMs for parametrized PDEs rely on a suitable offline–online computational splitting: computationally expensive tasks required to build the low-dimensional subspace, and assemble all the ROM arrays, are performed once for all during the so-called *offline* (or *ROM training*) stage, then allowing to compute – ideally – in an extremely efficient way the ROM approximation for any new parameter value, during the so-called *online* (or *ROM testing*) stage. This computational strategy, however, breaks down if *(i)* the dimension of the linear trial subspace becomes very large (compared to the intrinsic dimension of the solution manifold being approximated), or *(ii)* the hyper-reduction stage, required to approximate parameter-dependent nonlinear terms – at the limit, the whole residual vector and, possibly, the Jacobian matrix when dealing with implicit solvers such as, e.g., the Newton method – requires linear subspaces whose dimension becomes very large, too, in order to provide an approximation to FOM arrays sufficiently accurate. These can be recurrent issues when dealing with nonlinear, time-dependent parametrized PDEs, and *(i)* we aim at building a ROM able to provide problem approximations uniformly accurate over the whole parameter space, *(ii)* the parametrized problem features coherent structures (e.g., transport or wave phenomena) that propagate over time and depend on parameters. Last, but not least, ensuring ROM stability might require additional computational efforts, such as, e.g., when dealing with fluid flows using a mixed formulation (e.g., a velocity–pressure formulation for Navier–Stokes equations), see, e.g., [6–8].

To overcome these drawbacks, we have recently proposed in [9,10] a strategy to construct deep learning-based ROMs (DL-ROMs) for nonlinear time-dependent parametrized PDEs in a non-intrusive way, exploiting deep neural networks as main building block, and a set of FOM snapshots. In particular, the DL-ROM technique allows approximating both the solution manifold of the PDE by means of a low-dimensional, nonlinear trial manifold, and the nonlinear dynamics of the generalized coordinates on such reduced trial manifold, as a function of the time coordinate and the parameters. Both *(i)* the nonlinear trial manifold and *(ii)* the reduced dynamics are learnt in a non-intrusive way, thus avoiding the projection stage typical of the RB method; in particular, the trial manifold is learnt by means of the decoder function of a convolutional autoencoder (CAE) neural network, whereas the reduced dynamics through a (deep) feedforward neural network (DFNN), and the encoder function of the CAE (see Section 3.2 for further details).

DL-ROMs outperform the RB method – even involving local reduced bases – regarding both numerical accuracy (for the same ROM dimension) and computational efficiency during the *online* (or *testing*) stage, when applied to problems that are typically challenging for the RB method (such as, e.g., linear transport equation, nonlinear diffusion–reaction equations whose solution develops moving fronts depending on parameters) or problems featuring reduced bases with usually large dimension. A key aspect, still open in the setting of DL-ROMs and which this paper is mainly devoted to, deals with the computational efficiency of DL-ROMs during the *offline* (or *training*) stage, which is also related with the curse of dimensionality. Indeed, training the two networks representing the building blocks of the DL-ROM entails a number of networks parameters to be estimated (and, correspondingly, training data dimensions) that blow up with the dimension N_h of the FOM — this latter being related with, e.g., the mesh size required by a Galerkin-finite element approximation of the PDEs. Indeed, DL-ROMs have been applied so far to the reduction of scalar PDEs, with at most $N_h = O(10^4)$ degrees of freedom.

In this paper we propose a strategy to enhance DL-ROMs in order to make the *offline* training stage dramatically faster, allowing for much larger FOM dimensions, without affecting the number of networks parameters to be estimated and, ultimately, network complexity. This strategy exploits *(i)* dimensionality reduction of FOM snapshots through randomized POD (rPOD) [11], to be considered as the action of the first *layer* of the CAE, rather than the way to generate the linear trial manifold, as done instead in traditional POD-Galerkin ROMs, and *(ii)* a suitable multi-fidelity pretraining stage [12], where different models (built, e.g., by considering coarser discretizations or simplified physical models) can be efficiently combined, to iteratively initialize the network parameters.

These substantial enhancements of the DL-ROM technique provide a new way to build deep learning-based ROMs, which we refer to POD-DL-ROM. The resulting strategy represents a suitable combination of the best

features of deep learning (DL) algorithms and POD – namely, the non-intrusive character of the former, and the simplicity, combined with the rigorous mathematical foundation, of the latter – at the same time fixing their weaknesses – namely, the curse of dimensionality of DL-ROM for increasing FOM dimensions, and the modest approximation properties of POD-based linear trial manifolds for some classes of nonlinear parametrized PDEs. As a result, the POD-DL-ROM technique provides not only substantial gains during the *offline* training stage if compared to the DL-ROM strategy – keeping fixed the dimension of the FOM that we used to generate snapshots – but also during the *online* testing stage, if compared to POD-Galerkin ROMs, also in those cases where this latter technique provides satisfying results in terms of both accuracy and efficiency.

The structure of the paper is as follows. In Section 2 we briefly recall the construction of DL-ROMs by reinterpreting the classical ideas behind linear projection-based methods for parametrized PDEs, comparing our strategy with alternative ways to build ROMs relying on machine/deep learning algorithms. In Section 3 we then describe the POD-DL-ROM technique showing *(i)* how to enhance the DL-ROM technique by means of (randomized) POD, *(ii)* how to rely on a multi-fidelity pretraining stage, and *(iii)* how to generalize this technique to the case of vector problems. In Section 4 we assess the numerical accuracy and efficiency on four different test cases, namely *(i)* a linear advection–diffusion–reaction problem, *(ii)* a nonlinear diffusion–reaction problem arising from cardiac electrophysiology, *(iii)* nonlinear elastodynamics for hyperelastic compressible materials, and *(iv)* fluid dynamics, also showing the great versatility of the proposed technique.

2. An overview of deep learning-based ROMs

Before introducing the main features of POD-DL-ROMs, we review the construction of DL-ROMs and highlight the main differences between our framework and existing techniques in literature.

2.1. DL-ROMs in a nutshell

For the sake of generality, we consider a generic nonlinear, time-dependent PDE depending on a set of input parameters $\boldsymbol{\mu} \in \mathcal{P}$, where the parameter space $\mathcal{P} \subset \mathbb{R}^{n_{\boldsymbol{\mu}}}$ is given by a bounded and closed set; input parameters might represent physical or geometrical properties of the system, like, e.g., material properties, initial and boundary conditions, or the shape of the domain. Even if we only consider physical parameters, handling geometrical parameters does not require additional efforts. We adopt a fully algebraic perspective and assume to start from the high-fidelity (spatial) approximation of the PDE, which we refer to as full order model (FOM). Regardless of the spatial discretization adopted – such as, e.g., the finite element method, Isogeometric Analysis or the spectral element method – the FOM can be expressed as a nonlinear parametrized dynamical system. Hence, given $\boldsymbol{\mu} \in \mathcal{P}$, we aim at solving the initial value problem

$$\begin{cases} \mathbf{M}(\boldsymbol{\mu})\dot{\mathbf{u}}_h(t;\boldsymbol{\mu}) = \mathbf{f}(t, \mathbf{u}_h(t;\boldsymbol{\mu});\boldsymbol{\mu}) \quad t \in (0,T), \\ \mathbf{u}_h(0;\boldsymbol{\mu}) = \mathbf{u}_0(\boldsymbol{\mu}), \end{cases} \tag{1}$$

where:

- $\mathbf{u}_h : [0,T) \times \mathcal{P} \to \mathbb{R}^{N_h}$ is the parametrized solution of (1);
- $\mathbf{u}_0 : \mathcal{P} \to \mathbb{R}^{N_h}$ is the initial datum;
- $\mathbf{f} : (0,T) \times \mathbb{R}^{N_h} \times \mathcal{P} \to \mathbb{R}^{N_h}$ is a (nonlinear) function, encoding the system dynamics;
- $\mathbf{M}(\boldsymbol{\mu}) \in \mathbb{R}^{N_h \times N_h}$ is the parametric, mass matrix of this parametric FOM; without any loss of generality, $\mathbf{M}(\boldsymbol{\mu})$ is assumed here to be a symmetric positive definite matrix.

The FOM dimension N_h is related with the finite dimensional subspaces introduced for the sake of space discretization — here $h > 0$ denotes a discretization parameter, such as the maximum diameter of elements in a computational mesh; consequently, N_h can be extremely large if the PDE problem describes complex physical behaviors and/or high degrees of accuracy are required to its solution. In order to solve problem (1), suitable time discretizations are employed, such as backward differentiation formulas (BDFs) [1] and generalized-α [13] methods. We thus aim at approximating, in an efficient way, the set

$$\mathcal{S}_h = \{\mathbf{u}_h(t;\boldsymbol{\mu}) \mid t \in [0,T)\ ,\ \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N_h}, \tag{2}$$

of solutions to problem (1) when $(t; \boldsymbol{\mu})$ varies in $[0, T) \times \mathcal{P}$, also referred to as *solution manifold*. If for any $\boldsymbol{\mu} \in \mathcal{P}$, problem (1) admits a unique solution, for each $t \in (0, T)$, that is the dynamical system (1) has a unique trajectory for each parameter instance, the intrinsic dimension of the solution manifold is at most $n_{\boldsymbol{\mu}} + 1 \ll N_h$, where $n_{\boldsymbol{\mu}}$ is the number of parameters (time plays the role of an additional coordinate) [14]. In this case, each element $\mathbf{u}_h(t; \boldsymbol{\mu}) \in \mathcal{S}_h$ can be described in terms of at most $n_{\boldsymbol{\mu}}+1$ intrinsic coordinates, even if their explicit characterization is, in practice, computationally unaffordable. Equivalently, the tangent space to the manifold at any given $\mathbf{u}_h(t; \boldsymbol{\mu})$ is spanned by $n_{\boldsymbol{\mu}} + 1$ basis vectors.

When dealing with traditional projection-based ROMs, the approximated solution to (1) is sought in the subspace $\text{Col}(\mathbf{V}_n)$ of dimension $n \ll N_h$, spanned by the n columns of $\mathbf{V}_n \in \mathbb{R}^{N_h \times n}$. Hence, a linear ROM looks for an approximation $\tilde{\mathbf{u}}_h(t; \boldsymbol{\mu}) \approx \mathbf{u}_h(t; \boldsymbol{\mu})$ of the form

$$\tilde{\mathbf{u}}_h(t; \boldsymbol{\mu}) = \mathbf{V}_n \mathbf{u}_n(t; \boldsymbol{\mu}), \tag{3}$$

where $\tilde{\mathbf{u}}_h : [0, T) \times \mathcal{P} \to \tilde{\mathcal{S}}_h^{n,lin}$. Hence, the *reduced linear trial manifold* is given by

$$\tilde{\mathcal{S}}_h^{n,lin} = \{\mathbf{V}_n \mathbf{u}_n(t; \boldsymbol{\mu}) \ | \mathbf{u}_n(t; \boldsymbol{\mu}) \in \mathbb{R}^n, \ t \in [0, T) \ , \ \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N_h}. \tag{4}$$

In POD-Galerkin ROMs, $\tilde{\mathcal{S}}_h^{n,lin}$ is spanned by the first n singular vectors of

$$\mathbf{S} = [\mathbf{u}_h^1(t^1; \boldsymbol{\mu}_1) \mid \ldots \mid \mathbf{u}_h^1(t^{N_t}; \boldsymbol{\mu}_1) \mid \ldots \mid \ldots \mathbf{u}_h^1(t^1; \boldsymbol{\mu}_{N_{train}}) \mid \ldots \mid \mathbf{u}_h^1(t^{N_t}; \boldsymbol{\mu}_{N_{train}})], \tag{5}$$

a matrix collecting FOM solutions (or *snapshots*) computed for different parameter values $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_{N_{train}} \in \mathcal{P}$, suitably sampled[1] over the parameter space, at different time instants $\{t^1, \ldots, t^{N_t}\} \subset [0, T]$.

In the POD-Galerkin case, to model the reduced dynamics of the system, we replace $\mathbf{u}_h(t; \boldsymbol{\mu})$ by (3) in system (1), and impose that the residual

$$\mathbf{r}_h(\mathbf{V}_n \mathbf{u}_n(t; \boldsymbol{\mu})) = \mathbf{M}(\boldsymbol{\mu}) \mathbf{V}_n \dot{\mathbf{u}}_n(t; \boldsymbol{\mu}) - \mathbf{f}(t, \mathbf{V}_n \mathbf{u}_n(t; \boldsymbol{\mu}); \boldsymbol{\mu}) \tag{6}$$

is orthogonal to $\tilde{\mathcal{S}}_h^{n,lin}$. This condition yields the following POD-Galerkin ROM

$$\begin{cases} \mathbf{M}_n(\boldsymbol{\mu}) \dot{\mathbf{u}}_n(t; \boldsymbol{\mu}) = \mathbf{f}_n(t, \mathbf{V}_n \mathbf{u}_n(t; \boldsymbol{\mu}); \boldsymbol{\mu}) \quad t \in (0, T), \\ \mathbf{u}_n(0; \boldsymbol{\mu}) = \mathbf{V}_n^T \mathbf{u}_0(\boldsymbol{\mu}), \end{cases} \tag{7}$$

where:

- $\mathbf{M}_n(\boldsymbol{\mu}) = \mathbf{V}_n^T \mathbf{M}(\boldsymbol{\mu}) \mathbf{V}_n$ is the *reduced* mass matrix;
- $\mathbf{f}_n(t, \mathbf{V}_n \mathbf{u}_n(t; \boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{V}_n^T \mathbf{f}(t, \mathbf{V}_n \mathbf{u}_n(t; \boldsymbol{\mu}); \boldsymbol{\mu})$;
- $\mathbf{u}_n(0; \boldsymbol{\mu}) = \mathbf{V}_n^T \mathbf{u}_h(0; \boldsymbol{\mu})$ is the initial condition for $\mathbf{u}_n(t; \boldsymbol{\mu})$ associated with the initial condition for $\mathbf{u}_h(t; \boldsymbol{\mu})$, where we have assumed, without loss of generality, that $\mathbf{V}_n^T \mathbf{V}_n = \mathbf{I} \in \mathbb{R}^{n \times n}$.

The two main bottlenecks often arising with POD-Galerkin ROMs are *(i)* the increasing dimension $n \gg n_\mu + 1$ of the low-dimensional POD subspaces, much larger than the intrinsic dimension of the solution manifold, and *(ii)* the need to rely on hyper-reduction techniques to assemble the operators appearing in the ROM (7) in order not to rely on expensive N_h-dimensional arrays [5].

DL-ROMs have been introduced in [9] to overcome these limitations of POD-Galerkin ROMs, and further applied to cardiac electrophysiology in [10]. A DL-ROM describes both the trial manifold and the reduced dynamics (corresponding to the matrix $\mathbf{V}_n$ and the projection stage, respectively, in the POD-Galerkin case) through deep neural networks, which are trained on a set of FOM snapshots. In this way, DL-ROMs completely avoid the *projection* stage, are non-intrusive, and can be cheaply evaluated once trained. In particular:

- to describe the system dynamics on a suitable reduced nonlinear trial manifold (a task which we refer to as *reduced dynamics learning*), the intrinsic coordinates of the ROM approximation are defined as

$$\mathbf{u}_n(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}) = \boldsymbol{\phi}_n^{DF}(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}), \tag{8}$$

  where $\boldsymbol{\phi}_n^{DF}(\cdot; \cdot, \boldsymbol{\theta}_{DF}) : \mathbb{R}^{(n_\mu+1)} \to \mathbb{R}^n$ is a DFNN, consisting in the subsequent composition of a nonlinear activation function, applied to a linear transformation of the input, multiple times [12]. Here $\boldsymbol{\theta}_{DF}$ denotes the

[1] Sampling frequency in time does not necessarily coincide with the time stepping rule used to discretize in time (1); similarly, singular value decomposition can be done either on $\mathbf{S}$, or following a two-stage procedure, operating SVD across samples in time for each $\boldsymbol{\mu}_i$, $i = 1, \ldots, N_{train}$, and then on the resulting collection of selected singular vectors.

vector of parameters of the DFNN, collecting all the corresponding weights and biases of each layer of the DFNN;

- to model the reduced nonlinear trial manifold $\tilde{\mathcal{S}}_h^n \approx \mathcal{S}_h$ (a task which we refer to as *reduced trial manifold learning*) we employ the decoder function of a CAE [15,16], that is,

$$\tilde{\mathcal{S}}_h^n = \{\mathbf{f}_h^D(\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF});\boldsymbol{\theta}_D) \mid \mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF}) \in \mathbb{R}^n,\ t \in [0,T) \text{ and } \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N_h}, \tag{9}$$

where $\mathbf{f}^D(\cdot;\boldsymbol{\theta}_D) : \mathbb{R}^n \to \mathbb{R}^{N_h}$ denotes the decoder function of a CAE obtained as the composition of several layers (some of which are convolutional), depending upon a vector $\boldsymbol{\theta}_D$ collecting all the corresponding weights and biases.

