Journal of Scientific Computing (2022) 93:57
https://doi.org/10.1007/s10915-022-02001-8

Deep-HyROMnet: A Deep Learning-Based Operator Approximation for Hyper-Reduction of Nonlinear Parametrized PDEs

Ludovica Cicci[1] · Stefania Fresca[1] · Andrea Manzoni[1]

Received: 9 February 2022 / Revised: 24 June 2022 / Accepted: 24 July 2022

Abstract

To speed-up the solution of parametrized differential problems, reduced order models (ROMs) have been developed over the years, including projection-based ROMs such as the reduced-basis (RB) method, deep learning-based ROMs, as well as surrogate models obtained through machine learning techniques. Thanks to its physics-based structure, ensured by the use of a Galerkin projection of the full order model (FOM) onto a linear low-dimensional subspace, the Galerkin-RB method yields approximations that fulfill the differential problem at hand. However, to make the assembling of the ROM independent of the FOM dimension, intrusive and expensive hyper-reduction techniques, such as the discrete empirical interpolation method (DEIM), are usually required, thus making this strategy less feasible for problems characterized by (high-order polynomial or nonpolynomial) nonlinearities. To overcome this bottleneck, we propose a novel strategy for learning nonlinear ROM operators using deep neural networks (DNNs). The resulting hyper-reduced order model enhanced by DNNs, to which we refer to as Deep-HyROMnet, is then a physics-based model, still relying on the RB method approach, however employing a DNN architecture to approximate reduced residual vectors and Jacobian matrices once a Galerkin projection has been performed. Numerical results dealing with fast simulations in nonlinear structural mechanics show that Deep-HyROMnets are orders of magnitude faster than POD-Galerkin-DEIM ROMs, still ensuring the same level of accuracy.

Keywords Nonlinear time-dependent PDEs · Reduced order modeling · Deep learning · Proper orthogonal decomposition · Hyper-reduction

✉ Andrea Manzoni
andrea1.manzoni@polimi.it

Ludovica Cicci
ludovica.cicci@polimi.it

Stefania Fresca
stefania.fresca@polimi.it

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

Published online: 11 October 2022

Mathematics Subject Classification 65M60 · 68T01

1 Introduction

Mathematical models involving partial differential equations (PDEs) depending on a set of parameters are ubiquitous in applied sciences and engineering. These input parameters are defined to characterize, e.g., material properties, loads, boundary/initial conditions, source terms, or geometrical features. High-fidelity simulations based on full-order models (FOMs), like the finite element (FE) method, entail huge computational costs in terms of CPU time and memory, if a large number of degrees of freedom dofs) is required. Complexity is amplified whenever interested to go beyond a single direct simulation, such as in the multi-query contexts of optimization, parameter estimation and uncertainty quantification. To face these problems, several strategies to build reduced order models (ROMs) have been developed over the years, aiming at computing reliable solutions to parametrized PDEs at a greatly reduced cost. A large class of ROMs relies on a projection-based approach, which aims at approximating the unknown state quantities as a linear superimposition of basis functions; these latter then span a subspace which the governing equations are projected onto [8, 9]. Among these, the reduced basis (RB) method [35, 49] is a powerful and widely used technique, characterized by a splitting of the reduction procedure into an expensive, parameter-independent offline phase (however performed once and for all) and an efficient, parameter-dependent online phase. Its efficiency mainly relies on two crucial assumptions:

1. The solution manifold is low-dimensional, so that the FOM solutions can be approximated as a linear combination of few reduced modes with a small error;
2. The online stage is completely independent of the high-fidelity dimension [21].

Assumption 1 concerns the approximability of the solution set and is associated with the slow decay of the Kolmogorov N-width [46]. However, for physical phenomena characterized by a slow N-width decay, such as those featuring coherent structures that propagate over time [25], the manifold spanned by all the possible solutions is not of small dimension, so that ROMs relying on linear (global) subspaces might be inefficient. Alternative strategies to overcome this bottleneck can be, e.g., local RB methods [2, 43, 55], or nonlinear approximation techniques, mainly based on deep learning architectures, see, e.g., [22–24, 37, 39].

Assumption 2 is automatically verified in linear, affinely parametrized problems [49], but cannot be fulfilled when dealing with nonlinear problems, as the online assembling of the reduced operators requires to reconstruct the high-fidelity ones. To overcome this issue, a further level of approximation, or *hyper-reduction*, must be introduced. State-of-the-art methods, such as the empirical interpolation method (EIM) [6], the discrete empirical interpolation method (DEIM) [15], its variant matrix DEIM (MDEIM) [42], the missing point estimation [4] and the Gauss-Newton with approximated tensors (GNAT) [13], aim at recovering an affine expansion of the nonlinear operators by computing only a few entries of the nonlinear terms. EIM, DEIM and GNAT can be seen as *approximate-then-project* techniques, since operator approximation is performed at the level of FOM quantities, prior to the projection stage. On the other hand, *project-then-approximate* strategies have also been introduced, aiming at approximating directly ROM operators, such as the reduced nonlinear residual and its Jacobian. An option in this sense is represented by the so-called Energy Conserving Sampling and Weighting (ECSW) technique [20]; see. e.g., [21] for a detailed review.

Although extensively used in many applications, spanning from fluid flows models to cardiac mechanics [2, 11, 19, 27, 50, 54], these strategies are code-intrusive and, more importantly, might impact on the overall efficiency of the ROM approximation in complex applications. Indeed, when dealing with highly nonlinear problems, expensive hyper-reduction strategies are usually required if aiming at preserving the physical constraints at the ROM level, that is, if ROMs are built consistently with the FOM through a projection-based procedure. For instance, a large number of DEIM basis vectors are required to ensure the convergence of the reduced Newton systems arising from the linearization of the nonlinear hyper-ROM when dealing with highly nonlinear elastodynamics problems [17], even if few basis functions are required to approximate the state solution in a low-dimensional subspace through, e.g., proper orthogonal decomposition (POD). An alternative formulation of DEIM in a finite element framework, known as unassembled DEIM (UDEIM) [53], has been proposed to preserve the sparsity of the problem, while in [44] a localized DEIM selecting smaller local subspace for the approximation of the nonlinear term is presented.

Semi-intrusive strategies, avoiding the ROM construction through a Galerkin projection, have been recently proposed exploiting surrogate models to determine the RB approximation. For instance, neural networks (NNs) or Gaussian process regression can be employed to learn the map between the input parameters and the reduced-basis expansion coefficients in a non-intrusive way [32, 33, 36, 52]. An approximation of the nonlinear term arising in projection-based ROMs is obtained in [26] through deep NNs (DNNs) that exploit the projection of FOM solutions. NNs have also also been recently applied in the context of *operator inference* for (parametrized) differential equations, combining ideas from classical model reduction with data-driven learning. For instance, the design of NNs able to accurately represent linear/nonlinear operators, mapping input functions to output functions, has been proposed recently in [40]; based on the universal approximation theorem of operators [16], a general deep learning framework, called DeepONet, has been introduced to learn continuous operators, such as solution operators of PDEs, using DNNs; see also [56]. In [45] a non-intrusive projection-based ROM for parametrized time-dependent PDEs including low-order polynomial nonlinear terms is considered, inferring an approximation of the reduced operators directly from data of the FOM. Finally, the obtained low-dimensional system is solved—in this case, the learning task consists in the solution to a least squares problem; see also [7]. Projection-based ROMs and machine learning have been fused in [47] aiming at the approximation of linear and quadratic ROM operators, focusing on the solution to a large class of fluid dynamics applications. Similarly, in [5] a non-intrusive technique, exploiting machine learning regression algorithms, is proposed for the approximation of ROM operators related to projection-based methods for the solution of parametrized PDEs. Finally, principal component analysis-based model reduction with a NNs for approximation has been combined in [10], in a purely data-driven fashion, of infinite-dimensional solution maps, such as the solution operator for time-dependent PDEs.

In this paper, we develop a novel semi-intrusive, deep learning-enhanced hyper-reduced order modeling strategy, which hereon we refer to as *shape Deep-HyROMnet*, by leveraging a Galerkin-RB method for solution dimensionality reduction and DNNs to perform hyper-reduction. Since the efficiency of the nonlinear ROM hinges upon the cost-effective approximation of the projections of the (discrete) reduced residual operator and its Jacobian (when an implicit numerical scheme is employed), the key idea is to overcome the computational bottleneck associated with classical, intrusive hyper-reduction techniques, like DEIM, by relying on DNNs to approximate otherwise expensive reduced nonlinear operators at a greatly reduced cost. Unlike data-driven-based methods, for which the predicted output is not guaranteed to satisfy the underlying PDE, our proposed method is physics-based, as it computes the ROM solution by actually solving the reduced nonlinear systems by means of the Newton method, thus exploiting the physics of the problem. A further benefit of the proposed Deep-HyROMnet method lies on the fact that the inputs given to the NNs are low-dimensional arrays, so that the overwhelming training times and costs that may be required by even moderately large FOM dimensions can be avoided. Note that we aim at efficiently approximating the nonlinear operators that result from the composition of the reduced solution operator—that maps the input parameter vector and time to the corresponding ROM solution—and the reduced residual/Jacobian operator, that maps the ROM solution to the reduced residual/Jacobian evaluated on the ROM solution. We apply the proposed methodology to the solution of problems in nonlinear solid mechanics, focusing on parametrized nonlinear elastodynamics and complex (e.g., exponential nonlinear) constitutive relations of the material undergoing large deformations, showing that the Deep-HyROMnet approach outperforms the Galerkin-RB method equipped with DEIM in terms of computational speed-up during the online stage, achieving the same level of accuracy results.

The paper is structured as follows. We recall the formulation of the RB method for nonlinear unsteady parametrized PDEs in Sect. 2, relying on POD for the construction of the reduced subspace and on DEIM as hyper-reduction technique (hence obtaining POD-Galerkin-DEIM ROMs). The proposed Deep-HyROMnet and the DNN architecture employed to perform reduced operator approximation are detailed in Sect. 3. The numerical performances of the resulting method are assessed in Sect. 4 on several benchmark problems related with nonlinear elastodynamics, highlighting some conclusions in Sect. 5.

2 Projection-Based ROMs: The Reduced Basis Method

Our goal is to pursue an efficient solution to nonlinear unsteady PDE problems depending on a set of input parameters, which can be written in abstract form as follows: given an input parameter vector $\boldsymbol{\mu} \in \mathcal{P}$ and $\mathbf{u}(0; \boldsymbol{\mu})$, find $\mathbf{u}(t; \boldsymbol{\mu}) \in V$ such that, $\forall t \in (0, T]$,

$$R(\mathbf{u}(t; \boldsymbol{\mu}), t; \boldsymbol{\mu}) = 0 \quad \text{in } V', \tag{1}$$

where the parameter space $\mathcal{P} \subset \mathbb{R}^P$ is a compact set, R is the residual of a second-order differential equation, and V' is the dual of a suitable functional space $H_0^1(\Omega)^m \subseteq V \subseteq H^1(\Omega)^m$ over the bounded domain $\Omega \subset \mathbb{R}^d$, where V depends on the boundary conditions at hand. In particular, we are interested in vector problems ($m = 3$) set in $d = 3$ dimensions. The role of the parameter vector $\boldsymbol{\mu}$ depends on the particular application at hand; in the case of nonlinear elastodynamics, $\boldsymbol{\mu}$ is for instance related to the coefficients of the constitutive relation, the material properties and the boundary conditions.

After discretising problem (1) in space and time, we end up with a fully-discrete nonlinear system

$$\mathbf{R}(\mathbf{u}_h^n(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}) = \mathbf{0} \quad \text{in } \mathbb{R}^{N_h}, \tag{2}$$

at each time step t^n, $n = 1, \ldots, N_t$, which can be solved by means of the Newton method: given $\boldsymbol{\mu} \in \mathcal{P}$ and an initial guess $\mathbf{u}_h^{n,(0)}(\boldsymbol{\mu})$, for $k \geq 0$, find $\mathbf{u}_h^{n,(k)}(\boldsymbol{\mu}) \in \mathbb{R}^{N_h}$ such that

$$\begin{cases} \mathbf{J}(\mathbf{u}_h^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu})\mathbf{u}_h^{n,(k)}(\boldsymbol{\mu}) = -\mathbf{R}(\mathbf{u}_h^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}) \\ \mathbf{u}_h^{n,(k+1)}(\boldsymbol{\mu}) = \mathbf{u}_h^{n,(k)}(\boldsymbol{\mu}) + \mathbf{u}_h^{n,(k)}(\boldsymbol{\mu}) \end{cases} \tag{3}$$

until suitable stopping criteria are fulfilled. Here, $\mathbf{u}_h^{n,(k)}(\boldsymbol{\mu})$ represents the solution vector for a fixed parameter $\boldsymbol{\mu}$ computed at time step t^n and Newton iteration k, while $\mathbf{R} \in \mathbb{R}^{N_h}$ and

$\mathbf{J} \in \mathbb{R}^{N_h \times N_h}$ denote the residual vector and the corresponding Jacobian matrix, respectively. In particular, $\mathbf{u}_h^{n,(0)}(\boldsymbol{\mu})$ is selected as the solution vector obtained at time t^{n-1} once Newton iterations have converged. We refer to (3) as the high-fidelity, FOM for problem (1). In particular, we rely on a Galerkin-FE method for space approximation, and consider implicit finite difference schemes for time discretization, which do not require restrictions on Δt [48]. The high-fidelity dimension N_h is determined by the underlying mesh and the chosen FE polynomial order and can be extremely large in case of very fine meshes.