The DL-ROM approximation $\tilde{\mathbf{u}}_h(t;\boldsymbol{\mu}) \approx \mathbf{u}_h(t;\boldsymbol{\mu})$ is then given by

$$\tilde{\mathbf{u}}_h(t;\boldsymbol{\mu},\theta_{DF},\theta_D) = \mathbf{f}_h^D(\boldsymbol{\phi}_n^{DF}(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF});\boldsymbol{\theta}_D). \tag{10}$$

The encoder function $\mathbf{f}_n^E(\cdot;\boldsymbol{\theta}_E)$ of the CAE, provided when carrying out its training on the FOM snapshots, can then be exploited to map the FOM solution associated to $(t,\boldsymbol{\mu})$ onto a low-dimensional representation

$$\tilde{\mathbf{u}}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_E) = \mathbf{f}_n^E(\mathbf{u}_h(t;\boldsymbol{\mu});\boldsymbol{\theta}_E); \tag{11}$$

$\mathbf{f}_n^E(\cdot;\boldsymbol{\theta}_E) : \mathbb{R}^{N_h} \to \mathbb{R}^n$ denotes the encoder function, depending upon a vector $\boldsymbol{\theta}_E$ of parameters.

Computing the DL-ROM approximation of $\mathbf{u}_h(t;\boldsymbol{\mu}_{test})$, for any possible $t \in (0,T)$ and $\boldsymbol{\mu}_{test} \in \mathcal{P}$, corresponds to the testing stage of a DFNN and of the decoder function of a CAE; this does not require the evaluation of the encoder function. The training stage consists in solving the following optimization problem, in the variable $\boldsymbol{\theta} = (\boldsymbol{\theta}_E, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)$, after the snapshot matrix $\mathbf{S}$ has been formed:

$$\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{N_s} \sum_{i=1}^{N_{train}} \sum_{k=1}^{N_t} \mathcal{L}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}), \tag{12}$$

where $N_s = N_{train} N_t$ and

$$\mathcal{L}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) = \frac{\omega_h}{2}\mathcal{L}_{rec}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) + \frac{1-\omega_h}{2}\mathcal{L}_{int}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}), \tag{13}$$

where

$$\begin{aligned} \mathcal{L}_{rec}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) &= \|\mathbf{u}(t^k;\boldsymbol{\mu}_i) - \tilde{\mathbf{u}}(t^k;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)\|^2, \\ \mathcal{L}_{int}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) &= \|\tilde{\mathbf{u}}_n(t^k;\boldsymbol{\mu}_i,\boldsymbol{\theta}_E) - \mathbf{u}_n(t^k;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF})\|^2 \end{aligned} \tag{14}$$

with $\omega_h \in [0,1]$. The *per-example* loss function (13) combines the reconstruction error (that is, the error between the FOM solution and the DL-ROM approximation) and the error between the intrinsic coordinates and the output of the encoder.

We remark that our choice to provide the output of the DFNN, i.e. $\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF})$, as input to the decoder function $\mathbf{f}_h^D(\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF});\boldsymbol{\theta}_D)$ of the CAE, rather than replacing the DFNN with the encoder function $\mathbf{f}_n^E(\mathbf{u}_h(t;\boldsymbol{\mu});\boldsymbol{\theta}_E)$, is related to the fact that, at testing time, we do not want to feed the FOM solution as input to the DL-ROM. In this way, at testing time the evaluation of the trained network is completely independent from high-fidelity (and high-dimensional) full-order data. Moreover, motivated by the need to discard the encoder function at testing time, we maintain the same structure of the DL-ROM network also at training time, in order to enhance the decoder function robustness, making it stable with respect to possible perturbations affecting the intrinsic coordinates — these latter being the actual input provided to $\mathbf{f}_h^D(\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF});\boldsymbol{\theta}_D)$ at testing time. We also point out that, by exploiting this structure, the updates of the parameters of the DFNN $\boldsymbol{\theta}_{DF}$ depend not only on the gradients of the error between the intrinsic coordinates and the encoder output, but also on the gradients of the reconstruction error.

2.2. The content of this paper and other existing approaches

Several recent works have shown possible applications of DL algorithms for solving PDEs – thanks to their ability of effectively approximating nonlinear maps, and by their ability to learn from data and generalize to unseen data – both from a theoretical [17–19] and a computational standpoint. Regarding this latter, we mention, for instance, physics-informed neural networks (PINNs) [20–23] or physics-informed deep generative models [24,25]. DL algorithms and artificial neural networks (ANN), such as feedforward neural networks, are becoming more and more popular in reduced order modeling, too. In particular:

- ANNs have been employed to model the reduced dynamics in a data-driven and less intrusive way (avoiding, e.g., the costs entailed by the projection stage of projection-based ROMs); for instance, in [26–30] the use of ANNs or Gaussian Processes (GPs) regression models has been proposed to approximate the mapping from the input parameters (and, possibly, time) to the reduced coefficients, as an alternative to the assembly and solution of the reduced order system arising from POD-Galerkin ROMs, however still using a linear trial manifold built through POD; similar strategies have also been introduced in [31–35]. A hybrid strategy proposed in [36] merges ANN-based regression models and PINNs, training the network by minimizing the mean squared residual error of the reduced order equation on a set of points in parameter space; similar results can be found in [37]. Moreover, feedforward and recurrent neural networks have been exploited in [38,39] to address closure problems and model the effects of discarded modes on the set of retained POD modes. Very recently, an ANN-based methodology is proposed to learn mappings between infinite-dimensional spaces for parametric PDEs in [40], exploiting POD and showing mesh-independent properties.
- ANNs have been used to describe the reduced trial manifold where the approximation is sought (thus avoiding the linear superimposition of POD modes), either relying on a minimum residual formulation to derive the ROM [14,41], or without considering an explicit parameter dependence in the differential problem that is considered [42]. For instance, projection-based ROMs are built in [14] by performing a projection of the FOM onto a nonlinear trial manifold identified by means of the decoder function of a CAE – hence, still requiring the assembling and the solution of a ROM as in traditional POD-Galerkin ROMs. The use of CAEs has also been proposed in [42], where a reduced trial manifold is generated through a deep convolutional recurrent AE, which is then used to train a Long Short-Term Memory (LSTM) neural network that models the reduced dynamics.

Our DL-ROM approach [9,10] combines and improves the techniques introduced in [14,42]. As detailed in Section 2.1, the *(i)* nonlinear trial manifold is learnt by using the decoder function of a CAE, while *(ii)* the dynamics on the reduced manifold is modeled through a DFNN and the encoder function of a CAE. These two tasks are achieved simultaneously, by training both the CAE and the DFNN network architectures at the same time, by minimizing a loss function weighting two terms – see (13) – one for each task. The resulting procedure thus avoids both the expensive projection stage of [14] and the training of a more expensive LSTM neural network[2] [42]. Moreover, the DL-ROM technique is purely data-driven, non-intrusive: it only relies on the computation of a set of FOM snapshots. In this respect, DL does not replace the high-fidelity FOM as, e.g., in the works by Karniadakis and coauthors [20–23,43]; rather, DL techniques are built upon it, to enhance the repeated evaluation of the FOM for different values of the parameters. The computational benefits introduced by the use of DL-ROMs can be summarized as follows:

- the dimension of the DL-ROM can be kept extremely small, very close (or even equal) to the dimension of the solution manifold $n_\mu + 1$;
- the DL-ROM can be queried at any desired time instant over the time interval $(0, T)$,[3] and testing-parameter instance over the parameter space $\mathcal{P}$, without requiring the solution of a dynamical system until that time, differently from projection-based ROMs such as, e.g, POD-Galerkin ROMs;
- the time resolution required by the DL-ROM can be chosen to be larger than the one required by the numerical solution of dynamical systems at hand (see, e.g., [10]);
- DL-ROMs avoid the use of (intrusive and very often extremely expensive) hyper-reduction techniques, which are instead required by POD-Galerkin ROMs;
- DL-ROMs can avoid to account for those auxiliary variables of a problem which we might not be interested into (as pressure, compared to velocity, in fluid flow problems, or the gating variables, compared to the electric potential, in cardiac electrophysiology [10]).

[2] LSTMs are neural architectures designed to handle sequential data. Due to the propagation through time, both the forward and the backward passes in LSTMs are, in general, more expensive than training a feedforward/convolutional neural network, as in the proposed work. Indeed, unfolding the LSTM while propagating the gradients during training could entail high computational costs, especially when dealing with long time series, due to the sequential nature of the computations.

[3] A relevant issue is related with the generalization properties of the network outside the time interval where snapshots are sampled. Ensuring good approximation properties when interested in long-time scenarios, even in presence of almost periodic regimes, without more specific network architectures such as LSTMs, which may be computationally expensive and very hard to tune, is an open issue our efforts are focusing on — however, this represents a general aspect shared by several ROM techniques.

For all these reasons, DL-ROMs tremendously improve the computational efficiency of ROMs during the *online* testing stage. However, the *offline* training stage of DL-ROM would still depend on N_h, a fact which ultimately might entail overwhelming training times and costs when N_h is moderately large.

The POD-DL-ROM technique proposed in this work, thanks to a prior dimensionality reduction relying on POD and a suitable multi-fidelity pretraining, greatly enhances the efficiency of the DL-ROM during the training phase, thus dramatically decreasing training computational times, as shown by the numerical results discussed in following Sections.

3. A new deep learning-based reduced order model

POD-DL-ROMs provide a new, general-purpose, ROM approach combining a data dimensionality reduction obtained through POD with the DL-ROM approach [9,10]. After introducing the POD-DL-ROM approach, we discuss in more detail some of its building blocks, the extension to vector problems, finally reporting detailed algorithms for the *offline* (or training) and the *online* query (or testing) stages.

3.1. POD-enhanced DL-ROMs (POD-DL-ROMs)

The POD-DL-ROM technique consists in applying the DL-ROM technique to the intrinsic coordinates of a linear trial manifold generated through randomized singular value decomposition (rSVD) and approximating $\mathcal{S}_h$; alternatively, it can be seen as a ROM technique in which a two-step dimensionality reduction is performed: first, POD (realized through rSVD) is applied on a set of FOM snapshots, then a DL-ROM is built to approximate the map between $(t, \boldsymbol{\mu})$ and the POD generalized coordinates. In this way, all the DL-ROM features allowing its very efficient testing time are retained. As a matter of fact, as shown in Section 2.1, DL-ROMs might imply overwhelming training costs (and times) when the FOM dimension N_h becomes moderately large, although remaining extremely efficient at testing time. We emphasize that very often ROMs are designed to be efficient only regarding their *online* performances, no matter how expensive is the offline stage. Our (more ambitious) goal is instead to realize a ROM able of efficient computational performance during both offline and online stages, compared to classical projection-based ROMs.

Using randomized POD (see Section 3.2), we first build the N-dimensional subspace $\text{Col}(\mathbf{V}_N)$ spanned by the $N \leq N_h$ columns of $\mathbf{V}_N \in \mathbb{R}^{N_h \times N}$, the matrix of the first N singular vectors of the snapshot matrix $\mathbf{S}$. We denote the dimension of the linear manifold by N, to distinguish it from the dimension n of the nonlinear trial manifold, and to emphasize that this dimension can be taken (much) larger with respect to the one of the reduced linear trial manifold that would have been exploited in a POD-Galerkin ROM. Indeed, here linear dimensionality reduction is performed only for the sake of data compression, to avoid to feed training data of dimension N_h.

The POD-DL-ROM approximation $\tilde{\mathbf{u}}_h(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)$ of the FOM solution $\mathbf{u}_h(t; \boldsymbol{\mu})$ is given by

$$\tilde{\mathbf{u}}_h(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D) = \mathbf{V}_N \tilde{\mathbf{u}}_N(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D),$$

that is, it is sought in a linear trial manifold[4] of (potentially large) dimension N,

$$\tilde{\mathcal{S}}_h^{N,lin} = \{\mathbf{V}_N \tilde{\mathbf{u}}_N(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D) \; | \tilde{\mathbf{u}}_N(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D) \in \mathbb{R}^N, \; t \in [0, T) \text{ and } \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^{N_h}, \tag{15}$$

by applying the DL-ROM strategy of Section 2.1 to approximate $\mathbf{V}_N^T \mathbf{u}_h(t; \boldsymbol{\mu})$ – rather than $\mathbf{u}_h(t; \boldsymbol{\mu})$. The DL-ROM approximation $\tilde{\mathbf{u}}_N(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D) \approx \mathbf{V}_N^T \mathbf{u}_h(t; \boldsymbol{\mu})$ takes the form

$$\tilde{\mathbf{u}}_N(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D) = \mathbf{f}_N^D(\boldsymbol{\phi}_n^{DF}(t; \boldsymbol{\mu}, \boldsymbol{\theta}_{DF}); \boldsymbol{\theta}_D), \tag{16}$$

and is sought in a reduced nonlinear trial manifold $\tilde{\mathcal{S}}_N^n$ of dimension $n \ll N$. By adapting the DL-ROM formulation of Section 2.1 to the case at hand, we have that:

[4] Equivalently, we have replaced the original solution manifold $\mathcal{S}_h$ with the N-dimensional linear manifold

$$\mathcal{S}_N^h = \{\mathbf{V}_N^T \mathbf{u}_h(t; \boldsymbol{\mu}) \mid t \in [0, T) \text{ and } \boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^{n_\mu}\} \subset \mathbb{R}^N.$$

As in the case of $\mathcal{S}_h$, the intrinsic dimension of $\mathcal{S}_N^h$ is at most $n_\mu + 1 \ll N$.

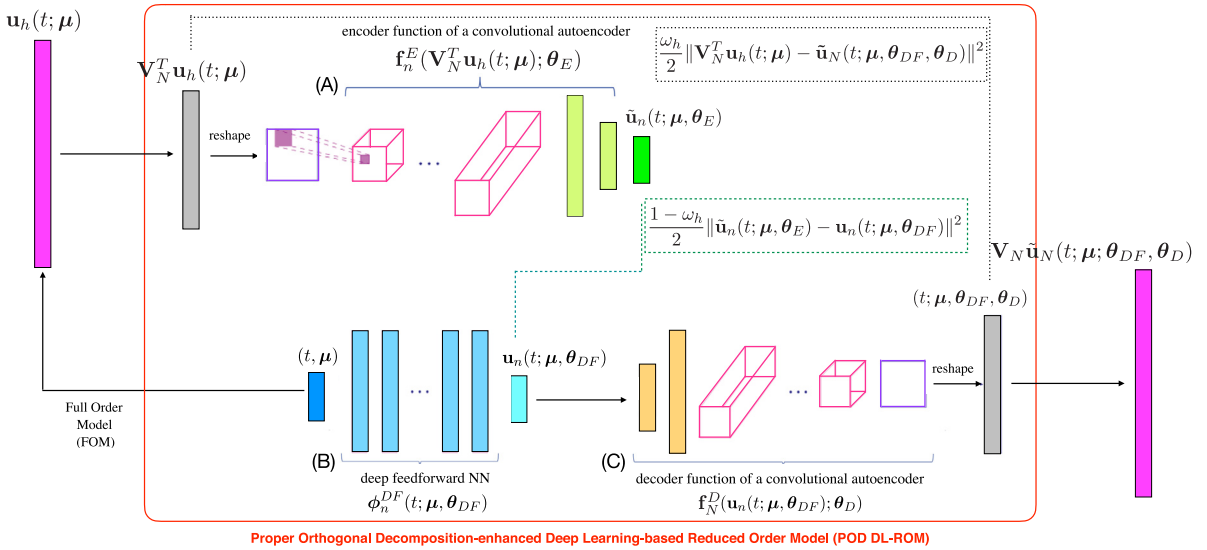

Fig. 1. POD-DL-ROM architecture. Starting from the FOM solution $\mathbf{u}_h(t;\boldsymbol{\mu})$, the intrinsic coordinates $\mathbf{V}_N^T\mathbf{u}_h(t;\boldsymbol{\mu})$ are computed, by means of rSVD, and the neural network provides as output $\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu})$, an approximation of them. The reconstructed solution $\tilde{\mathbf{u}}_h(t;\boldsymbol{\mu})$ is then recovered through the rPOD basis matrix.