To reduce the FOM numerical complexity, we introduce a projection-based ROM, by relying on the RB method [49]. The idea of the RB method is to suitably select $N \ll N_h$ vectors of $\mathbb{R}^{N_h}$, forming the so-called RB matrix $\mathbf{V} \in \mathbb{R}^{N_h \times N}$, and to generate a reduced problem by performing a Galerkin projection of the FOM onto the subspace $V_N = \mathrm{Col}(\mathbf{V}) \subset \mathbb{R}^{N_h}$ generated by these vectors. This method relies on the assumption that the reduced-order approximation can be expressed as a linear combination of few, problem-dependent, basis functions, that is

$$\mathbf{u}_h^n(\boldsymbol{\mu}) \approx \mathbf{V}\mathbf{u}_N^n(\boldsymbol{\mu})$$

for $n = 1, \dots, N_t$, where $\mathbf{u}_N^n(\boldsymbol{\mu}) \in \mathbb{R}^N$ denotes the vector of the ROM-dofs at time $t^n \geq 0$. The latter is obtained by imposing that the projection of the FOM residual computed on the ROM solution is orthogonal to the trial subspace (in the case of a Galerkin projection): given $\boldsymbol{\mu} \in \mathcal{P}$, at each time t^n, for $n = 1, \dots, N_t$, we seek $\mathbf{u}_N^n(\boldsymbol{\mu}) \in \mathbb{R}^N$ such that

$$\mathbf{V}^T \mathbf{R}(\mathbf{V}\mathbf{u}_N^n(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}) = \mathbf{0}.$$

From now on, we will denote the reduced residual $\mathbf{V}^T\mathbf{R}$ and the corresponding Jacobian $\mathbf{V}^T\mathbf{J}\mathbf{V}$ as $\mathbf{R}_N$ and $\mathbf{J}_N$, respectively. Then, the associated reduced Newton problem at time t^n reads: given $\mathbf{u}_N^{n,(0)}(\boldsymbol{\mu})$, for $k \geq 0$, find $\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}) \in \mathbb{R}^N$ such that

$$\begin{cases} \mathbf{J}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu})\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}) = -\mathbf{R}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}), \\ \mathbf{u}_N^{n,(k+1)}(\boldsymbol{\mu}) = \mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}) + \mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), \end{cases} \tag{4}$$

until a suitable stopping criterion is fulfilled.

2.1 Solution-Space Reduction: Proper Orthogonal Decomposition

In this section we provide an overview of the proper orthogonal decomposition (POD) technique used to compute the reduced basis $\mathbf{V}$ through the so-called method of snapshots [8, 14]. Let

$$\mathcal{M}_{u_h} = \{\mathbf{u}_h^n(\boldsymbol{\mu}) \in \mathbb{R}^{N_h} \mid \boldsymbol{\mu} \in \mathcal{P},\ n = 1, \dots, N_t\}$$

be the (discrete) solution manifold identified by the image of $\mathbf{u}_h$, that is, the set of all the PDE solutions for $\boldsymbol{\mu}$ varying in the parameter space and t^n in the partition of the time interval. Our goal is to approximate $\mathcal{M}_{u_h}$ with a reduced linear manifold, the *trial manifold*

$$\mathcal{M}_{u_N}^{lin} = \{\mathbf{V}\mathbf{u}_N^n(\boldsymbol{\mu}) \mid \mathbf{u}_N^n(\boldsymbol{\mu}) \in \mathbb{R}^N,\ \boldsymbol{\mu} \in \mathcal{P},\ n = 1, \dots, N_t\}.$$

To do this, given $n_s < N_h$ sampled instances of $\boldsymbol{\mu} \in \mathcal{P}$, the snapshots matrix $\mathbf{S}$ is constructed by collecting column-wise the FOM solution $\mathbf{u}_h^n(\boldsymbol{\mu}_\ell)$ at each time step $n = 1, \dots, N_t$, for $\ell = 1, \dots, n_s$, that is

$$\mathbf{S} = \left[\mathbf{u}_h^1(\boldsymbol{\mu}_1) \,|\dots|\, \mathbf{u}_h^{N_t}(\boldsymbol{\mu}_1) \,|\dots|\, \mathbf{u}_h^1(\boldsymbol{\mu}_{n_s}) \,|\dots|\, \mathbf{u}_h^{N_t}(\boldsymbol{\mu}_{n_s})\right] \in \mathbb{R}^{N_h \times N_t n_s}.$$

Sampling can be performed, e.g., through a latin hypercube sampling design, as well as through suitable low-discrepancy points sets. The POD basis $\mathbf{V} \in \mathbb{R}^{N_h \times N}$ spanning the subspace V_N is usually obtained by performing the singular value decomposition $\mathbf{S} = \mathbf{U6Z}^T$ of $\mathbf{S}$, and then collecting the first N columns of $\mathbf{U}$, corresponding to the N largest singular values stored in the diagonal matrix $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \dots, \sigma_r) \in \mathbb{R}^{n_s \times n_s}$, with $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$ and $r \leq N_h \wedge n_s$ being the rank of $\mathbf{S}$. This yields an orthonormal basis such that

$$\|\mathbf{S} - \mathbf{V}\mathbf{V}^T\mathbf{S}\|_F^2 = \min_{\mathbf{W} \in \mathbb{R}^{N_h \times N} :\, \mathbf{W}^T\mathbf{W} = \mathbf{I}_N} \|\mathbf{S} - \mathbf{W}\mathbf{W}^T\mathbf{S}\|_F^2 = \sum_{i=N+1}^{r} \sigma_i^2,$$

where $\|\cdot\|_F$ is the Frobenius norm. Hence, singular values' decay directly impacts on the size N, which can be computed as the minimum integer satisfying

$$RIC(N) = \frac{\sum_{\ell=1}^{N} \sigma_i^2}{\sum_{\ell=1}^{r} \sigma_i^2} \geq 1 - \varepsilon_{POD}^2 \tag{5}$$

for a given tolerance $\varepsilon_{POD} > 0$. In this work we exploit the so-called randomized-SVD, which offers a powerful tool to perform low-rank matrix approximations when dealing with massive datasets [34], such as high-dimensional snapshots matrices.

2.2 Hyper-Reduction: The Discrete Empirical Interpolation Method

In the case of parametrized PDEs featuring nonaffine dependence on the parameter and/or nonlinear (high-order polynomial or nonpolynomial) dependence on the field variable, a further level of reduction, known as *hyper-reduction*, must be introduced [30, 42]. Note that if nonlinearities only include quadratic (or, at most, cubic) terms and do not feature any parameter dependence, assembling of nonlinear terms in the ROM can be performed by projection of the corresponding FOM quantities, once and for all [28].

For the case at hand, the residual $\mathbf{R}_N$ and the Jacobian $\mathbf{J}_N$ appearing in the reduced Newton system (4) depend on the solution at the previous iteration and, therefore, must be computed at each step $k \geq 0$. It follows that, for any new instance of the parameter $\boldsymbol{\mu}$, we need to assemble the high-dimensional FOM-arrays before projecting them onto the reduced subspace, entailing a computational complexity which is still of order N_h. To setup an efficient offline–online computational splitting, an approximation of the nonlinear operators that is independent of the high-fidelity dimension is required.