- to describe the system dynamics on the nonlinear trial manifold $\tilde{\mathcal{S}}_N^n$ – with n as close as possible to $n_{\boldsymbol{\mu}}+1$ – the intrinsic coordinates of the approximation $\tilde{\mathbf{u}}_N$ are defined as

$$\mathbf{u}_n(t;\boldsymbol{\mu};\boldsymbol{\theta}_{DF}) = \boldsymbol{\phi}_n^{DF}(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF}),$$

  where $\boldsymbol{\phi}_n(\cdot;\cdot,\boldsymbol{\theta}_{DF}) : [0,T)\times\mathbb{R}^{n_\mu+1}\to\mathbb{R}^n$ is a DFNN;
- to model the reduced nonlinear trial manifold $\tilde{\mathcal{S}}_N^n$, we employ the decoder function of a CAE, that is,

$$\begin{aligned}\tilde{\mathcal{S}}_N^n = \{\tilde{\mathbf{u}}_N(t;\boldsymbol{\mu}) = &\mathbf{f}_N^D(\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF});\boldsymbol{\theta}_D) \mid \\ &\mathbf{u}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_{DF})\in\mathbb{R}^n,\ t\in[0,T)\,,\ \boldsymbol{\mu}\in\mathcal{P}\subset\mathbb{R}^{n_\mu}\}\subset\mathbb{R}^N,\end{aligned} \tag{17}$$

  where $\mathbf{f}_N^D(\cdot;\boldsymbol{\theta}_D):\mathbb{R}^n\to\mathbb{R}^N$.

The encoder function of the convolutional AE can then be exploited to map the intrinsic coordinates $\mathbf{V}_N^T\mathbf{u}_h(t,\boldsymbol{\mu})$ associated to $(t,\boldsymbol{\mu})$ onto a low-dimensional representation

$$\tilde{\mathbf{u}}_n(t;\boldsymbol{\mu},\boldsymbol{\theta}_E) = \mathbf{f}_n^E(\mathbf{V}_N^T\mathbf{u}_h(t;\boldsymbol{\mu});\boldsymbol{\theta}_E),$$

where $\mathbf{f}_n^E$ denotes the encoder function, depending upon a vector $\boldsymbol{\theta}_E$ of parameters.

The architecture of the POD-DL-ROM neural network, employed at training time, is the one shown in Fig. 1; note that, at testing time, as in the DL-ROM technique we can discard the encoder function.

Computing the ROM approximation (16) by means of a POD-DL-ROM thus consists in solving the optimization problem (12) where the per-example loss function (13) is now replaced by

$$\begin{aligned}\mathcal{L}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) = &\frac{\omega_h}{2}\|\mathbf{V}_N^T\mathbf{u}_h(t^k;\boldsymbol{\mu}_i)-\tilde{\mathbf{u}}_N(t^k;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)\|^2 \\ &+\frac{1-\omega_h}{2}\|\tilde{\mathbf{u}}_n(t^k;\boldsymbol{\mu}_i,\boldsymbol{\theta}_E)-\mathbf{u}_n(t^k;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF})\|^2,\end{aligned} \tag{18}$$

where the reconstruction error in (14) becomes

$$\mathcal{L}_{rec}(t^k,\boldsymbol{\mu}_i;\boldsymbol{\theta}) = \|\mathbf{V}_N^T\mathbf{u}_h(t^k;\boldsymbol{\mu}_i)-\tilde{\mathbf{u}}_N(t^k;\boldsymbol{\mu}_i,\boldsymbol{\theta}_{DF},\boldsymbol{\theta}_D)\|^2.$$

We point out that the POD-DL-ROM technique relies on a further level of dimensionality reduction compared to [28,30], since the dimension of the reduced linear problem – that is, the number of intrinsic coordinates included in

$\tilde{\mathbf{u}}_N$ – is decreased until (almost, or exactly) matching the intrinsic dimension $n_\mu + 1$ of the parametrized problem. In addition, we introduce the use of convolutional layers, which results better suited to high-dimensional spatial data, thus implying lower computational costs, with respect to dense layers. Moreover, the POD-DL-ROM approach allows to model and approximate the entire intrinsic coordinates vector $\tilde{\mathbf{u}}_N$ all at once, without requiring additional SVDs, if compared to the data-driven RB method, employing GPs as regression models, proposed in [27] and further extended in [29].

We highlight that by shaping the POD-DL-ROM neural network as a zero-extended neural network and assuming that the input–output map is locally Lipschitz, it is possible to prove the convergence of the framework here proposed by following the approach presented in [40].

3.2. (Randomized) POD for dimensionality reduction

Even though POD is not able to generate linear subspaces whose dimension is close to (or matches) the intrinsic dimension of the problems under consideration, it is still able to perform a moderate dimensionality reduction, thus yielding a linear subspace of dimension $N \ll N_h$. However, computing the SVD of the snapshot matrix $\mathbf{S} \in \mathbb{R}^{N_h \times N_{train} N_t}$ can be extremely time consuming for large-scale problems; whenever dealing with FOMs of moderately large dimension N_h and/or a high number of training-parameter or time instances, the computational time and memory required by SVD may become prohibitive, scaling superlinearly in N_h and $N_{train} N_t$ [44]. In order to speed up computations, we rely on the randomized matrix approximation techniques developed in [45]; in particular, we exploit rSVD, which computes an approximated SVD, using randomization.

More precisely, a Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{N_{train} N_t \times m}$ is drawn, where $N \leq m \leq N_{train} N_t$; $m - N$ is called the oversampling parameter. The matrix

$$\mathbf{Y} = (\mathbf{S}\mathbf{S}^T)^q \mathbf{S}\mathbf{\Omega}$$

is then assembled, by normally setting $q = 1$ or 2; then, an iterative QR factorization $\mathbf{Y} = \mathbf{Q}\mathbf{R}$, is computed, where $\mathbf{Q} \in \mathbb{R}^{N_h \times m}$. Computing $\mathbf{Q}$ in such a way consists of applying an adaptive randomized range finder algorithm to approximate the range of $\mathbf{S}$ by means of a matrix $\mathbf{Q}$ whose columns are orthonormal, i.e. $\mathbf{S} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{S}$ [45]. Once the matrix $\mathbf{Q}$ has been computed, it is restricted to the first N columns, denoted by $\mathbf{Q} = \mathbf{Q}(:, 1 : N) \in \mathbb{R}^{N_h \times N}$, and the SVD of the matrix

$$\mathbf{B} = \mathbf{Q}^T\mathbf{S} = \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{Z}} \tag{19}$$

is computed. The SVD factorization of $\mathbf{S}$ is then recovered by setting

$$\mathbf{V}_N = \mathbf{Q}\tilde{\mathbf{V}}. \tag{20}$$

The rSVD approximation can then be computed through the following steps:

1. an approximated basis for Col($\mathbf{S}$), i.e. $\mathbf{Q} \in \mathbb{R}^{N_h \times m}$, is obtained by using randomization,
2. the SVD of the matrix $\mathbf{B} \in \mathbb{R}^{N \times N_{train} N_t}$ in (19) is computed,
3. the matrix $\mathbf{V}_N \in \mathbb{R}^{N_h \times N}$ is recovered by means of $\mathbf{Q}$ as in (20).

In particular, $\mathbf{V}_N$ refers to the rPOD basis matrix. For further details about rSVD we refer to [11,45].

3.3. Pretraining

Directly training a model to solve a specific task can be very demanding if the model is complex and hard to optimize, and/or the task is very difficult. More viable options are either *(i)* to train a simpler model to solve the task, then make the model more complex, or *(ii)* to train the model to solve a simpler task, then move on to the final task. Both these strategies are known in the DL literature as *pretraining* [12]. In particular, a finer tuning of a pretrained model is equivalent to *transfer learning*, if the data used to perform fine tuning are of different nature with respect to the data used during pretraining. Pretraining can then be seen as a form of transfer learning, where a pretrained model is used as initial state of the network [46]; this strategy works extremely well in many objects classification tasks [47] and natural language processing problems [48]. In the area of scientific ML, pretraining has been used, for instance, in [49] where a pretrained neural network has been used to perform parameter identification on a new dataset.

Relying on a suitable pretraining, we are able to further enhance the training phase of a POD-DL-ROM, combining models of different fidelities (e.g., by considering coarser/finer spatial discretizations, as well as different physical laws, more/less parameters or larger/smaller parameter ranges). In particular, we train the POD-DL-ROM neural network on an initial simpler task, obtaining a set of parameters $\boldsymbol{\theta}_S^* = \{(\mathbf{W}_{S,i}^*, \mathbf{b}_{S,i}^*)\}_{i=1}^L$, and then use them to initialize the training of the POD-DL-ROM neural network on a more complex problem, by setting $\boldsymbol{\theta}_C^0 = \boldsymbol{\theta}_S^*$. Numerical results of Section 4 will show how pretraining, combined with the dimensionality reduction obtained through rPOD, represents a cornerstone in view of drastically reducing the training computational time of a POD-DL-ROM, if compared to the time required for training, from scratch, a neural network on the more complex task.

3.4. Extension to vector problems

Compared to the DL-ROM technique, applied so far to scalar problems only, we have further generalized the POD-DL-ROM technique in order to handle vector problems, in analogy to what happens when treating *red-green-blue* (RGB) images in general. DL algorithms. By considering the spatial discretization of a vector PDE problem, whose solution is a d-dimensional vector field, problem (1) can be rewritten as

$$\begin{cases} \dot{\mathbf{u}}_h^1(t;\boldsymbol{\mu}) = \mathbf{f}^1(t, \mathbf{u}_h^1(t;\boldsymbol{\mu}), \dots, \mathbf{u}_h^d(t;\boldsymbol{\mu});\boldsymbol{\mu}), \qquad t \in (0,T), \\ \vdots \\ \dot{\mathbf{u}}_h^d(t;\boldsymbol{\mu}) = \mathbf{f}^d(t, \mathbf{u}_h^1(t;\boldsymbol{\mu}), \dots, \mathbf{u}_h^d(t;\boldsymbol{\mu});\boldsymbol{\mu}), \qquad t \in (0,T), \\ \mathbf{u}_h^1(0;\boldsymbol{\mu}) = \mathbf{u}_0^1(\boldsymbol{\mu}), \\ \vdots \\ \mathbf{u}_h^d(0;\boldsymbol{\mu}) = \mathbf{u}_0^d(\boldsymbol{\mu}), \end{cases} \tag{21}$$

where $\mathbf{u}_h^i : [0,T) \times \mathcal{P} \to \mathbb{R}^{N_h^i}$ is the solution of the ith equation in (21), $\mathbf{u}_0^i : \mathcal{P} \to \mathbb{R}^{N_h^i}$ is the ith initial datum, $\mathbf{f}^i : (0,T) \times \mathbb{R}^{N_h^i} \times \mathcal{P} \to \mathbb{R}^{N_h^i}$ describes the dynamics of $\mathbf{u}_h^i(t;\boldsymbol{\mu})$, $i = 1, \dots, d$, with $d = 2, 3$. Depending on the problem at hand, some of the equations appearing in (21) might not involve the derivatives and related initial conditions; this is what happens, for instance, in the case of unsteady Navier–Stokes equations for incompressible flows, where the equation expressing flow incompressibility involves the velocity components, but no time derivatives.

Provided the solution of (21), associated to a particular instance $(t, \boldsymbol{\mu})$, and the orthonormal basis $\mathbf{V}_{N,i} \in \mathbb{R}^{N_h^i \times N}$, with $i = 1, \dots, d$, found though rSVD, we compute the intrinsic components $\mathbf{V}_{N,1}^T \mathbf{u}_h^1(t;\boldsymbol{\mu}), \dots, \mathbf{V}_{N,d}^T \mathbf{u}_h^d(t;\boldsymbol{\mu})$, reshape each component in a square matrix of dimension $(\sqrt{N}, \sqrt{N})$, where $N = 2^{(2m)}$ with $m \in \mathbb{N}$, and stack them together forming a tensor with d *channels*. Thus, each vectorial component of the solution of problem (21) is reshaped in a square matrix; then, they are stacked together forming a tensor of dimension $(\sqrt{N}, \sqrt{N}, 3)$.

This approach allows the dimensions N_h^i, $i = 1, \dots, d$, of each FOM component, to be different. Indeed, it is the rPOD dimension N used to reduce each vector component that must be kept equal for $i = 1, \dots, d$. We remark that by stacking all components together allows to reduce the number of parameters, and then the training and testing computational times of POD-DL-ROM.

3.5. Training and testing algorithms

We summarize in this section both the training and the testing stage of the POD-DL-ROM technique. Regarding the setting of the optimization algorithm, the way to select the hyperparameters and the architecture of the neural networks, we refer to [9,10]. We denote by $\mathbf{M} \in \mathbb{R}^{(n_{\boldsymbol{\mu}}+1) \times N_s}$ the matrix collecting all the parameter instances corresponding to the computed snapshots; these latter are included in the snapshot matrix $\mathbf{S}$, defined in (5). Note that, in the case of a vector problem, the snapshot matrix $\mathbf{S} \in \mathbb{R}^{\sum_i N_h^i \times N_s}$ takes the form

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_1 \\ \dots \\ \mathbf{S}_d \end{bmatrix} = \begin{bmatrix} \mathbf{u}_h^1(t^1;\boldsymbol{\mu}_1) \mid \dots \mid \mathbf{u}_h^1(t^{N_t};\boldsymbol{\mu}_1) \mid \dots \mid \mathbf{u}_h^1(t^1;\boldsymbol{\mu}_{N_{train}}) \mid \dots \mid \mathbf{u}_h^1(t^{N_t};\boldsymbol{\mu}_{N_{train}}) \\ \dots \\ \mathbf{u}_h^d(t^1;\boldsymbol{\mu}_1) \mid \dots \mid \mathbf{u}_h^d(t^{N_t};\boldsymbol{\mu}_1) \mid \dots \mid \mathbf{u}_h^d(t^1;\boldsymbol{\mu}_{N_{train}}) \mid \dots \mid \mathbf{u}_h^d(t^{N_t};\boldsymbol{\mu}_{N_{train}}) \end{bmatrix}.$$

The input and the output of the POD-DL-ROM are normalized by applying to each channel of the d-dimensional tensor $\mathbf{S}$ the affine transformation detailed below. After splitting the data in $\mathbf{M} = [\mathbf{M}^{train}, \mathbf{M}^{val}]$ and $\mathbf{S} = [\mathbf{S}^{train}, \mathbf{S}^{val}]$ – where $\mathbf{M}^{val}, \mathbf{S}^{val} \in \mathbb{R}^{\sum_i N_h^i \times \alpha N_s}$, and α is a user-defined training-validation splitting fraction – we define

$$M^i_{max} = \max_{j=1,\dots,N_s} M^{train}_{ij}, \qquad M^i_{min} = \min_{j=1,\dots,N_s} M^{train}_{ij}, \tag{22}$$

so that parameters are normalized by applying the following transformation

$$M^{train}_{ij} \mapsto \frac{M^{train}_{ij} - M^i_{max}}{M^i_{max} - M^i_{min}}, \qquad i = 1 \dots, n_{\boldsymbol{\mu}} + 1, \;\; j = 1, \dots, N_s, \tag{23}$$

that is, each feature of the training parameter matrix is rescaled according to its maximum and minimum values. Regarding instead the training snapshot matrix $\mathbf{S}^{train} \in \mathbb{R}^{\sum_i N_h^i \times N_s}$, we define

$$S^k_{max} = \max_{i=1,\dots,N_h} \max_{j=1,\dots,N_s} S^{train}_{ij}, \qquad S^k_{min} = \min_{i=1,\dots,N_h} \min_{j=1,\dots,N_s} S^{train}_{ij}, \qquad k = 1, \dots, d \tag{24}$$

and apply transformation (23), by replacing M^i_{max}, M^i_{min} with $S_{max}, S_{min} \in \mathbb{R}$ respectively, to each channel of $\mathbf{S}$ – that is, we use the same maximum and minimum values for all the features of the snapshot matrix, as in [14,42]. Using the latter approach or employing each feature's maximum and minimum values, for the matrix $\mathbf{S}^{train}$, does not lead to remarkable changes in the POD-DL-ROM performance. Transformation (23) is applied also to the validation and testing sets, but considering as maximum and minimum the values computed over the training set. In order to rescale the reconstructed solution to the original values, we apply the inverse transformation of (23).