Several techniques have been employed to provide this further level of approximation [4, 6, 13, 15, 20]; among these, DEIM has been successfully applied to stationary or quasi-static nonlinear mechanical problems [11, 27]. Its key idea is to replace the nonlinear arrays in (4) with a collateral reduced basis expansion, computed through an inexpensive interpolation procedure. In this framework, the high-dimensional residual $\mathbf{R}(\boldsymbol{\mu})$ is projected onto a reduced subspace of dimension $m < N_h$ spanned by a basis $\mathbf{\Phi}_{\mathcal{R}} \in \mathbb{R}^{N_h \times m}$

$$\mathbf{R}(\boldsymbol{\mu}) \approx \mathbf{\Phi}_{\mathcal{R}}\mathbf{r}(\boldsymbol{\mu}),$$

where $\mathbf{r}(\boldsymbol{\mu}) \in \mathbb{R}^m$ is the vector of the unknown amplitudes. The matrix $\mathbf{\Phi}_{\mathcal{R}}$ can be precomputed offline by performing POD on a set of high-fidelity residuals collected when solving (4) for n_s' training input parameters

$$\mathbf{S}_\rho = \left[\mathbf{R}(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}_\ell), t^n; \boldsymbol{\mu}_\ell)), k \geq 0\right]_{n=1,\dots,N_t}^{\ell=1,\dots,n_s'}.$$

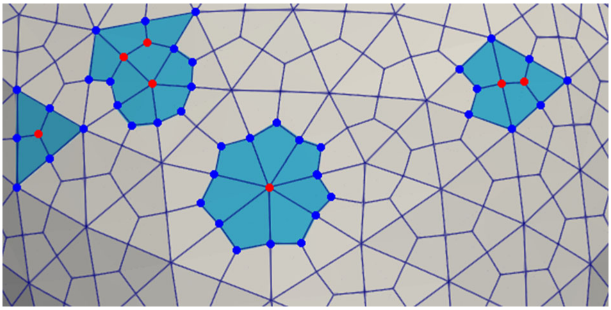


Fig. 1 Sketch of a reduced mesh for an hexahedral computational grid in two dimensions. The red dots represent the points selected by the DEIM algorithm and, together with the blue ones, yield the vertices of the elements (light blue) of the reduced mesh (Color figure online)

The unknown parameter-dependent coefficient $\mathbf{r}(\boldsymbol{\mu})$ is obtained online by collocating the approximation at the m components selected by a greedy procedure, and requiring that, for these components, $\mathbf{P}^T\mathbf{R}(\boldsymbol{\mu}) = \mathbf{P}^T\boldsymbol{\Phi}_{\mathcal{R}}\mathbf{r}(\boldsymbol{\mu})$, so that

$$\mathbf{r}(\boldsymbol{\mu}) = (\mathbf{P}^T\boldsymbol{\Phi}_{\mathcal{R}})^{-1}\mathbf{P}^T\mathbf{R}(\boldsymbol{\mu}),$$

where $\mathbf{P} \in \mathbb{R}^{N_h \times m}$ is the boolean matrix associated with the interpolation constraints. We thus define the hyper-reduced residual vector as

$$\mathbf{R}_{N,m}(\boldsymbol{\mu}) := \mathbf{V}^T\boldsymbol{\Phi}_{\mathcal{R}}(\mathbf{P}^T\boldsymbol{\Phi}_{\mathcal{R}})^{-1}\mathbf{P}^T\mathbf{R}(\boldsymbol{\mu}) \approx \mathbf{R}_N \equiv \mathbf{V}^T\mathbf{R}(\boldsymbol{\mu}).$$

Finally, the associated Jacobian approximation $\mathbf{J}_{N,m}(\boldsymbol{\mu})$ can be computed as the derivative of $\mathbf{R}_{N,m}(\boldsymbol{\mu})$ with respect to the reduced displacement, obtaining

$$\mathbf{J}_{N,m}(\boldsymbol{\mu}) = \mathbf{V}^T\boldsymbol{\Phi}_{\mathcal{R}}(\mathbf{P}^T\boldsymbol{\Phi}_{\mathcal{R}})^{-1}\mathbf{P}^T\mathbf{J}(\boldsymbol{\mu})\mathbf{V},$$

or relying on the MDEIM algorithm [42], see [11, 41].

However, the application of DEIM in the case of nonlinear time-dependent PDEs can be rather inefficient, especially when turning to complex problem which require a high number of residual basis (and, thus, of interpolation points) to ensure the convergence of the hyper-reduced Newton system

$$\begin{cases} \mathbf{J}_{N,m}(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu})\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}) = -\mathbf{R}_{N,m}(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}) \\ \mathbf{u}_N^{n,(k+1)}(\boldsymbol{\mu}) = \mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}) + \mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}). \end{cases}$$

In fact, the m points selected by the DEIM algorithm correspond to a subset of nodes of the computational mesh, which, together with the neighboring nodes (i.e. those sharing the same cell), form the so-called *reduced mesh*; see, e.g., the sketch reported in Fig. 1. Since the entries of any FE-vector are associated with the dofs of the problem, $\mathbf{P}^T\mathbf{R}(\boldsymbol{\mu})$ is computed by integrating the residual only on the quadrature points belonging to the elements forming the reduced mesh, which, nevertheless, can be rather dense.

A modification of the DEIM algorithm, UDEIM, has been proposed in [54] to exploit the sparsity of the problem and minimize the number of element function calls. However, a high number of nonlinear function evaluations is still required when the number of magic points is sufficiently large. Indeed, DEIM-based affine approximations are effective, in terms of computational costs, provided that few entries of the nonlinear terms can be cheaply computed; however, this situation does not occur neither for dynamical systems arising from the linearization of a nonlinear system around a steady state, nor when dealing with global nonpolynomial nonlinearities.

In this paper, we propose an alternative technique to perform hyper-reduction, which is independent of the underlying mesh and relies on a DNN architecture to approximate reduced residual vectors and Jacobian matrices. The introduction of a surrogate model to perform operator approximation is justified by the fact that, often, most of the CPU time needed online for each new parameter instance is required by POD-Galerkin-DEIM ROMs for assembling arrays such as residual vectors or corresponding Jacobian matrices on the reduced mesh.

3 Operator Approximation: A Deep Learning-Based Technique (Deep-HyROMnet)

To recover the offline–online efficiency of the RB method, overcoming the need to assemble the nonlinear arrays onto the computational mesh as in the case of DEIM, we present a novel projection-based method which relies on DNNs for the approximation of the nonlinear terms. We refer to this strategy as to a *hyper-reduced order model enhanced by deep neural networks* (Deep-HyROMnet). Our goal is the efficient numerical approximation of the sets

$$\mathcal{M}_{R_N} = \{\mathbf{R}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}) \in \mathbb{R}^N \mid \boldsymbol{\mu} \in \mathcal{P},\ n = 1, \dots, N_t,\ k \geq 0\},$$
$$\mathcal{M}_{J_N} = \{\mathbf{J}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}) \in \mathbb{R}^{N\times N} \mid \boldsymbol{\mu} \in \mathcal{P},\ n = 1, \dots, N_t,\ k \geq 0\},$$

which represents the *reduced residual manifold* and *reduced Jacobian manifold*, respectively, in a way that depends only on the ROM dimension N and on the number of parameters P. To achieve this task, we employ the DNN architecture developed in [23] for the so-called DL-ROM techniques. It is worthy to note that, except for the approximation error of the reduced nonlinear operators, the proposed Deep-HyROMnet approach is a physics-based method and that the computed solution satisfies the nonlinear equation of the problem at hand, up to a further approximation of ROM residual and Jacobian arrays—thus, similarly to what happened for a POD-Galerkin-DEIM ROM. The main idea of the deep learning-based operator approximation approach that replaces DEIM in our new Deep-HyROMnet strategy, is to learn the following input-to-residual and input-to-Jacobian maps, respectively:

$$\boldsymbol{\rho}_N : (\boldsymbol{\mu}, t^n, k) \longmapsto \mathbf{R}_N(\mathbf{V}\mathbf{u}^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}),$$
$$\boldsymbol{\iota}_N : (\boldsymbol{\mu}, t^n, k) \longmapsto \mathbf{J}_N(\mathbf{V}\mathbf{u}^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}),$$

provided $(\boldsymbol{\mu}, t^n, k) \in \mathcal{P} \times \{t^1, \dots, t^{N_t}\} \times \mathbb{N}^+$ as inputs and to finally replace the linear system in (4) with

$$\boldsymbol{\iota}_N(\boldsymbol{\mu}, t^n, k)\mathbf{u}^{n,(k)}(\boldsymbol{\mu}) = -\boldsymbol{\rho}_N(\boldsymbol{\mu}, t^n, k).$$

Hence, we aim at approximating the residual vector and the Jacobian matrix obtained after their projection onto the reduced space of dimension $N \ll N_h$. Indeed, performing a Galerkin projection onto the POD solution space allows to severely reduce the problem dimension from N_h to N and, hence, to ease the learning task with respect the reconstruction of the full-order residual $\mathbf{R}$ and Jacobian $\mathbf{J}$.

Remark 1 As an alternative to Newton iterative scheme, we can rely on Broyden's method [12], which belongs to the class of quasi-Newton methods. This allows to avoid the computation of the Jacobian matrix at each iteration $k \geq 0$ by relying on rank-one updates, based on residuals computed at previous iterations. However, we are able to compute Jacobian matrices very efficiently using automatic differentiation (AD), so that the computational burden is the assembling of residual vectors. For this reason, in this paper we will focus on the Newton method only, that is, the solution of problem (4).

To summarize, in the case of the Newton approach, we end up with the following reduced problem: given $\boldsymbol{\mu} \in \mathcal{P}$ and, for $n = 1, \dots, N_t$, the initial guess $\mathbf{u}_N^{n,(0)}(\boldsymbol{\mu}) = \mathbf{u}_N^{n-1}(\boldsymbol{\mu})$, find $\delta\mathbf{u}_N^{n,(k)} \in \mathbb{R}^N$ such that, for $k \geq 0$,

$$\begin{cases} \boldsymbol{\iota}_N(\boldsymbol{\mu}, t^n, k)\delta\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}) = -\boldsymbol{\rho}_N(\boldsymbol{\mu}, t^n, k), \\ \mathbf{u}_N^{n,(k+1)}(\boldsymbol{\mu}) = \mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}) + \delta\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), \end{cases} \tag{6}$$

until $\|\boldsymbol{\rho}_N(\boldsymbol{\mu}, t^n, k)\|_2 / \|\boldsymbol{\rho}_N(\boldsymbol{\mu}, t^n, 0)\|_2 < \varepsilon$, where $\varepsilon > 0$ is a given tolerance. In Algorithms 1 and 2, we report a summary of the offline and online stages of Deep-HyROMnet, respectively.

Algorithm 1 Deep-HyROMnet, offline (training) stage.

```
INPUT: $\boldsymbol{\mu}_\ell$, for $\ell = 1, \dots, n_s$, and $\boldsymbol{\mu}_{\ell'}$, for $\ell' = 1, \dots, n'_s$
OUTPUT: $\mathbf{V} \in \mathbb{R}^{N_h \times N}$
for $\ell = 1, \dots, n_s$ do
    for $n = 1, \dots, N_t$ do
        for $k \geq 0$ until convergence do
            Assemble and solve problem (3)
            Collect $\mathbf{S}_u \leftarrow \mathbf{S}_u \cup \left[\mathbf{u}_h^{n,(k)}(\boldsymbol{\mu}_\ell)\right]$ column-wise
Construct $\mathbf{V} = \mathrm{POD}(\mathbf{S}_u, \varepsilon_{POD})$
for $\ell' = 1, \dots, n'_s$ do
    for $n = 1, \dots, N_t$ do
        for $k \geq 0$ until convergence do
            Assemble and solve reduced problem (4)
            Collect $\mathbf{S}_{\boldsymbol{\rho}} \leftarrow \mathbf{S}_{\boldsymbol{\rho}} \cup \left[\mathbf{R}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}_{\ell'}), t^n; \boldsymbol{\mu}_{\ell'})\right]$ column-wise
            Collect $\mathbf{S}_{\boldsymbol{\iota}} \leftarrow \mathbf{S}_{\boldsymbol{\iota}} \cup \left[\mathbf{J}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}_{\ell'}), t^n; \boldsymbol{\mu}_{\ell'})\right]$ column-wise
Train the DNNs (see Algorithm 3)
```

Algorithm 2 Deep-HyROMnet, online (testing) stage.

```
INPUT: $\boldsymbol{\mu} \in \mathcal{P}$
OUTPUT: $\mathbf{V}\mathbf{u}_N^n(\boldsymbol{\mu}) \in \mathbb{R}^{N_h}$, for $n = 1, \dots, N_t$
for $n = 0, \dots, N_t - 1$ do
    for $k \geq 0$ until convergence do
        Compute $\boldsymbol{\rho}_N(\boldsymbol{\mu}, t^n, k)$ and $\boldsymbol{\iota}_N(\boldsymbol{\mu}, t^n, k)$ (see Algorithm 4)
        Solve hyper-reduced problem (6)
Recover $\mathbf{V}\mathbf{u}_N^n(\boldsymbol{\mu})$, for $n = 1, \dots, N_t$
```

3.1 Deep Neural Network Construction

For the sake of generality, we will focus on the DNN-based approximation of the reduced residual vector only, that is

$$\boldsymbol{\rho}_N(\boldsymbol{\mu}, t^n, k) \approx \mathbf{R}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}) \in \mathbb{R}^N.$$

In fact, by relying on a suitable transformation, we can easily write the Jacobian matrix as a vector of dimension N^2 and apply to it the same procedure described in the following for the residual vector. In particular, we define the transformation

$$vec \colon \mathbb{R}^{N\times N} \to \mathbb{R}^{N^2}, \quad vec(\mathbf{J}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu})) = \mathbf{j}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}),$$

which consists in stacking the columns of $\mathbf{J}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu})$ in a vector that is passed to the DNN as training sample, thus obtaining

$$\widetilde{\boldsymbol{\iota}}_N(\boldsymbol{\mu}, t^n, k) \approx \mathbf{j}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}) \in \mathbb{R}^{N^2}.$$

Finally, the vec operation is reverted, so that $\boldsymbol{\iota}_N(\boldsymbol{\mu}, t^n, k) = vec^{-1}(\widetilde{\boldsymbol{\iota}}_N(\boldsymbol{\mu}, t^n, k))$.