We detail the algorithms through which the training and the testing of the neural network are performed in Algorithms 1 and 2. During the training phase, the optimal parameters of the POD-DL-ROM are found by solving the optimization problem (12)–(18) through the back-propagation and ADAM algorithms (see Algorithm 1). At testing time, the encoder function is instead discarded (see Algorithm 2). By exploiting an early stopping criterion, we stop the training if the loss function does not decrease over a certain number of epochs over the validation set.

We remark that with $\widetilde{\mathbf{S}}_n$ we refer to a matrix collecting in its columns the output of the encoder function of the convolutional AE applied to each column of the snapshot matrix $\mathbf{S}$. In the same way, the columns of $\mathbf{S}_n$ collect the minimal coordinates, output of the DFNN, for each sample in the parameter matrix $\mathbf{M}$, and $\widetilde{\mathbf{S}}_h$ is a matrix whose columns are the intrinsic coordinates approximations, outputs of the decoder function of the convolutional AE, associated to the columns of $\mathbf{S}_n$.

4. Numerical results

We assess the numerical performance of the proposed POD-DL-ROM technique, by focusing on the training and testing computational times required to construct and deploy a POD-DL-ROM, and the use of pretraining, on four different linear or nonlinear parametrized PDE problems: *(i)* a linear unsteady advection–diffusion–reaction equation; *(ii)* the monodomain system for cardiac electrophysiology; *(iii)* a nonlinear elastodynamics problem, and *(iv)* the unsteady Navier–Stokes equations for incompressible flows.

To evaluate the performance of the POD-DL-ROM technique, we rely on the following error indicators:

- the error indicator $\epsilon_{rel} \in \mathbb{R}$ given by

$$\epsilon_{rel}(\mathbf{u}_h, \tilde{\mathbf{u}}_h) = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \left(\frac{\sqrt{\sum_{k=1}^{N_t} \|\mathbf{u}^k_h(\boldsymbol{\mu}_{test,i}) - \tilde{\mathbf{u}}^k_h(\boldsymbol{\mu}_{test,i})\|^2}}{\sqrt{\sum_{k=1}^{N_t} \|\mathbf{u}^k_h(\boldsymbol{\mu}_{test,i})\|^2}} \right), \tag{25}$$

- the maximum error indicator $\epsilon_{max} \in \mathbb{R}$ provided by

$$\epsilon_{max}(\mathbf{u}_h, \tilde{\mathbf{u}}_h) = \max_{i \in \{1,\dots,N_{test}\}} \frac{\sqrt{\sum_{k=1}^{N_t} \|\mathbf{u}^k_h(\boldsymbol{\mu}_{test,i}) - \tilde{\mathbf{u}}^k_h(\boldsymbol{\mu}_{test,i})\|^2}}{\sqrt{\sum_{k=1}^{N_t} \|\mathbf{u}^k_h(\boldsymbol{\mu}_{test,i})\|^2}}, \tag{26}$$

Algorithm 1 POD-DL-ROM training algorithm

Input: Parameter matrix $\mathbf{M} \in \mathbb{R}^{(n_{\boldsymbol{\mu}}+1)\times N_s}$, snapshot matrix $\mathbf{S} \in \mathbb{R}^{\sum_i N_h^i \times N_s}$, training-validation splitting fraction α, starting learning rate η, batch size N_b, maximum number of epochs N_{epochs}, number of minibatches $N_{mb} = (1-\alpha)N_s/N_b$.
Output: Optimal model parameters $\boldsymbol{\theta}^* = (\boldsymbol{\theta}_E^*, \boldsymbol{\theta}_{DF}^*, \boldsymbol{\theta}_D^*)$.

Compute rPOD basis matrix $\mathbf{V}_N = [\mathbf{V}_{N,1}|\dots|\mathbf{V}_{N,d}]^T$
Randomly shuffle $\mathbf{M}$ and $\mathbf{S}$
Split data in $\mathbf{M} = [\mathbf{M}^{train}, \mathbf{M}^{val}]$ and $\mathbf{S} = [\mathbf{S}^{train}, \mathbf{S}^{val}]$ (with $\mathbf{M}^{val}, \mathbf{S}^{val} \in \mathbb{R}^{\sum_i N_h^i \times \alpha N_s}$)
Compute intrinsic coordinates $\mathbf{S}_N^{train} = [\mathbf{S}_1^{train}|\dots|\mathbf{S}_d^{train}]^T$ where $\mathbf{S}_i^{train} = \mathbf{V}_{N,i}^T \mathbf{S}_i^{train}$, $i = 1, \dots, d$
Compute intrinsic coordinates $\mathbf{S}_N^{val} = [\mathbf{S}_1^{val}|\dots|\mathbf{S}_d^{val}]^T$ where $\mathbf{S}_i^{val} = \mathbf{V}_{N,i}^T \mathbf{S}_i^{val}$, $i = 1, \dots, d$
Normalize data in $\mathbf{M}$ and $\mathbf{S}_N = [\mathbf{S}_N^{train}, \mathbf{S}_N^{val}]$
Randomly initialize $\boldsymbol{\theta}^0 = (\boldsymbol{\theta}_E^0, \boldsymbol{\theta}_{DF}^0, \boldsymbol{\theta}_D^0)$
$n_e = 0$
while (¬early-stopping **and** $n_e \leq N_{epochs}$) **do**
  for $k = 1 : N_{mb}$ **do**
    Sample a minibatch $(\mathbf{M}^{batch}, \mathbf{S}_N^{batch}) \subseteq (\mathbf{M}^{train}, \mathbf{S}_N^{train})$
    $\mathbf{S}_N^{batch} = \text{reshape}(\mathbf{S}_N^{batch}) \in \mathbb{R}^{N_b \times \sqrt{N} \times \sqrt{N} \times d}$
    $\widetilde{\mathbf{S}}_n^{batch}(\boldsymbol{\theta}_E^{N_{mb}n_e+k}) = \mathbf{f}_n^E(\mathbf{S}_N^{batch}; \boldsymbol{\theta}_E^{N_{mb}n_e+k})$
    $\mathbf{S}_n^{batch}(\boldsymbol{\theta}_{DF}^{N_{mb}n_e+k}) = \boldsymbol{\phi}_n^{DF}(\mathbf{M}^{batch}; \boldsymbol{\theta}_{DF}^{N_{mb}n_e+k})$
    $\widetilde{\mathbf{S}}_N^{batch}(\boldsymbol{\theta}_{DF}^{N_{mb}n_e+k}, \boldsymbol{\theta}_D^{N_{mb}n_e+k}) = \mathbf{f}_N^D(\mathbf{S}_n^{batch}(\boldsymbol{\theta}_{DF}^{N_{mb}n_e+k}); \boldsymbol{\theta}_D^{N_{mb}n_e+k})$
    $\widetilde{\mathbf{S}}_N^{batch} = \text{reshape}(\widetilde{\mathbf{S}}_N^{batch}) \in \mathbb{R}^{N_b \times N \times d}$
    Accumulate loss (18) on $(\mathbf{M}^{batch}, \mathbf{S}_N^{batch})$ and compute $\widehat{\nabla}_\theta \mathcal{J}$
    $\boldsymbol{\theta}^{N_{mb}n_e+k+1} = \text{ADAM}(\eta, \widehat{\nabla}_\theta \mathcal{J}, \boldsymbol{\theta}^{N_{mb}n_e+k})$
  end for
  Repeat instructions 12–16 on $(\mathbf{M}^{val}, \mathbf{S}_N^{val})$ with the updated weights $\boldsymbol{\theta}^{N_{mb}n_e+k+1}$
  Accumulate loss (18) on $(\mathbf{M}^{val}, \mathbf{S}_N^{val})$ to evaluate early-stopping criterion
  $n_e = n_e + 1$
end while

Algorithm 2 POD-DL-ROM testing algorithm

Input: Testing parameter matrix $\mathbf{M}^{test} \in \mathbb{R}^{(n_{\boldsymbol{\mu}}+1)\times(N_{test}N_t)}$, rPOD basis matrix $\mathbf{V}_N$, $(\boldsymbol{\theta}_{DF}^*, \boldsymbol{\theta}_D^*)$.
Output: ROM approximation matrix $\widetilde{\mathbf{S}}_h \in \mathbb{R}^{\sum_i N_h^i \times (N_{test}N_t)}$.

Load $\boldsymbol{\theta}_{DF}^*$ and $\boldsymbol{\theta}_D^*$
$\mathbf{S}_n(\boldsymbol{\theta}_{DF}^*) = \boldsymbol{\phi}_n^{DF}(\mathbf{M}^{test}; \boldsymbol{\theta}_{DF}^*)$
$\widetilde{\mathbf{S}}_N(\boldsymbol{\theta}_{DF}^*, \boldsymbol{\theta}_D^*) = \mathbf{f}_N^D(\mathbf{S}_n(\boldsymbol{\theta}_{DF}^*); \boldsymbol{\theta}_D^*)$
$\widetilde{\mathbf{S}}_N = \text{reshape}(\widetilde{\mathbf{S}}_N)$
$\widetilde{\mathbf{S}}_h = \mathbf{V}_N \widetilde{\mathbf{S}}_N$

- the relative error $\boldsymbol{\epsilon}_k \in \mathbb{R}^{\sum_i N_h^i}$, for $k = 1, \dots, N_t$, defined as

$$\boldsymbol{\epsilon}_k(\mathbf{u}_h, \tilde{\mathbf{u}}_h) = \frac{|\mathbf{u}_h^k(\boldsymbol{\mu}_{test}) - \tilde{\mathbf{u}}_h^k(\boldsymbol{\mu}_{test})|}{\sqrt{\frac{1}{N_t}\sum_{k=1}^{N_t} \|\mathbf{u}_h^k(\boldsymbol{\mu}_{test})\|^2}}. \tag{27}$$

The coefficient $\omega_h \in [0, 1]$ in (18) is set equal to 0.5 according to the results shown in [9], where a detailed analysis suggested to select values of ω_h equidistant from the extrema of [0, 1]. The rPOD dimension N is selected, in all test cases, in such a way that $\epsilon_{rel}(\mathbf{u}_h, \mathbf{V}_N\mathbf{V}_N^T\mathbf{u}_h) \approx 10^{-4}$, whereas the dimension of the nonlinear trial manifold

n is set trying to match the dimension of the solution manifold $n_\mu + 1$. The POD-DL-ROM neural network is implemented by means of the `Tensorflow` DL framework [50].

4.1. Test 1: unsteady advection–diffusion–reaction equation

The first test case we consider deals with the solution $u = u(\mathbf{x}, t; \boldsymbol{\mu})$ of the following advection–diffusion–reaction system

$$\begin{cases} \dfrac{\partial u}{\partial t} - \text{div}(\mu_1 \nabla u) + \mathbf{b}(t; \mu_2) \cdot \nabla u + cu = f(\mu_3, \mu_4) & (\mathbf{x}, t) \in \Omega \times (0, T), \\ \mu_1 \nabla u \cdot \mathbf{n} = 0 & (\mathbf{x}, t) \in \partial\Omega \times (0, T), \\ u(0) = 0 & \mathbf{x} \in \Omega, \end{cases} \tag{28}$$

in the two-dimensional domain $\Omega = (0, 1)^2$, where

$$f(\mathbf{x}; \mu_3, \mu_4) = 10 \exp(-((x - \mu_3)^2 + (y - \mu_4)^2)/0.07^2)$$

and

$$\mathbf{b}(t; \mu_2) = [\cos(\pi/\mu_2 t), \sin(\pi/\mu_2 t)]^T.$$

We consider $n_\mu = 4$ parameters, belonging to $\mathcal{P} = [0.002, 0.005] \times [30, 70] \times [0.4, 0.6]^2$; we build a FOM considering a space discretization made by linear ($\mathbb{P}_1$) finite elements, considering $N_h = 10\,657$ DOFs, and a Backward Differentiation Formula (BDF) of order 2 considering a time step $\Delta t = 2\pi/20$ over $(0, T)$ with $T = 10\pi$, as time discretization. For different values of μ_3 and μ_4, the solution of (28) exhibits different patterns, due to the location of the distributed source; the dependence on μ_1 and μ_2 impact instead on the relative importance of diffusion and advection terms, and on the direction of this latter. We expect that, for the case at hand, POD-Galerkin ROMs might involve a large number of basis functions, also because; note also that the dependence of the solution on μ_3 and μ_4 is nonlinear (that is, the problem is nonaffinely parametrized).

Regarding the construction of the proposed POD-DL-ROM, for the training of the neural networks, we consider $N_t = 100$ time instances and $N_{train} = 5 \times 5 \times 5 \times 4 = 500$ training-parameter instances, uniformly distributed in each parametric direction. At testing phase, $N_{test} = 4 \times 4 \times 4 \times 3 = 192$ testing-parameter instances have been considered instead, different from the training ones. The maximum number of epochs is set equal to $N_{epochs} = 10\,000$, the batch size is $N_b = 120$ and, regarding the early-stopping criterion, we stop the training if the loss function does not decrease within 500 epochs. We set $N = 64$ as dimension of the rPOD basis (i.e. the linear trial manifold generated by means of rSVD), and $n = n_\mu + 1 = 5$ as dimension of the reduced nonlinear trial manifold. The training and testing phases of the POD-DL-ROM neural network have been performed on a Tesla V100 32 GB GPU.

In Fig. 2 we show the FOM and the POD-DL-ROM solutions, for the testing-parameter instances $\boldsymbol{\mu}_{test} = (0.425, 0.425, 35, 0.0045)$ and $\boldsymbol{\mu}_{test} = (0.575, 0.475, 45, 0.0045)$ at $t = 29.53$, respectively, together with the relative error (27). The error indicator (25) is equal to $\epsilon_{rel} = 1.40 \times 10^{-2}$, while the maximum error indicator (26) is given by $\epsilon_{max} = 1.95 \times 10^{-2}$, thus showing that the level of accuracy achieved by the POD-DL-ROM is preserved over the entire testing set (hence over the whole parameter space $\mathcal{P}$). Moreover, we compute the value of (25) on the training set, which is $\epsilon_{rel} = 1.02 \times 10^{-2}$, in order to assess the ability of the POD-DL-ROM neural network to generalize to new, unseen data and learn the actual distribution of the parameter-dependent data seen during the training phase.

The comparison between some components of the intrinsic coordinates vector $\mathbf{V}_N^T \mathbf{u}_h(t; \boldsymbol{\mu}_{test})$ and their POD-DL-ROM approximation, for the testing parameter instance $\boldsymbol{\mu}_{test} = (0.575, 0.475, 45, 0.0045)$, is shown is Fig. 3. We remark that, as expected, the first components are the ones retaining most of the energy of the system; thus being the ones with higher magnitude [2].

In Fig. 3 (right) we show the CPU times, as function of N, required by (exact) SVD and rSVD to compute the linear POD space, and thus performing the first level of dimensionality reduction required by the POD-DL-ROM. The CPU time required by SVD is not affected by N, while the one required by rSVD increases with respect to N – note that the two times almost coincide for $N = 4096$, a dimension for which constructing a ROM could in principle be avoided. Hence, rSVD is always preferable with respect to SVD; in particular, for the choice $N = 64$, using rSVD allows a speed-up equal to 32 with respect to SVD.