We thus aim at efficiently approximating the whole set $\mathcal{M}_{R_N}$ by means of the reduced residual trial manifold, defined as

$$\mathcal{M}_{\rho_N} = \{\boldsymbol{\rho}_N(\boldsymbol{\mu}, t^n, k) \mid \boldsymbol{\mu} \in \mathcal{P},\ n = 1, \ldots, N_t,\ k \geq 0\} \subset \mathbb{R}^N.$$

The approximation of the ROM residual $\mathbf{R}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu})$ takes the form

$$\boldsymbol{\rho}_N(\boldsymbol{\mu}, t^n, k) = \widetilde{\mathbf{R}}_N(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D) = \mathbf{f}_N^D(\boldsymbol{\phi}_q^{DF}(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_{DF}); \boldsymbol{\theta}_D)$$

where:

- $\boldsymbol{\phi}_q^{DF}(\cdot\,; \boldsymbol{\theta}_{DF}) \colon \mathbb{R}^{P+2} \to \mathbb{R}^q$ such that

$$\boldsymbol{\phi}_q^{DF}(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_{DF}) = \mathbf{R}_q(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_{DF})$$

  is a deep feedforward neural network (DFNN), consisting in the subsequent composition of a nonlinear activation function, applied to a linear transformation of the input, multiple times. Here, $\boldsymbol{\theta}_{DF}$ denotes the vector of parameters of the DFNN, collecting all the corresponding weights and biases of each layer and q is as close as possible to the input size $P+2$;
- $\mathbf{f}_N^D(\cdot\,; \boldsymbol{\theta}_D) \colon \mathbb{R}^q \to \mathbb{R}^N$ such that

$$\mathbf{f}_N^D(\mathbf{R}_q(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_{DF}); \boldsymbol{\theta}_D) = \widetilde{\mathbf{R}}_N(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)$$

  is the decoder function of a convolutional autoencoder (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 encoder function of the CAE is exploited, during the training stage only, to map the reduced residual $\mathbf{R}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu})$ associated to $(\boldsymbol{\mu}, t^n, k)$ onto a low-dimensional representation

$$\mathbf{f}_q^E(\mathbf{R}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}); \boldsymbol{\theta}_E) = \widetilde{\mathbf{R}}_q(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_E),$$

where $\mathbf{f}_q^E(\cdot\,; \boldsymbol{\theta}_E) \colon \mathbb{R}^N \to \mathbb{R}^q$ denotes the encoder function, depending upon a vector $\boldsymbol{\theta}_E$ of parameters. The choice of a CAE is due to the fact that, thanks to the shared parameters and local connectivity properties which allow to reduce the numbers of parameters of the

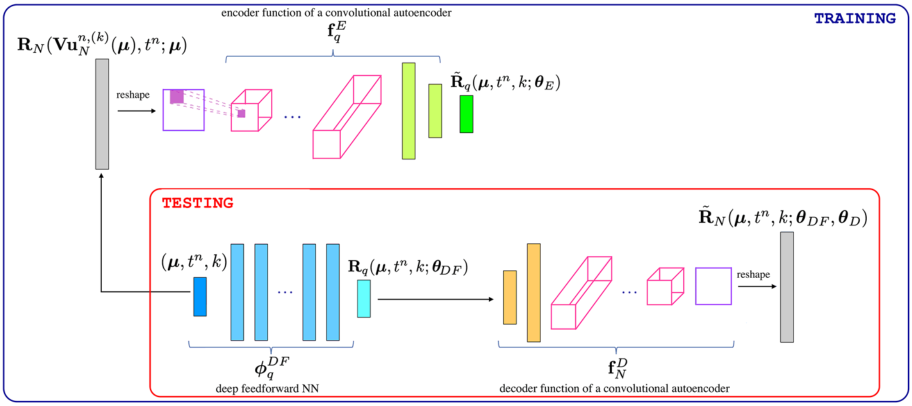

encoder function of a convolutional autoencoder
$\mathbf{f}_q^E$
TRAINING
$\mathbf{R}_N(\mathbf{Vu}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu})$
reshape
...
$\tilde{\mathbf{R}}_q(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_E)$
TESTING
$\tilde{\mathbf{R}}_N(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)$
$(\boldsymbol{\mu}, t^n, k)$
...
$\mathbf{R}_q(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_{DF})$
...
reshape
$\boldsymbol{\phi}_q^{DF}$
deep feedforward NN
$\mathbf{f}_N^D$
decoder function of a convolutional autoencoder

Fig. 2 DNN architecture for the approximation of the reduced residual vector. A similar architecture is considered to approximate the reduced Jacobian matrix

network and the number of associated computations, convolutional layers are better suited than dense layers to handle high-dimensional correlated data.

Remark 2 We point out that the input of the encoder function—that is, the reduced residual vector $\mathbf{R}_N$—is reshaped into a square matrix by rewriting its elements in row-major order, thus obtaining $\mathbf{R}_N^{reshape} \in \mathbb{R}^{\sqrt{N}\times\sqrt{N}}$. If N is not a square, the input $\mathbf{R}_N$ is zero-padded as explained in [29], and the additional elements are subsequently discarded.

Regarding the prediction of the reduced residual for new unseen instances of the inputs, computing $\boldsymbol{\rho}_N(\boldsymbol{\mu}_{test}, t^n, k)$ for any given $\boldsymbol{\mu}_{test} \in \mathcal{P}$, and for any possible $n = 1, \ldots, N_t, k \geq 0$, corresponds to the testing stage of a DFNN and of the decoder function of a convolutional AE. Thus, we discard the encoder function at testing time. The architecture used during the training stage is reported in Fig. 2; only the block highlited in the inner (red) box is then used during the testing phase.

Let $\mathbf{M} \in \mathbb{R}^{(P+2)\times N_{train}}$, with $N_{train} = n_s' N_t N_k$, be the parameter matrix of the input triples, i.e.

$$\mathbf{M} = \left[(\boldsymbol{\mu}_\ell, t^n, k)\right]_{\ell=1,\ldots,n_s', n=1,\ldots,N_t, k\geq 0}.$$

The corresponding training dataset for $\boldsymbol{\rho}_N$ is given by the reduced residual snapshots matrix $\mathbf{S}_\rho \in \mathbb{R}^{N\times N_{train}}$ defined as

$$\begin{aligned}
\mathbf{S}_\rho = [&\mathbf{R}_N(\mathbf{Vu}_N^{1,(1)}(\boldsymbol{\mu}_1), t^1; \boldsymbol{\mu}_1)|\ldots|\mathbf{R}_N(\mathbf{Vu}_N^{1,(N_{k_1})}(\boldsymbol{\mu}_1), t^1; \boldsymbol{\mu}_1)|\\
&\mathbf{R}_N(\mathbf{Vu}_N^{2,(1)}(\boldsymbol{\mu}_1), t^2; \boldsymbol{\mu}_1)|\ldots|\mathbf{R}_N(\mathbf{Vu}_N^{2,(N_{k_2})}(\boldsymbol{\mu}_1), t^2; \boldsymbol{\mu}_1)|\\
&\vdots\\
&\mathbf{R}_N(\mathbf{Vu}_N^{N_t,(1)}(\boldsymbol{\mu}_{n_s'}), t^{N_t}; \boldsymbol{\mu}_{n_s'})|\ldots|\mathbf{R}_N(\mathbf{Vu}_N^{N_t,(N_{k_{n_s'}})}(\boldsymbol{\mu}_{n_s'}), t^{N_t}; \boldsymbol{\mu}_{n_s'})\ ]\\
= &\left[\mathbf{R}_N(\mathbf{Vu}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}_\ell)\right]_{\ell=1,\ldots,n_s', n=1,\ldots,N_t, k\geq 0},
\end{aligned}$$

that is, by the matrix collecting column-wise ROM residuals computed for n_s' sampled parameters $\boldsymbol{\mu}_\ell \in \mathcal{P}$, at different time instances $t^1, \ldots, t^{N_t}$ and for each Newton iteration $k \geq 0$. The training stage consists in solving the following optimization problem in the weights variable $\boldsymbol{\theta} = (\boldsymbol{\theta}_E, \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)$:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N_{train}} \sum_{\ell=1}^{n_s'} \sum_{n=1}^{N_t} \sum_{k=0}^{N_k} \mathcal{L}(\boldsymbol{\mu}_\ell, t^n, k; \boldsymbol{\theta}) \rightarrow \min_{\boldsymbol{\theta}}$$

where

$$\begin{aligned}\mathcal{L}(\boldsymbol{\mu}_\ell, t^n, k; \boldsymbol{\theta}) = &\frac{1}{2}\|\mathbf{R}_N(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}_\ell), t^n; \boldsymbol{\mu}_\ell) - \widetilde{\mathbf{R}}_N(\boldsymbol{\mu}_\ell, t^n, k; \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)\|^2 \\ &+ \frac{1}{2}\|\widetilde{\mathbf{R}}_q(\boldsymbol{\mu}_\ell, t^n, k; \boldsymbol{\theta}_E) - \mathbf{R}_q(\boldsymbol{\mu}_\ell, t^n, k; \boldsymbol{\theta}_{DF})\|^2.\end{aligned} \tag{7}$$

The loss function (7) combines the reconstruction error, i.e. the error between the ROM residual and the corresponding DNN approximation, and the error between the intrinsic coordinates and the output of the encoder. The training stage of the DNN involved in Deep-HyROMnet is detailed in Algorithm 3; in particular, we denote by α the training-validation splitting fraction, by η the starting learning rate of the ADAM optimizer, by N_b the batch size, by $n_b = (1-\alpha)N_{train}/N_b$ the number of minibatches and by N_e the maximum number of epochs. We recall that the total number of training samples is given by $N_{train} = n_s' N_t N_k$. The testing stage of the DNN is detailed in Algorithm 4. As suggested by [23, 24], we set $\alpha = 8 : 2$, $\eta = 10^{-4}$, $N_b = 20$ and $N_e = 10^4$. To avoid overfitting, we employ the early-stopping regularization technique [29], that is, we stop the training if (and when) the loss evaluated on the validation set does not decrease over 500 epochs. Regarding the NN architecture, a 12-layer DFNN with 50 neurons per hidden layer and q neurons in the output layer is employed, where q is problem-dependent and set equal to the intrinsic dimension, i.e. $q = P + 2$ (the time instant and the Newton iteration are considered as additional parameters); further details about the CAE architecture are summarized in Tables 1 and 2. In all cases, except for the last convolutional layer of the decoder, we consider the ELU nonlinear activation function [18], selected by assessing the impact of different activation functions on the validation loss.

Remark 3 Differently from the min-max scaling used in [23, 24], we standardize the input and output of the DNN as follows. After splitting the data into training and validation sets according to a user-defined training-validation splitting fraction $\boldsymbol{\alpha}$, $\mathbf{M} = \left[\mathbf{M}^{train}, \mathbf{M}^{val}\right]$ and $\mathbf{S}_{\boldsymbol{\rho}} = \left[\mathbf{S}_{\boldsymbol{\rho}}^{train}, \mathbf{S}_{\boldsymbol{\rho}}^{val}\right]$, we define for each row of the training set the corresponding mean and standard deviation

Table 1 Attributes of convolutional and dense layers of the encoder function of the CAE

Layer	Input dimension	Output dimension	Kernel size	# filters	Stride	Padding
1			[5,5]	8	1	SAME
2			[5,5]	16	2	SAME
3			[5,5]	32	2	SAME
4			[5,5]	64	2	SAME
5	N	64				
6	64	q				

Table 2 Attributes of dense and transposed convolutional layers of the decoder function of the CAE

Layer	Input dimension	Output dimension	Kernel size	# filters	Stride	Padding
1	q	64				
2	64	N				
3			[5,5]	64	2	SAME
4			[5,5]	32	2	SAME
5			[5,5]	16	2	SAME
6			[5,5]	1	1	SAME

Algorithm 3 Training stage for the DNN, based on Algorithm 1 of [23]

```
INPUT: $\mathbf{M} \in \mathbb{R}^{(P+2)\times N_{train}}$, $\mathbf{S}_{\boldsymbol{\rho}} \in \mathbb{R}^{N \times N_{train}}$, $\alpha$, $\eta$, $N_b$, $n_b$, $N_e$, early-stopping criterion
OUTPUT: $\boldsymbol{\theta}^* = (\boldsymbol{\theta}_E^*, \boldsymbol{\theta}_{DF}^*, \boldsymbol{\theta}_D^*)$ (optimal)
 1: Randomly shuffle M and S
 2: Split data in $\mathbf{M} = \left[\mathbf{M}^{train}, \mathbf{M}^{val}\right]$ and $\mathbf{S}_{\boldsymbol{\rho}} = \left[\mathbf{S}_{\boldsymbol{\rho}}^{train}, \mathbf{S}_{\boldsymbol{\rho}}^{val}\right]$ (according to $\alpha$)
 3: Normalize M and S according to (8)
 4: Randomly initialize $\boldsymbol{\theta}^0 = (\boldsymbol{\theta}_E^0, \boldsymbol{\theta}_{DF}^0, \boldsymbol{\theta}_D^0)$
 5: $n_e = 0$
 6: while ¬early-stopping and $n_e \leq N_e$ do
 7:   for $k = 1, \dots, n_b$ do
 8:     Sample a minibatch $(\mathbf{M}^{batch}, \mathbf{S}^{batch}) \subset (\mathbf{M}^{train}, \mathbf{S}^{train})$
 9:     $\mathbf{S}^{batch} = reshape(\mathbf{S}^{batch})$
10:     $\widetilde{\mathbf{S}}_q^{batch}(\boldsymbol{\theta}_E^{n_b n_e + k}) = \mathbf{f}_q^E(\mathbf{S}^{batch}; \boldsymbol{\theta}_E^{n_b n_e + k})$
11:     $\mathbf{S}_q^{batch}(\boldsymbol{\theta}_{DF}^{n_b n_e + k}) = \boldsymbol{\phi}_q^{DF}(\mathbf{M}^{batch}; \boldsymbol{\theta}_{DF}^{n_b n_e + k})$
12:     $\widetilde{\mathbf{S}}_N^{batch}(\boldsymbol{\theta}_{DF}^{n_b n_e + k}, \boldsymbol{\theta}_D^{n_b n_e + k}) = \mathbf{f}_N^D(\mathbf{S}_q^{batch}(\boldsymbol{\theta}_{DF}^{n_b n_e + k}); \boldsymbol{\theta}_D^{n_b n_e + k})$
13:     $\widetilde{\mathbf{S}}_N^{batch} = reshape(\widetilde{\mathbf{S}}_N^{batch})$
14:     Accumulate loss (7) on $(\mathbf{M}^{batch}, \mathbf{S}^{batch})$ and compute $\hat{\nabla}_{\boldsymbol{\theta}} \mathcal{J}$
15:     $\boldsymbol{\theta}^{n_b n_e + k + 1} = \text{ADAM}(\eta, \hat{\nabla}_{\boldsymbol{\theta}} \mathcal{J}, \boldsymbol{\theta}^{n_b n_e + k})$
16:   Repeat instructions 9-13 on $(\mathbf{M}^{val}, \mathbf{S}^{val})$ to evaluate early-stopping criterion
17:   $n_e = n_e + 1$
```