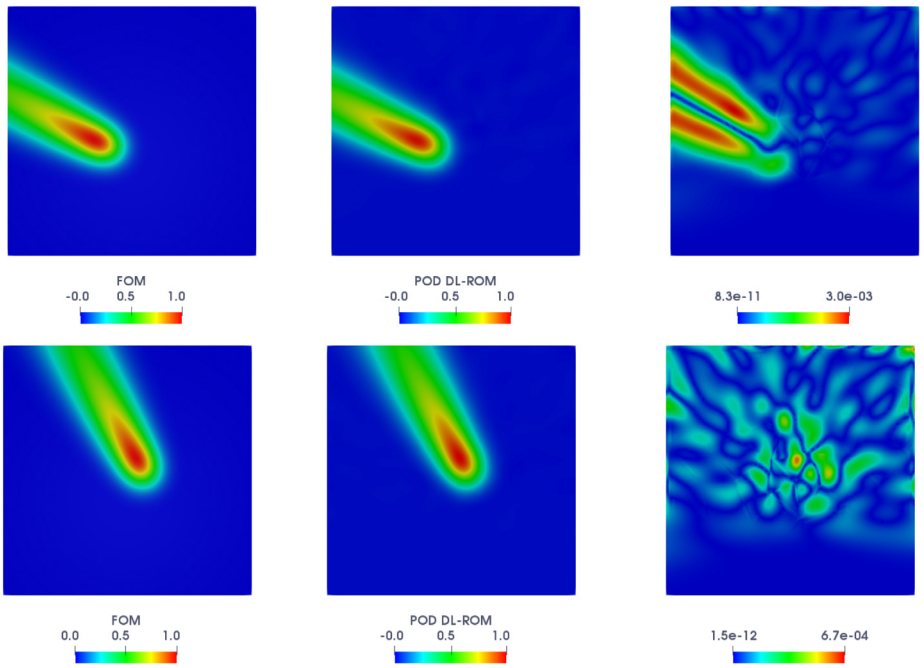

Fig. 2. *Test 1*: FOM (left), POD-DL-ROM (center), with $n = 5$ and $N = 64$, solutions and relative error $\boldsymbol{\epsilon}_k$ (right), for the testing-parameter instances $\boldsymbol{\mu}_{test} = (0.425, 0.425, 35, 0.0045)$ (top) and $\boldsymbol{\mu}_{test} = (0.575, 0.475, 45, 0.0045)$ (bottom) at $t = 29.53$.

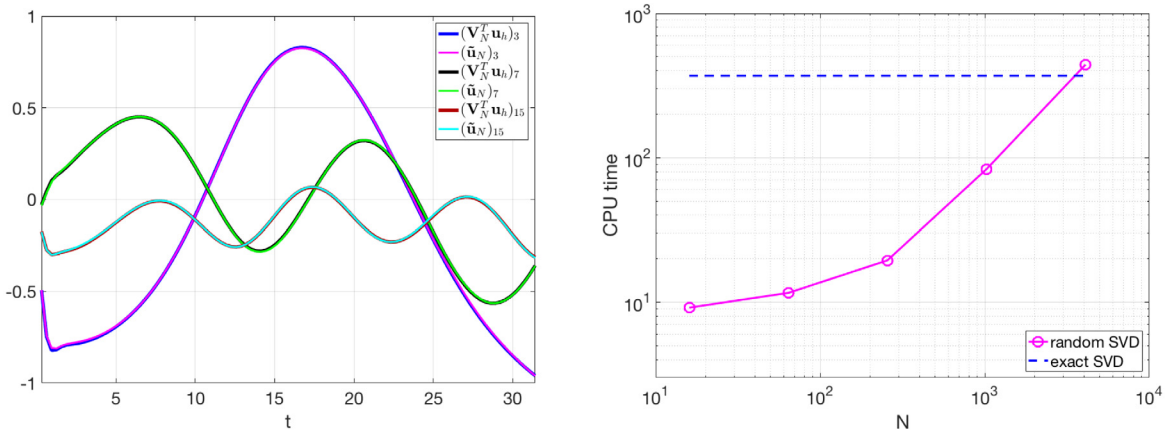

Fig. 3. *Test 1*. Left: comparison between the intrinsic coordinates $\mathbf{V}_N^T\mathbf{u}_h$ components and the POD-DL-ROM approximation $\tilde{\mathbf{u}}_N$ for the testing-parameter instance $\boldsymbol{\mu}_{test} = (0.575, 0.475, 45, 0.0045)$. Right: SVD and rSVD CPU times vs. N.

The trend of the relative error (27) over time, for the selected testing-parameter instance $\boldsymbol{\mu}_{test} = (0.575, 0.475, 45, 0.0045)$, is displayed in Fig. 4 (left), where the mean (over the domain), the median, the first and third quartile of the relative error, as well as its minimum, are reported. The interquartile range (IQR) shows that the distribution of the error is almost uniform over time. Indeed, the snapshots related to different time instances are treated as independent by a POD-DL-ROM, thus preventing errors to accumulate over time, and possible stability issues to occur.

We also analyze the convergence properties of the POD-DL-ROM by varying the number of training-parameter instances provided to the neural network. In particular, in Fig. 4 (right) we report the trend of the error indicator (25), over the testing set, versus N_{train}, i.e. the size of the training dataset. With ϵ_{rel}^{3723} we refer to the value of the error indicator obtained by setting the maximum number of epochs equal to $N_{epochs} = 3723$, which are the iterations

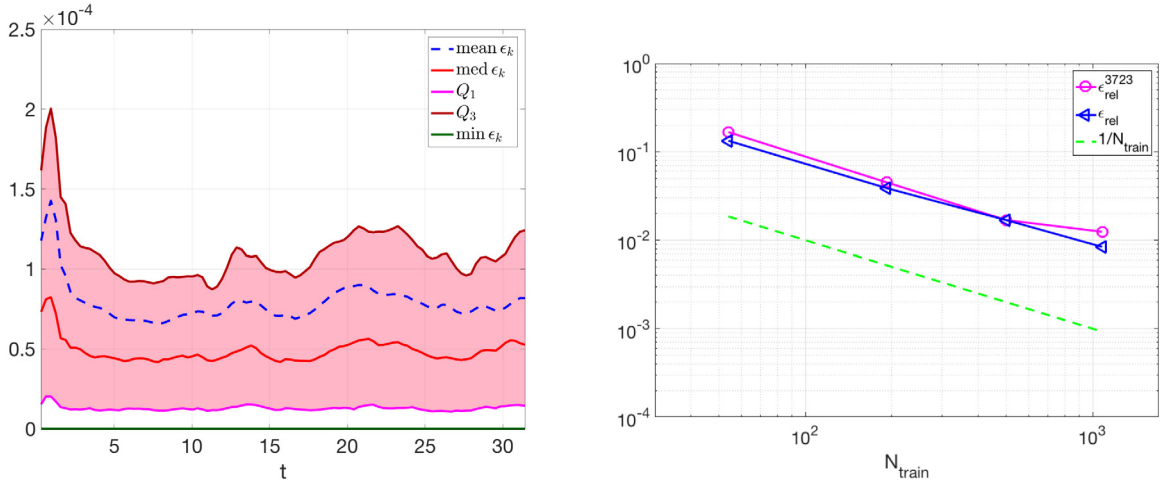

Fig. 4. *Test 1.* Left: trend of the relative error over time for $\boldsymbol{\mu}_{test} = (0.575, 0.475, 45, 0.0045)$. Right: error indicator ϵ_{rel} vs. N_{train}.

performed during the training phase by considering $N_{train} = 500$. By increasing N_{train}, that is by providing more data to the POD DL-ROM neural network, its approximation capability increases, thus yielding a decrease in the error indicator. In particular, the error indicator (25) decays with a rate that is about $1/N_{train}$.

We finally assess the accuracy and the efficiency of the POD-DL-ROM with respect to the dimension N of the POD space. In Fig. 5 (left) we display the error indicator (25) computed on the FOM and POD-DL-ROM solutions, on the FOM and the optimal-POD (i.e., the projection of the FOM solution onto the linear trial manifold generated through rSVD) solutions, and the intrinsic coordinates $\mathbf{V}_N^T\mathbf{u}_h$ and the approximated ones $\tilde{\mathbf{u}}_N$. The reconstruction error consists of two components: the rPOD projection error and the error generated by the neural network when approximating the intrinsic coordinates; indeed, at the same extent, the following relation holds $\epsilon_{rel}(\mathbf{u}_h, \tilde{\mathbf{u}}_h) \leq \epsilon_{rel}(\mathbf{V}_N^T\mathbf{u}_h, \tilde{\mathbf{u}}_N) + \epsilon_{rel}(\mathbf{u}_h, \mathbf{V}_N\mathbf{V}_N^T\mathbf{u}_h)$. The value of $\epsilon_{rel}(\mathbf{u}_h, \tilde{\mathbf{u}}_h)$, for $N = 16$, is dictated by the projection error $\epsilon_{rel}(\mathbf{u}_h, \mathbf{V}_N\mathbf{V}_N^T\mathbf{u}_h)$, thus indicating that, for $N = 16$, the number of rPOD basis selected is too small in order to accurately reconstruct the FOM solution, that is the linear trial manifold generated through rSVD is not able to approximate the solution manifold with sufficient accuracy. For $N = 64$, by considering a larger number of rPOD basis functions, the reconstruction error, completely determined by $\epsilon_{rel}(\mathbf{V}_N^T\mathbf{u}_h, \tilde{\mathbf{u}}_N)$ in this case, decreases. Instead, by setting the rPOD dimension $N > 64$, there is a mild improvement of the POD-DL-ROM performance, i.e. the error indicator $\epsilon_{rel}(\mathbf{u}_h, \tilde{\mathbf{u}}_h)$, remains almost the same. Such an improvement is not significant, in general due to the fact that, by increasing N, the number of parameters of the DL-ROM neural network slightly increases, thus implying almost the same approximation capability.

The GPU training and testing computational times versus N are pointed out in Fig. 5 (right). The training time refers to the total time required for the training and validation phases; for the sake of completeness we also show the number of epochs n_e along with N. The training time varies between 2 h 30 m and 4 h 40 m, we remark that we deal with $N_{train} = 500$ training-parameter instances. The testing time consists instead in the time required to compute N_t time instances for a testing-parameter instance. The trend is proportional to $N^{1/2}$ and, for example, for $N = 64$ the testing time is equal to 4.2×10^{-3} s thus leading to a speed-up 1.2×10^4 with respect to the solution of the FOM on a MacBook Pro Intel Core i7 6-core with 16 GB RAM.

Finally, we compare the DL-ROM and POD-DL-ROM GPU total and testing computational times, for $N = 64$ and the same level of accuracy, in Table 1. The use of the POD-enhanced DL-ROM enables to decrease the training and validation time of a factor 9.

4.2. Test 2: coupled PDE-ODE Monodomain/Aliev–Panfilov system

We now consider a coupled PDE–ODE nonlinear system modeling the electrical behavior of the cardiac tissue, from the cellular scale to the tissue level: the Monodomain equation [51] coupled with the Aliev–Panfilov ionic

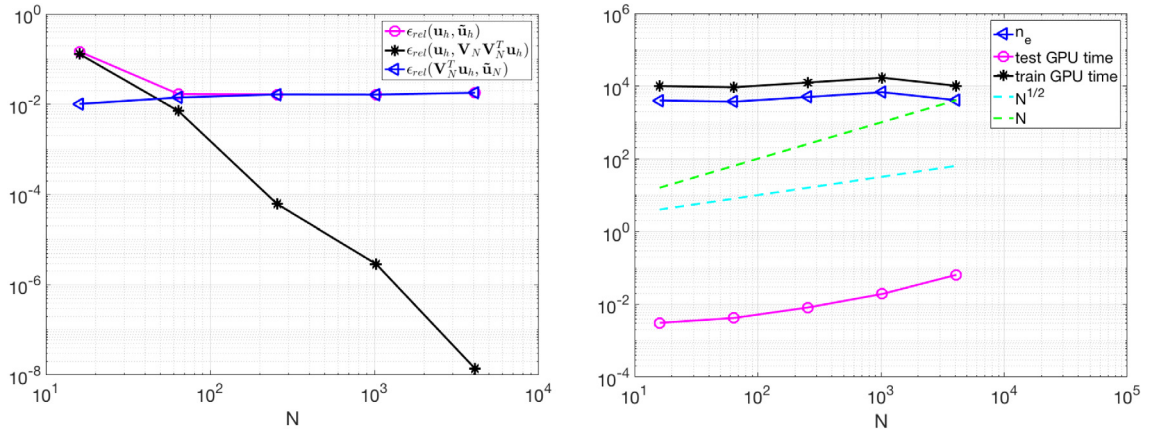

Fig. 5. *Test 1*: Left: Error indicator ϵ_{rel} vs. N. Right: GPU training and testing computational times vs. N.

Table 1
Test 1: DL-ROM and POD-DL-ROM GPU total and testing times.

	Total time	Test [s]
DL-ROM	23.3 h	0.035
POD-DL-ROM	2.5 h	0.004

model [52], in a square slab of tissue $\Omega = (0, 10)^2$ cm:

$$\begin{cases} \dfrac{\partial u}{\partial t} - \text{div}(\mathbf{D}\nabla u) + Ku(u-a)(u-1) + uw = I_{app}(\mathbf{x}, t) & (\mathbf{x}, t) \in \Omega \times (0, T), \\ \dfrac{\partial w}{\partial t} + \left(\epsilon_0 + \dfrac{c_1 w}{c_2 + u}\right)(-w - Ku(u-b-1)) = 0 & (\mathbf{x}, t) \in \Omega \times (0, T), \\ \nabla u \cdot \mathbf{n} = 0 & (\mathbf{x}, t) \in \partial\Omega \times (0, T), \\ u(\mathbf{x}, 0) = 0, \;\; w(\mathbf{x}, 0) = 0 & \mathbf{x} \in \Omega. \end{cases} \tag{29}$$

We consider two ($n_\mu = 2$) parameters, consisting in the electric conductivities in the longitudinal and the transversal directions to the fibers, i.e., the conductivity tensor $\mathbf{D}(\mathbf{x}; \boldsymbol{\mu})$ takes the form

$$\mathbf{D}(\mathbf{x}; \boldsymbol{\mu}) = \mu_2 I + (\mu_1 - \mu_2)\mathbf{f}_0(\mathbf{x}) \otimes \mathbf{f}_0(\mathbf{x}), \tag{30}$$

where $\mathbf{f}_0 = (1, 0)^T$ and the parameter space is $\mathcal{P} = 12.9 \cdot [0.06, 0.2] \times 12.9 \cdot [0.03, 0.1]$ cm^2/ms. The applied current is defined as

$$I_{app}(\mathbf{x}, t) = \frac{C}{2\pi\alpha} \exp\left(-\frac{\|\mathbf{x}\|^2}{2\beta}\right)\mathbf{1}_{[0,t]}(t),$$

where $C = 100$ mA, $\alpha = 1$, $\beta = 1$ cm^2 and $t = 2$ ms. The parameters of the Aliev–Panfilov ionic model are set to $K = 8$, $a = 0.01$, $b = 0.15$, $\varepsilon_0 = 0.002$, $c_1 = 0.2$, and $c_2 = 0.3$, see, e.g., [53]. The equations have been discretized in space through linear ($\mathbb{P}_1$) finite elements by considering $N_h = 64 \times 64 = 4096$ grid points. For the time discretization and the treatment of nonlinear terms, we use a one-step, semi-implicit, first order scheme (see, e.g., [54] for further details) by considering a time step $\Delta t = 0.1$ ms over the interval $(0, T)$, with $T = 400$ ms.

For the training phase, we uniformly sample $N_t = 1000$ time instances in the interval $(0, T)$ and consider $N_{train} = 25$ training-parameters, i.e. $\boldsymbol{\mu}_{train} = 12.9 \cdot (0.06 + i0.035, 0.03 + j0.0175)$ with $i, j = 0, \ldots, 4$. For the testing phase, $N_{test} = 16$ testing-parameter instances have been considered, each of them given by $\boldsymbol{\mu}_{test} = 12.9 \cdot (0.0775 + i0.035, 0.0387 + j0.0175)$ with $i, j = 0, \ldots, 3$. The maximum number of epochs is $N_{epochs} = 10\,000$, the batch size is $N_b = 40$ and, regarding the early-stopping criterion, we stop the training if the loss function does not decrease within 500 epochs. The simulations are performed on a GTX 1070 8 GB GPU. We considered problem (29) first in [9] and we now compare the increased efficiency entailed by the use of POD-DL-ROM compared to a DL-ROM and to a POD-Galerkin ROM.

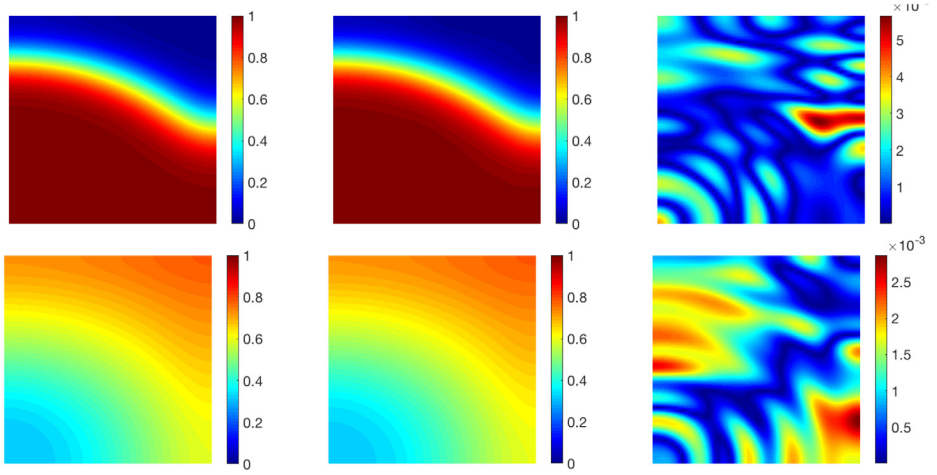

Fig. 6. *Test 2*: FOM (left), POD-DL-ROM (center) solutions and relative error $\boldsymbol{\epsilon}_k$ (right) for the testing-parameter instance $\boldsymbol{\mu}_{test} = 12.9 \cdot (0.1825, 0.0912)$ cm^2/ms at $t = 47.7$ ms and $t = 379.7$ ms, with $n = 3$.

Table 2
Test 2: Error indicator ϵ_{rel} for $\omega_h = 0.5$ and 1.

	$\omega_h = 0.5$	$\omega_h = 1$
ϵ_{rel}	4.03×10^{-3}	7.69×10^{-3}

In Fig. 6 we report the FOM and POD-DL-ROM solutions, the latter obtained by setting $n = 3$ and $N = 64$, along with the relative error (27), for the testing-parameter instance $\boldsymbol{\mu}_{test} = 12.9 \cdot (0.1825, 0.0912)$ cm^2/ms at $t = 47.7$ ms (top) and $t = 379.7$ ms (bottom). The performance of the POD-DL-ROM is measured over the testing set by the error indicator (25), equal to $\epsilon_{rel} = 4.03 \times 10^{-3}$, and the maximum error indicator (26), given by $\epsilon_{max} = 1.02 \times 10^{-2}$. Also in this case, the neural network maintains its accuracy properties over the entire testing set.

We point out that setting ω_h in (18) equal to 0.5, that is, performing a second level of dimensionality reduction by matching the dimension of the minimal components of the problem, leads to higher accuracies with respect to the case $\omega_h = 1$. In this respect, in Table 2, we show the values of the error indicator ϵ_{rel}, over the testing set, by solving the optimization problem (12) with and without considering the term $\mathcal{L}_{int}(t^k, \boldsymbol{\mu}_i; \boldsymbol{\theta})$ in (18). In particular, setting $\omega_h = 0.5$ allows to halve the error.

The comparison among the DL-ROM and POD-DL-ROM neural networks number of parameters, the GPU training and validation time for one epoch, the total number of epochs, the total GPU training and validation (total time) and testing GPU computational times, and the speed-up obtained, at testing time, with respect to the solution of the FOM[5] are reported in Table 3. We also report the total CPU offline and online computational times[6] required by the POD-Galerkin ROM with $N_c = 6$ clusters, which corresponds to the choice providing the most efficient results (see, e.g., Test 4 of [9]), by keeping for all the models the same degree of accuracy $\epsilon_{rel} = 4.03 \times 10^{-3}$ and running the code on the hardware it is optimized for.

The use of a POD-DL-ROM not only entails even faster testing computational times with respect to a DL-ROM, due to the remarkable reduction of the number of parameters of the neural network, but also reduces the training time of a factor 37.5 (resp. 5) compared to a POD-Galerkin ROM (resp. a DL-ROM). The POD-DL-ROM thus results to be the most efficient ROM both at training and testing stages.

We now investigate the use of pretraining, introduced in Section 3.3, in two different scenarios:

- when increasing the FOM dimension N_h;

[5] The FOM simulation is carried out on a MacBook Pro Intel Core i7 6-core with 16 GB RAM.

[6] Here we employ a full 64 GB node (20 Intel® Xeon® E5-2640 v4 2.4 GHz cores) of the HPC cluster available at MOX, Politecnico di Milano.

Table 3
Test 2: DL-ROM, POD-DL-ROM and POD-Galerkin ROM computational times.

	#params	Train - val [s/epoch]	#epochs	Total time	Test [s]	Speed-up
DL-ROM (GPU)	2 342 595	7.5–0.8	6981	15 h	0.08	3.03×10^3
POD-DL-ROM (GPU)	269 057	1.5–0.15	866	24 m	0.015	1.62×10^4
POD-Galerkin ROM ($N_c = 6$)	–	–	–	115 m	8	3.04×10^1

Table 4
Test 2: GPU computational times of pretrained and from scratch POD-DL-ROM for $N_h = 16\,384, 65\,536$.

	#params	#epochs	Total time	Test [s]
POD-DL-ROM ($N_h = 16\,384$)	269 057	1378	38 m	0.06
POD-DL-ROM PRETRAINED ($N_h = 16\,384$)	269 057	165	5 m	0.06
POD-DL-ROM ($N_h = 65\,536$)	269 057	1540	42 m	3
POD-DL-ROM PRETRAINED ($N_h = 65\,536$)	269 057	461	12 m	3

Table 5
Test 2: GPU computational times of pretrained and from scratch POD-DL-ROM for $\mathcal{P} = 12.9 \cdot [0.02, 0.2] \times 12.9 \cdot [0.01, 0.1]$ cm^2/ms.

	#params	#epochs	Total time	Test [s]
POD-DL-ROM	269 057	1486	41 m	0.015
POD-DL-ROM PRETRAINED	269 057	588	16 m	0.015

- when increasing the dimension of the parameter space $\mathcal{P}$;

where the number of epochs run by the optimization algorithm are selected as in the case in which pretraining is not employed. In particular, we set a maximum number of epochs N_{epochs} and we employ an early-stopping technique, that is we stop the training stage if the loss does not decrease over a certain number of epochs. Both N_{epochs} and the number of epochs on which to check the early-stopping criterion before stopping the training are set equal to the values of the hyperparameters we employed for the test run without pretraining. First, we use the optimal parameters, weights and biases, of the POD-DL-ROM neural network found in the case of a FOM dimension $N_h = 4096$, to initialize the POD-DL-ROM neural network parameters associated to two larger FOM dimensions, $N_h = 128 \times 128 = 16\,384$ and $N_h = 256 \times 256 = 65\,536$, by fixing $N = 64$ as rPOD dimension, for a prescribed degree of accuracy $\epsilon_{rel} = 4.03 \times 10^{-3}$. The use of pretraining is possible in this framework because the POD-DL-ROM neural network does not depend on N_h, but only on N. Pretraining then allows to reduce the GPU training computational times of a factor $7 - 3$, as shown in Table 4, where we compare the training and validation (total) time, in presence of pretraining, with the ones of the network trained from scratch. We point out that, by increasing the FOM dimension, the total computational time with pretraining increases. In general, we expect that, by performing multiple, intermediate levels of pretraining, speed-ups will be obtained also for very large FOM dimensions. Moreover, we remark that as long as the physics and the parameter dependence of the solution of the problem become more complex (so the training computational time becomes higher), the gain in terms of acceleration, given by the use of pretraining, increases. Irrespectively of pretraining, testing computational times increase with respect to the case $N_h = 4096$ due to the dependence on N_h of the matrix–vector product $\mathbf{V}_N \mathbf{u}_N$ required to recover the final POD-DL-ROM approximation, since $\mathbf{V}_N \in \mathbb{R}^{N_h \times N}$ has a higher number of rows.

Then, we report the results referred to a larger parameter space, in the case $N_h = 4096$ and $N = 64$. In particular, we use the optimal weights associated to $\mathcal{P} = 12.9 \cdot [0.06, 0.2] \times 12.9 \cdot [0.03, 0.1]$ cm^2/ms as initial guess of the POD-DL-ROM neural network parameters in the case $\mathcal{P} = 12.9 \cdot [0.02, 0.2] \times 12.9 \cdot [0.01, 0.1]$ cm^2/ms. We show the GPU training computational times in Table 5 for a prescribed level of accuracy, i.e. $\epsilon_{rel} = 4.03 \times 10^{-3}$, and a fixed number of training-parameter instances $N_{train} = 25$, with pretraining and without, respectively. Once again, the use of pretraining allows us to speed up the construction of a POD-DL-ROM remarkably.

4.3. Test 3: nonlinear elastodynamics for hyperelastic compressible materials

We now consider the solution of an elastodynamics problem, consisting of the following initial/boundary-value problem [55] for nonlinear elasticity equations, in a three-dimensional beam $\Omega = (0, 1) \times (0, 5) \times (0, 1)$ cm:

$$\begin{cases} \rho \dfrac{\partial^2 \mathbf{d}}{\partial t^2} - \text{div}\,(\mathbf{P}(\mathbf{d})) = \mathbf{f} & (\mathbf{x}, t) \in \Omega \times (0, T), \\ \mathbf{d} = \mathbf{0} & (\mathbf{x}, t) \in \Gamma_D \times (0, T), \\ \mathbf{P}(\mathbf{d})\mathbf{n} = \mathbf{0} & (\mathbf{x}, t) \in \Gamma_N \times (0, T), \\ \mathbf{d}(0) = \mathbf{0} & \mathbf{x} \in \Omega, \ t = 0 \\ \dfrac{\partial \mathbf{d}}{\partial t}(0) = \mathbf{0} & \mathbf{x} \in \Omega, \ t = 0. \end{cases} \tag{31}$$

Here we consider $\rho = 1$ kg/cm^3, $\mathbf{f} = (-0.01, 0, -0.02)$ kg/(cm s^2), $\Gamma_D = \{(x, z) \in (0, 1)^2, y = 0\}$ and $\Gamma_N = \partial\Omega \backslash \Gamma_D$. We consider a St. Venant-Kirchhoff constitutive law involving a hyperelastic nonlinear model to describe the behavior of compressible materials [56], characterized by the following strain energy function

$$\psi(\mathbf{F}) = \nu \mathbf{E} : \mathbf{E} + \frac{\lambda}{2}(\text{tr}(\mathbf{E}))^2.$$

Here $\mathbf{E} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}) = \frac{1}{2}(\mathbf{C} - \mathbf{I})$ is the Green–Lagrange strain tensor, $\mathbf{F} = \mathbf{I} + \nabla \mathbf{d}$ is the deformation tensor defined in terms of the displacement $\mathbf{d} = (d_1, d_2, d_3)$, ν and λ are the Lamé coefficients, and

$$\mathbf{P}(\mathbf{F}) = \mathbf{F}\,(2\nu\mathbf{E} + \lambda \text{tr}(\mathbf{E})\mathbf{I})$$

is the first Piola–Kirchhoff stress tensor. Here $n_\mu = 2$ parameters are considered, given by the Young modulus μ_1 and the Poisson ratio μ_2, belonging to the parameter space $\mathcal{P} = [1, 3]$ Pa $\times$ $[0.25, 0.42]$; they affect the expression of the Lamé coefficients as follows

$$\nu = \frac{\mu_1}{2(1 + \mu_1)} \quad \text{and} \quad \lambda = \frac{\mu_1 \mu_2}{(1 + \mu_2)(1 - 2\mu_2)}.$$

Eqs. (31) are discretized in space by means of quadratic ($\mathbb{P}_2$) finite elements, yielding a dynamical system of dimension $N_h = 5674 \times 3 = 17\,022$. For time integration, we use the generalized-α method [13] over the interval $(0, T)$, with $T = 15$ s and a time-step $\Delta t = 0.2$ s.

We consider $N_t = 75$ time instances over $(0, T)$, $N_{train} = 10 \times 5 = 50$ training-parameter instances $\boldsymbol{\mu}_{train} = (1 + i2/9, 0.25 + j0.0425)$, for $i = 0, \ldots, 9$ and $j = 0, \ldots, 4$, and $N_{test} = 9 \times 4 = 36$ testing-parameter instances $\boldsymbol{\mu}_{test} = (1.111 + i2/9, 0.2712 + j0.0425)$, for $i = 0, \ldots, 9$ and $j = 0, \ldots, 4$. We set the rPOD dimension to $N = 64$ for each of the three components of the displacement and the dimension of the nonlinear trial manifold $\tilde{S}_n$ to $n = 3$, for a total number of degrees of freedom of the POD-DL-ROM solution equal to 3. The maximum number of epochs is $N_{epochs} = 10\,000$, the batch size is $N_b = 20$ and, regarding the early-stopping criterion, we stop the training if the loss function does not decrease along 500 epochs. The training and testing phases are performed on a GTX 1070 8 GB GPU.

In Fig. 7 we show the FOM solution and the POD-DL-ROM one, with $n = 3$, along with the relative error (27), for the testing-parameter instance $\boldsymbol{\mu}_{test} = (2.88$ Pa, $0.3987)$ at $T = 15$ s. The maximum relative error, which is associated to the portion of the domain undergoing the maximum displacement, is about 10^{-3}. Indeed, the error indicators evaluated over the testing set are $\epsilon_{rel} = 1.63 \times 10^{-3}$ and $\epsilon_{max} = 4.46 \times 10^{-3}$. For the sake of completeness, we also computed the error indicator (25) over the training set, which is equal to $\epsilon_{rel} = 1.35 \times 10^{-3}$, in order to highlight the generalization capabilities of the POD-DL-ROM technique. In Fig. 8 we report the FOM and the POD-DL-ROM solutions, obtained by choosing $n = 3$, and the three components of the displacement vector $\mathbf{d} = (d_1, d_2, d_3)$ over the longitudinal axis, i.e. the line which connects the two points $P_1 = (0.5, 0, 0.5)$ cm and $P_2 = (0.5, 5, 0.5)$ cm, for the testing-parameter instance $\boldsymbol{\mu}_{test} = (2.88$ Pa, $0.3987)$ at $T = 15$ s. We remark that all the three components are accurately captured by the POD-DL-ROM.

Here we also want to investigate how the use of pretraining involving different fidelity models impacts on the POD-DL-ROM technique, starting from the previous low-fidelity model. In particular, we consider the nearly-incompressible Neo-Hookean constitutive law [56], an hyperelastic nonlinear model whose strain energy function

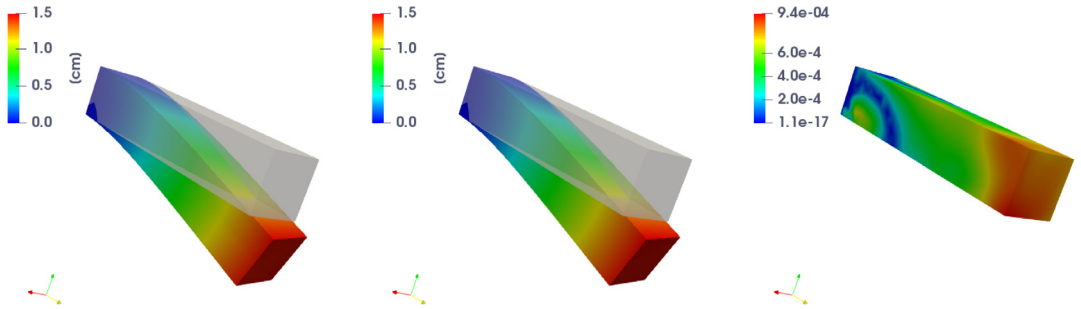

Fig. 7. *Test 3*: FOM (left), POD-DL-ROM (center) solutions and relative error ϵ_k (right), for the testing-parameter instance $\boldsymbol{\mu}_{test} = (2.88\ \text{Pa}, 0.3987)$ at $T = 15$ s, with $n = 3$.

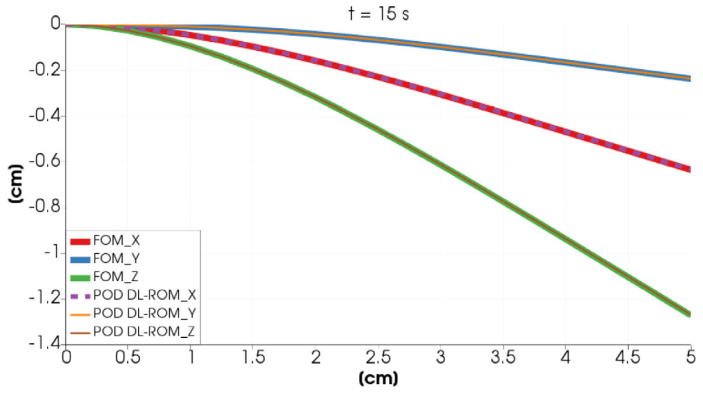

Fig. 8. *Test 3*: FOM and POD-DL-ROM solutions components over the longitudinal axis, for the testing-parameter instance $\boldsymbol{\mu}_{test} = (2.88\ \text{Pa}, 0.3987)$ at $T = 15$ s, with $n = 3$.

is specified in terms of an isochoric–volumetric splitting

$$\psi(\mathbf{F}) = \underbrace{\frac{G}{2}\left(\bar{I}_1 - 3\right)}_{\psi_{\text{iso}}} + \underbrace{\frac{K}{4}\left((J-1)^2 + (\ln J)^2\right)}_{\psi_{\text{vol}}},$$

where $\bar{I}_1 = J^{-2/3} I_1 = J^{-2/3}\text{tr}(\mathbf{C})$, $J = \det(\mathbf{F})$, G is the shear modulus and K the bulk modulus. The coefficients G and K depend on the Young modulus and the Poisson coefficient and are defined as follows

$$G = \frac{\mu_1}{2(1+\lambda)} \quad \text{and} \quad K = \frac{2}{3}G + \lambda.$$

For this model, we consider $n_\mu = 2$ parameters, belonging to the parameter space $\mathcal{P} = [0.1, 1]\ \text{Pa} \times [0.3, 0.45]$, and an external force $\mathbf{f} = (-0.001t, 0, -0.002t)$; moreover, the final time is $T = 22.5$ s, that is, we enlarge the time interval, and the time-step is set equal to 0.25 s. We consider $N_t = 90$ time instances over $(0, T)$, $N_{train} = 50$ training-parameter instances and $N_{test} = 36$ testing-parameter instances uniformly distributed over the parameter space. We use the optimal weights and biases found on the first low-fidelity model, as initial guess for the parameters of the POD-DL-ROM on this second configuration which thus features *(i)* a more involved constitutive law, *(ii)* a different parameter space which reflects in larger displacements, and *(iii)* a longer time interval where to compute the dynamics.

In Fig. 9 we show the FOM and DL-ROM solutions, with $n = 3$, together with the relative error (27), for the testing-parameter instances $\boldsymbol{\mu}_{test} = (0.25\ \text{Pa}, 0.32)$ and $\boldsymbol{\mu}_{test} = (0.95\ \text{Pa}, 0.43)$ at $T = 22.5$ s. In Fig. 10 we compare the FOM and POD-DL-ROM, displaying the three components of the displacement vector over the longitudinal axis, for the testing-parameter instance $\boldsymbol{\mu}_{test} = (0.25\ \text{Pa}, 0.32)$ at $T = 22.5$ s.

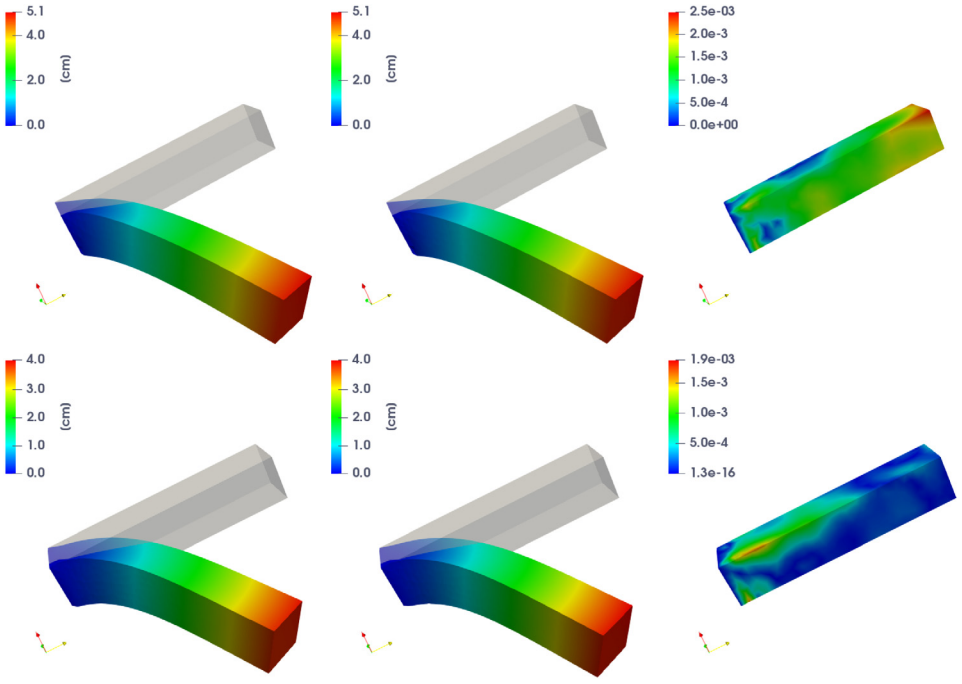

Fig. 9. *Test 3*: FOM (left), POD-DL-ROM (center) solutions and relative error ϵ_k (right), for the testing-parameter instances $\boldsymbol{\mu}_{test} = (0.25\ \text{Pa}, 0.32)$ (top) and $\boldsymbol{\mu}_{test} = (0.95\ \text{Pa}, 0.43)$ (bottom) at $T = 22.5$ s, with $n = 3$.

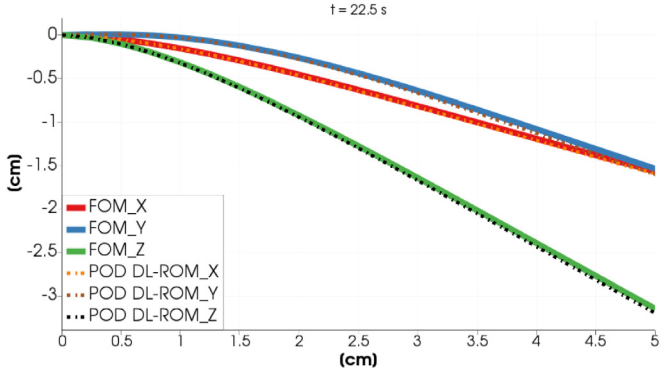

Fig. 10. *Test 3*: FOM and POD-DL-ROM solutions components over the longitudinal axis, for the testing-parameter instance $\boldsymbol{\mu}_{test} = (0.25\ \text{Pa}, 0.32)$ at $T = 22.5$ s, with $n = 3$.

In Table 6 we finally compare the GPU total and testing computational times of the POD-DL-ROM neural network with and without the use of pretraining. In particular, the use of pretraining allows to strongly reduce the total training and validation time, which already entailed a speed-up equal to 23 with respect to the DL-ROM total time, for the same level of accuracy. The testing computational time, which refers to the time needed by the POD-DL-ROM to compute $N_t = 90$ time instances for a testing-parameter instance, is equal to 0.006 s, and is remarkably lower than the final time $T = 22.5$ s, that is our technique is able to return even faster than real-time solutions. In particular, the testing time reflects in a speed-up, if we consider the time required by the solution of the FOM,[7] equal to 4.12×10^4.

[7] The simulation is performed on 20 cores of 1.7 TB node (192 Intel® Xeon Platinum® 8160 2.1 GHz cores) of the HPC cluster available at MOX, Politecnico di Milano.

Table 6
Test 3: GPU computational times of pretrained and from scratch POD-DL-ROM.

	#params	#epochs	Total time	Test [s]
POD-DL-ROM	270 259	6490	63 m	0.006
POD-DL-ROM PRETRAINED	270 259	1519	15 m	0.006

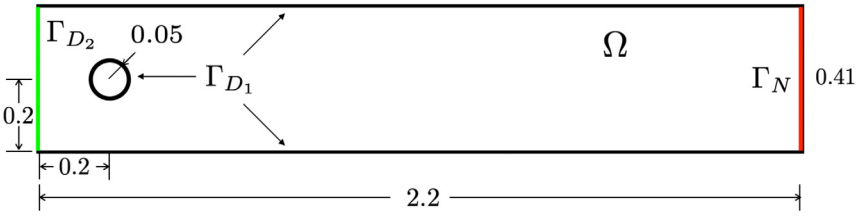

Fig. 11. *Test 4*: geometrical configuration, domain and boundaries.

4.4. Test 4: unsteady Navier–Stokes equations

We finally focus on the unsteady Navier–Stokes equations [1] for incompressible flows in primitive variables (fluid velocity $\mathbf{u}$ and pressure p), considering the flow around a cylinder test case, a well-known benchmark for the evaluation of numerical algorithms for incompressible Navier–Stokes equations in the laminar case:

$$\begin{cases} \rho \dfrac{\partial \mathbf{u}}{\partial t} + \rho \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \cdot \boldsymbol{\sigma}(\mathbf{u}, p) = \mathbf{0} & (\mathbf{x}, t) \in \Omega \times (0, T), \\ \nabla \cdot \mathbf{u} = 0 & (\mathbf{x}, t) \in \Omega \times (0, T), \\ \mathbf{u} = \mathbf{0} & (\mathbf{x}, t) \in \Gamma_{D_1} \times (0, T), \\ \mathbf{u} = \mathbf{h}(\mu) & (\mathbf{x}, t) \in \Gamma_{D_2} \times (0, T), \\ \boldsymbol{\sigma}(\mathbf{u}, p)\mathbf{n} = \mathbf{0} & (\mathbf{x}, t) \in \Gamma_N \times (0, T), \\ \mathbf{u}(0) = \mathbf{0} & \mathbf{x} \in \Omega, \ t = 0. \end{cases} \tag{32}$$

The domain consists in a two-dimensional pipe with a circular obstacle, i.e. $\Omega = (0, 2.2) \times (0, 0.41) \backslash \bar{B}_r(0.2, 0.2)$ with radius $r = 0.05$ (see Fig. 11 for a sketch of the geometry); the boundary is given by $\partial\Omega = \Gamma_{D_1} \cup \Gamma_{D_2} \cup \Gamma_N$, where $\Gamma_{D_1} = \{x_1 \in [0, 2.2], x_2 = 0\} \cup \{x_1 \in [0, 2.2], x_2 = 0.41\} \cup \partial B_{0.05}((0.2, 0.2))$, being $B_r(\mathbf{x}_c)$ the ball of radius $r > 0$ centered at $\mathbf{x}_c$, $\Gamma_{D_2} = \{x_1 = 0, x_2 \in [0, 0.41]\}$, and $\Gamma_N = \{x_1 = 2.2, x_2 \in [0, 0.41]\}$, while $\mathbf{n}$ denotes the (outward directed) normal unit vector to $\partial\Omega$. We denote by ρ the fluid density, and by $\boldsymbol{\sigma}$ the stress tensor,

$$\boldsymbol{\sigma}(\mathbf{u}, p) = -p\mathbf{I} + 2\nu\boldsymbol{\epsilon}(\mathbf{u}).$$

Here ν denotes the dynamic viscosity of the fluid, while $\boldsymbol{\epsilon}(\mathbf{u})$ is the strain tensor,

$$\boldsymbol{\epsilon}(\mathbf{u}) = \frac{1}{2}\left(\nabla \mathbf{u} + \nabla \mathbf{u}^T\right).$$

The density of the fluid is $\rho = 1$, no-slip boundary conditions are applied on Γ_1, a parabolic inflow profile

$$\mathbf{h}(\mathbf{x}, t; \mu) = \left(\frac{4U(t, \mu)x_2(0.41 - x_2)}{0.41^2}, 0 \right), \qquad \text{where} \quad U(t; \mu) = \mu \sin(\pi t/8), \tag{33}$$

is prescribed at the inlet Γ_{D_2}, while zero-stress Neumann conditions are imposed at the outlet Γ_N. We consider as parameter ($n_\mu = 1$) $\mu \in \mathcal{P} = [1, 2]$, which reflects on the Reynolds number varying in the range $[66, 133]$. Eqs. (32) have been discretized in space by means of linear–quadratic ($\mathbb{P}_2 - \mathbb{P}_1$), inf–sup stable, finite elements, and in time through a BDF of order 2 with semi-implicit treatment of the convective term (see, e.g., [57]) over the time interval $T = 8$, with a time-step $\Delta t = 2 \times 10^{-3}$.

We uniformly sample $N_t = 400$ time instances and consider $N_{train} = 11$ and $N_{test} = 10$ training- and testing-parameter instances uniformly distributed over $\mathcal{P}$. We are interested in reconstructing the velocity field, for which the FOM dimension is equal to $N_h = 32\,446 \times 2 = 64\,892$, selecting $N = 256$ as dimension of the rPOD basis

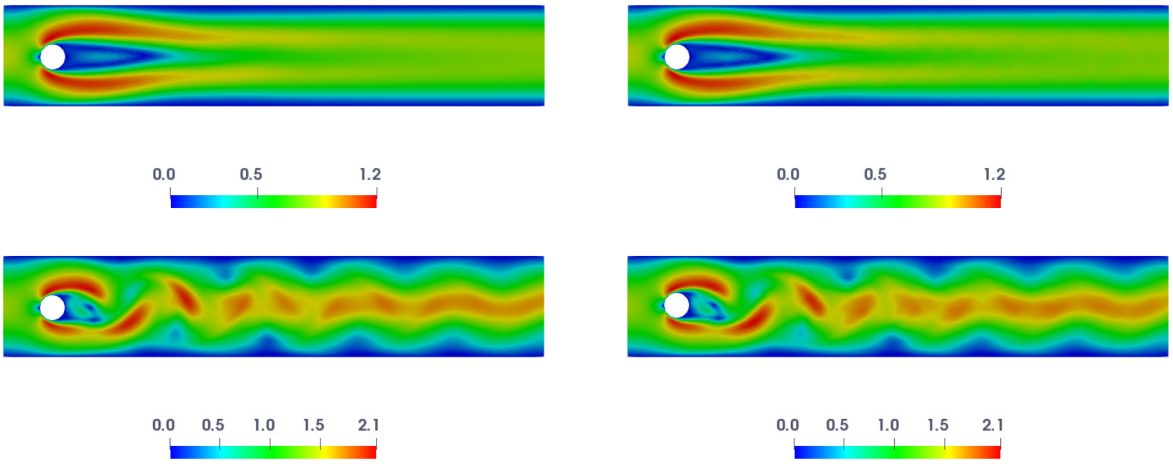

Fig. 12. *Test 4*: FOM (left) and POD-DL-ROM (right) solutions for the testing-parameter instances $\mu_{test} = 1.05$ (top) and $\mu_{test} = 1.75$ (bottom) at $t = 5.64$, with $n = 2$.

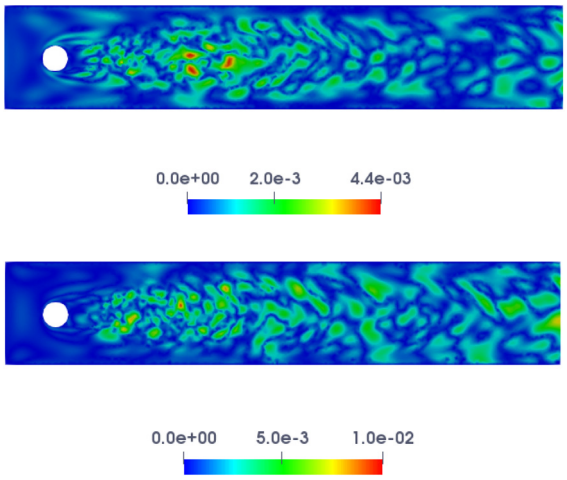

Fig. 13. *Test 4*: Relative error for the testing-parameter instances $\mu_{test} = 1.05$ (top) and $\mu_{test} = 1.75$ (bottom) at $t = 5.64$.

for each of the two velocity components. We choose $n = 2$ as dimension of the nonlinear trial manifold $\tilde{S}_n$. We highlight the possibility, by using POD-DL-ROM, to reconstruct the field of interest, i.e the velocity $\mathbf{u}$, without the need of taking into account the approximation of the pressure p.

In Fig. 12 we compare the FOM and POD-DL-ROM solutions, the latter for $n = 2$, together with the relative error $\boldsymbol{\epsilon}_k$ in Fig. 13, for two testing-parameter instances $\mu_{test} = 1.05$ (Re $= 70$) and $\mu_{test} = 1.75$ (Re $= 117$) at $t = 5.64$. We highlight the ability of the POD-DL-ROM approximation to accurately capture the variability of the solution over the parameter space $\mathcal{P}$: indeed, in the case Re $= 70$ (Fig. 12, top) we do not assist to any vortex shedding; this latter is instead present in the case Re $= 117$ (Fig. 12, bottom).

The computational training and testing time of the POD-DL-ROM neural network on a Tesla V100 32 GB GPU are equal to 50 min and 0.1 s, respectively, in contrast to the DL-ROM computational times, on the same GPU, equal to 58 h and 1.2 s, for the same level of accuracy.

To show the ability of the POD-DL-ROM to provide accurate evaluations of output quantities of interest, we consider the computation of the flow rate on the outflow boundary Γ_N, defined as

$$f_r(t;\mu) = \int_{\Gamma_N} \mathbf{u}(\mathbf{x}, t;\mu) \cdot \mathbf{n}\, d\sigma.$$

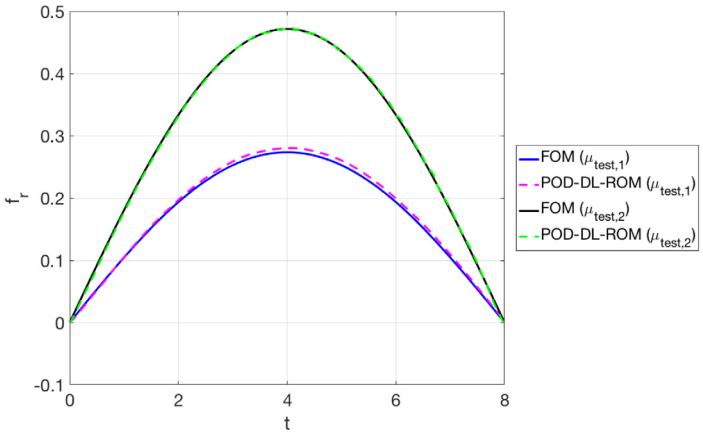

Fig. 14. *Test 4*: FOM and POD-DL-ROM flow rates for the testing-parameter instances $\mu_{test,1} = 1.05$ and $\mu_{test,2} = 1.75$.

Table 7
Test 4: POD-DL-ROM speed-ups.

	$N_h = 64\,892$	$N_h = 257\,528$
Speed-up	2.15×10^5	6.59×10^5

In this respect, in Fig. 14 we show the FOM and POD-DL-ROM flow rates over time, for the two testing-parameter instances $\mu_{test,1} = 1.05$ and $\mu_{test,2} = 1.75$. The POD-DL-ROM technique is able to capture the shape of f_r, related to the prescribed μ-dependent input profile in (33), in both cases, introducing a maximum relative error equal to 1.65%.

We then investigate the use of pretraining when aiming at reducing the complexity of the solution of the same problem, whose high-fidelity discretization is set on a finer computational mesh. In this case, we increase the FOM dimension to $N_h = 128\,764 \times 2 = 257\,528$, and train the networks starting from the optimal parameters found on the low-fidelity model related to $N_h = 32\,446 \times 2 = 64\,892$. In particular, 434 epochs, which results in a training computational time of 10 min, are required to achieve in the case $N_h = 128\,764 \times 2 = 257\,528$ the same accuracy $\epsilon_{rel} = 1.49 \times 10^{-2}$ (or, equivalently, $\epsilon_{max} = 2.53 \times 10^{-2}$) obtained for the previous case with $N_h = 32\,446 \times 2 = 64\,892$. Finally, we show in Table 7 the speed-ups introduced, at testing time, by the use of the POD-DL-ROM technique with respect the solution of the FOM[8] when aiming at evaluating the fluid velocity over the interval $(0, T)$, for both cases $N_h = 64\,892$ and $257\,528$.

We remark that ensuring ROM stability in the classical POD-Galerkin framework usually requires additional computational efforts, such as a suitable enrichment of the velocity reduced basis, and a consequent increase of the size of the ROM; see, e.g., [6,7,58].

5. Conclusions

In this work we proposed a strategy to enhance DL-ROMs in order to make the offline training stage dramatically faster. Indeed, a key aspect in the setting of DL-ROMs concerns computational efficiency during the offline (or training) stage, which is also related with the curse of dimensionality. This strategy, which we refer to as POD-DL-ROM, overcomes the main computational bottleneck of the DL-ROM technique, namely the (strong) limitation related to the FOM dimension N_h. In particular, it exploits *(i)* dimensionality reduction of FOM snapshots by means of randomized POD (or randomized SVD) and *(ii)* a suitable multi-fidelity pretraining stage exploiting snapshots computed through lower-fidelity models to initialize the parameters of neural networks in a sequential procedure. Moreover, the POD-DL-ROM approximations retain all the features of DL-ROM solutions, enabling extremely efficient testing computational times.

[8] The simulations are performed on 20 cores of 1.7 TB node (192 Intel® Xeon Platinum® 8160 2.1 GHz cores) of the HPC cluster available at MOX, Politecnico di Milano.

We assessed computational performance, numerical accuracy and robustness of the POD-DL-ROM technique on several time-dependent parametrized PDEs, namely *(i)* a linear advection–diffusion–reaction problem, *(ii)* a nonlinear diffusion–reaction problem arising from cardiac electrophysiology, *(iii)* nonlinear elastodynamics for hyperelastic compressible materials, and *(iv)* fluid dynamics. In all these cases, POD-DL-ROMs are able to match the intrinsic dimension of the problems investigated, to overcome the main computational bottleneck shown by conventional projection-based methods, and to make the training phase of ROMs extremely fast. Through the numerical test cases assessed in Section 4, POD-DL-ROMs have shown to yield extremely efficient numerical approximations to (scalar and vector) nonlinear time-dependent parametrized PDEs, thus providing a *turn-key* strategy to build ROMs only relying on a set of FOM snapshots, and ultimately leading to the possibility to simulate in less than real-time, during the *online* testing stage, physical phenomena occurring on a time scale of seconds.

Declaration of competing interest

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

Acknowledgments

We gratefully acknowledge Prof. A. Quarteroni and Prof. L. Dede' (MOX, Politecnico di Milano) for their insightful remarks. This work has been supported by Fondazione Cariplo, Italy, Grant n. 2019-4608.

References

[1] A. Quarteroni, A. Valli, Numerical Approximation of Partial Differential Equations, Vol. 23, Springer, 1994.
[2] A. Quarteroni, A. Manzoni, F. Negri, Reduced Basis Methods for Partial Differential Equations: An Introduction, Vol. 92, Springer, 2016.
[3] P. Benner, A. Cohen, M. Ohlberger, K. Willcox, Model Reduction and Approximation: Theory and Algorithms, SIAM, 2017.
[4] P. Benner, S. Gugercin, K. Willcox, A survey of projection-based model reduction methods for parametric dynamical systems, SIAM Rev. 57 (4) (2015) 483–531.
[5] C. Farhat, S. Grimberg, A. Manzoni, A. Quarteroni, Computational bottlenecks for PROMs: Pre-computation and hyperreduction, in: P. Benner, S. Grivet-Talocia, A. Quarteroni, G. Rozza, W. Schilders, L. Silveira (Eds.), Model Order Reduction. Volume 2: Snapshot-Based Methods and Algorithms, De Gruyter, Berlin, 2020, pp. 181–244.
[6] N. Dal Santo, S. Deparis, A. Manzoni, A. Quarteroni, An algebraic least squares reduced basis method for the solution of nonaffinely parametrized Stokes equations, Comput. Methods Appl. Mech. Engrg. 344 (2019) 186–208.
[7] F. Ballarin, A. Manzoni, A. Quarteroni, G. Rozza, Supremizer stabilization of POD-Galerkin approximation of parametrized steady incompressible Navier-Stokes equations, Internat. J. Numer. Methods Engrg. 102 (5) (2015) 1136–1161.
[8] G. Rozza, D. Huynh, A. Manzoni, Reduced basis approximation and a posteriori error estimation for Stokes flows in parametrized geometries: Roles of the inf-sup stability constants, Numer. Math. 125 (1) (2013) 115–152.
[9] S. Fresca, L. Dedè, A. Manzoni, A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs, J. Sci. Comput. 87 (2021) 61.
[10] S. Fresca, A. Manzoni, L. Dedè, A. Quarteroni, Deep learning-based reduced order models in cardiac electrophysiology, PLOS ONE 15 (10) (2020) 1–32.
[11] A. Szlam, Y. Kluger, M. Tygert, An implementation of a randomized algorithm for principal component analysis, 2014, arxiv preprint arXiv:1412.3510v1.
[12] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[13] J. Chung, G.M. Hulbert, A time integration algorithm for structural dynamics with improved numerical dissipation: The generalized-α method, J. Appl. Mech. 60 (2) (1993) 371–375.
[14] K. Lee, K.T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, J. Comput. Phys. 404 (2020) 108973.
[15] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient based learning applied to document recognition, Proc. IEEE (1998) 533–536.
[16] G.E. Hinton, R.S. Zemel, Autoencoders, minimum description length, and Helmholtz free energy, in: Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS'1993), 1994.
[17] G. Kutyniok, P. Petersen, M. Raslan, R. Schneider, A theoretical analysis of deep neural networks and parametric PDEs, 2019, arXiv preprint arXiv:1904.00377.
[18] J.A.A. Opschoor, P.C. Petersen, C. Schwab, Deep ReLU networks and high-order finite element methods, Anal. Appl. 18 (05) (2020) 715–770.
[19] F. Laakmann, P. Petersen, Efficient approximation of solutions of parametric linear transport equations by ReLU DNNs, 2020, arXiv preprint arXiv:2001.11441.
[20] M. Raissi, G.E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.

[21] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations, 2017, arXiv preprint arXiv:1711.10561.
[22] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (Part II): Data-driven discovery of nonlinear partial differential equations, 2017, arXiv preprint arXiv:1711.10566.
[23] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[24] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, 2017, arXiv preprint arXiv:1707.02568.
[25] Y. Yang, P. Perdikaris, Physics-informed deep generative models, 2018, arXiv preprint arXiv:11812.0351.
[26] M. Guo, J.S. Hesthaven, Reduced order modeling for nonlinear structural analysis using Gaussian process regression, Comput. Methods Appl. Mech. Engrg. 341 (2018) 807–826.
[27] M. Guo, J.S. Hesthaven, Data-driven reduced order modeling for time-dependent problems, Comput. Methods Appl. Mech. Engrg. 345 (2019) 75–99.
[28] J. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, J. Comput. Phys. 363 (2018) 55–78.
[29] M. Kast, M. Guo, J.S. Hesthaven, A non-intrusive multifidelity method for the reduced order modeling of nonlinear problems, Comput. Methods Appl. Mech. Engrg. 364 (2020) 112947.
[30] Q. Wang, J.S. Hesthaven, D. Ray, Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem, J. Comput. Phys. 384 (2019) 289–307.
[31] J.N. Kani, A.H. Elsheikh, DR-RNN: A deep residual recurrent neural network for model reduction, 2017, arXiv preprint arXiv:1709.00939.
[32] A. Mohan, D.V. Gaitonde, A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks, 2018, arXiv preprint arXiv:1804.0926.
[33] Z. Wan, P. Vlachas, P. Koumoutsakos, T. Sapsis, Data-assisted reduced-order modeling of extreme events in complex dynamical systems, PLOS ONE 13 (2018).
[34] R. Pulch, M. Youssef, Machine learning for trajectories of parametric nonlinear dynamical systems, J. Mach. Learn. Model. Comput. 1 (1) (2020) 75–95.
[35] A. Bērzinš, J. Helmig, F. Key, S. Elgeti, Standardized non-intrusive reduced order modeling using different regression models with application to complex flow problems, 2020, arXiv preprint arXiv:2006.13706v1.
[36] W. Chen, Q. Wang, J.S. Hesthaven, C. Zhang, Physics-informed machine learning for reduced-order modeling of nonlinear problems, 2020, Preprint.
[37] J.N. Kani, A.H. Elsheikh, Reduced-order modeling of subsurface multi-phase flow models using deep residual recurrent neural networks, Transp. Porous Media 126 (3) (2018) 713–741.
[38] O. San, R. Maulik, Neural network closures for nonlinear model order reduction, Adv. Comput. Math. 44 (6) (2018) 1717–1750.
[39] Q. Wang, N. Ripamonti, J.S. Hesthaven, Recurrent neural network closure of parametric POD-Galerkin reduced-order models based on the mori-zwanzig formalism, J. Comput. Phys. 410 (2020) 109402.
[40] K. Bhattacharya, B. Hosseini, N.B. Kovachki, A. Stuart, Model reduction and neural networks for parametric PDEs, 2020, arXiv preprint arXiv:2005.03180.
[41] Y. Kim, Y. Choi, D. Wideman, T. Zohdi, A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder, 2020, arXiv preprint arXiv:2009.11990.
[42] F.J. González, M. Balajewicz, Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, 2018, arXiv preprint arXiv:1808.01346.
[43] M. Raissi, Deep hidden physics models: Deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. 19 (2018) 1–24.
[44] P. Drineas, R. Kannan, M.W. Mahoney, Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix, SIAM J. Comput. 36 (2006) 158–183.
[45] N. Halko, P. Martinsson, J.A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2011) 217–288.
[46] M.E. Taylor, P. Stone, Transfer learning for reinforcement learning domains: A survey, J. Mach. Learn. Res. 10 (2009) 1633–1685.
[47] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks? in: Advances in Neural Information Processing Systems 27, Vol. 27, 2014, pp. 3320–3328.
[48] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, 2018, arXiv preprint arXiv:1810.04805.
[49] E. Haghighat, M. Raissi, A. Moure, H. Gomez, R. Juanes, A deep learning framework for solution and discovery in solid mechanics, 2020, arXiv preprint arXiv:2003.02751.
[50] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: A system for large-scale machine learning, 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.
[51] P. Colli Franzone, L.F. Pavarino, S. Scacchi, Mathematical Cardiac Electrophysiology, in: Modeling, Simulation & Applications, vol. 13, Springer, 2014.
[52] R.R. Aliev, A.V. Panfilov, A simple two-variable model of cardiac excitation, Chaos Solitons Fractals 7 (3) (1996) 293–301.
[53] S. Göktepe, J. Wong, E. Kuhl, Atrial and ventricular fibrillation: Computational simulation of spiral waves in cardiac tissue, Arch. Appl. Mech. 80 (2010) 569–580.

[54] S. Pagani, A. Manzoni, A. Quarteroni, Numerical approximation of parametrized problems in cardiac electrophysiology by a local reduced basis method, Comput. Methods Appl. Mech. Engrg. 340 (2018) 530–558.
[55] M.E. Gurtin, An Introduction to Continuum Mechanics, Vol. 158, Academic Press, 1982.
[56] R.W. Ogden, Non-Linear Elastic Deformations, Courier Corporation, 1997.
[57] D. Forti, L. Dedè, Semi-implicit BDF time discretization of the Navier-Stokes equations with VMS-LES modeling in a high performance computing framework, Comput. & Fluids 117 (2015) 168–182.
[58] G. Rozza, K. Veroy, On the stability of the reduced basis method for Stokes equations in parametrized domains, Comput. Methods Appl. Mech. Engrg. 196 (7) (2007) 1244–1260.