Algorithm 4 Testing stage for the DNN, based on Algorithm 2 of [23]

```
INPUT: $(\boldsymbol{\mu}, t^n, k) \in \mathcal{P} \times \{t^1, \dots, t^{N_t}\} \times \mathbb{N}^+$, $(\boldsymbol{\theta}_{DF}^*, \boldsymbol{\theta}_D^*)$ (optimal)
OUTPUT: $\widetilde{\mathbf{S}}_N$ (i.e. $\boldsymbol{\rho}_N(\boldsymbol{\mu}, t^n, k)$ or $\boldsymbol{\iota}_N(\boldsymbol{\mu}, t^n, k)$)
1: $\mathbf{S}_q(\boldsymbol{\theta}_{DF}^*) = \boldsymbol{\phi}_q^{DF}(\boldsymbol{\mu}, t^n, k; \boldsymbol{\theta}_{DF}^*)$
2: $\widetilde{\mathbf{S}}_N(\boldsymbol{\theta}_{DF}^*, \boldsymbol{\theta}_D^*) = \mathbf{f}_N^D(\mathbf{S}_q(\boldsymbol{\theta}_{DF}^*); \boldsymbol{\theta}_D^*)$
3: $\widetilde{\mathbf{S}}_N = reshape(\widetilde{\mathbf{S}}_N)$
```

$$M_{mean}^i = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} M_{ij}^{train}, \quad M_{sd}^i = \sqrt{\frac{1}{N_{train}-1} \sum_{j=1}^{N_{train}} (M_{ij}^{train} - M_{mean}^i)^2},$$

so that parameters are normalized by applying the following transformation

$$M_{ij}^{train} \mapsto \frac{M_{ij}^{train} - M_{mean}^i}{M_{sd}^i}, \quad i = 1, \dots, P+2, \quad j = 1, \dots, N_{train}, \tag{8}$$

that is, each feature of the training parameter matrix is standardized. The same procedure is applied to the training snapshots matrix $\mathbf{S}_{\boldsymbol{\rho}}^{train}$ by replacing M_*^i with S_*^i, where $* \in \{mean, sd\}$ respectively. Transformation (8) is applied to the validation and testing sets as well, but considering the mean and the standard deviation computed over the training set. In order to rescale the reconstructed solution to the original values, we apply the inverse transformation.

4 Numerical Results

In this section we assess the performances of the proposed Deep-HyROMnet strategy on different applications related to parametrized nonlinear time-dependent PDE problems, focusing on structural mechanics. In particular, we consider (1) a series of structural tests on a rectangular beam, with different loading conditions and a simple nonlinear constitutive law, and then (2) a test case on an idealized left ventricle geometry, simulating cardiac contraction.

4.1 Nonlinear Elastodynamics

Let us consider a continuum body $\mathcal{B}$ embedded in a three-dimensional Euclidean space. For a given parameter vector $\boldsymbol{\mu} \in \mathcal{P}$, the displacement vector field

$$\mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu}) = \chi(\mathbf{X}, t; \boldsymbol{\mu}) - \mathbf{X}$$

relates the material position $\mathbf{X}$ of a particle in the reference configuration Ω_0, which we assume to coincide with the initial configuration, to its spatial position $\mathbf{x}$ in the current configuration Ω_t at time $t > 0$, being $\chi(\mathbf{X}, t; \boldsymbol{\mu}) = \mathbf{x}$ the motion of the body. The corresponding deformation gradient

$$\mathbf{F}(\mathbf{X}, t; \boldsymbol{\mu}) = \frac{\partial \chi(\mathbf{X}, t; \boldsymbol{\mu})}{\partial \mathbf{X}} = \mathbf{I} + \nabla_0 \mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu}),$$

characterizes the change of material elements during motion, so that the change in volume between the reference and the current configurations at time $t > 0$ is given by $J(\mathbf{X}, t; \boldsymbol{\mu}) = \det \mathbf{F}(\mathbf{X}, t; \boldsymbol{\mu}) > 0$. The equation of motion for a continuous medium reads as follows:

$$\rho_0 \partial_t^2 \mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu}) - \nabla_0 \cdot \mathbf{P}(\mathbf{F}(\mathbf{X}, t; \boldsymbol{\mu})) = \mathbf{b}_0(\mathbf{X}, t; \boldsymbol{\mu}), \qquad \mathbf{X} \in \Omega_0,\ t > 0$$

where ρ_0 is the density of the body, $\mathbf{P}(\mathbf{F})$ is the first Piola-Kirchhoff stress tensor and $\mathbf{b}_0$ is an external body force. Proper boundary and initial conditions must be specified to ensure the well-posedness of the problem. In addition, we need a constitutive equation for $\mathbf{P}$, that is, a stress-strain relationship describing the material behavior. Here, we consider hyperelastic materials, for which the existence of a strain-energy density function $\mathcal{W}\colon Lin^+ \to \mathbb{R}$ such that

$$\mathbf{P}(\mathbf{F}) = \frac{\partial \mathcal{W}(\mathbf{F})}{\partial \mathbf{F}}$$

is postulated. Note that, since $\mathbf{F}$ depends on the displacement $\mathbf{u}$, we can equivalently write $\mathbf{P}(\mathbf{F})$ or $\mathbf{P}(\mathbf{u})$.

The strong formulation of a general initial boundary-valued problem in elastodynamics thus reads as follows: given a body force $\mathbf{b}_0 = \mathbf{b}_0(\mathbf{X}, t; \boldsymbol{\mu})$, a prescribed displacement

$\bar{\mathbf{u}} = \bar{\mathbf{u}}(\mathbf{X}, t; \boldsymbol{\mu})$ and surface traction $\bar{\mathbf{T}} = \bar{\mathbf{T}}(\mathbf{X}, t, \mathbf{N}; \boldsymbol{\mu})$, find the unknown displacement field $\mathbf{u}(\boldsymbol{\mu})\colon \Omega_0 \times (0, T] \to \mathbb{R}^3$ so that

$$\begin{cases} \rho_0 \partial_t^2 \mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu}) - \nabla_0 \cdot \mathbf{P}(\mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu})) = \mathbf{b}_0(\mathbf{X}, t; \boldsymbol{\mu}) & \text{in} \quad \Omega_0 \times (0, T] \\ \mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu}) = \bar{\mathbf{u}}(\mathbf{X}, t; \boldsymbol{\mu}) & \text{on} \quad \Gamma_0^D \times (0, T] \\ \mathbf{P}(\mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu}))\mathbf{N} = \bar{\mathbf{T}}(\mathbf{X}, t, \mathbf{N}; \boldsymbol{\mu}) & \text{on} \quad \Gamma_0^N \times (0, T] \\ \mathbf{P}(\mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu}))\mathbf{N} + \alpha \mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu}) + \beta \partial_t \mathbf{u}(\mathbf{X}, t; \boldsymbol{\mu}) = \mathbf{0} & \text{on} \quad \Gamma_0^R \times (0, T] \\ \mathbf{u}(\mathbf{X}, 0; \boldsymbol{\mu}) = \mathbf{u}_0(\mathbf{X}; \boldsymbol{\mu}), \quad \partial_t \mathbf{u}(\mathbf{X}, 0; \boldsymbol{\mu}) = \dot{\mathbf{u}}_0(\mathbf{X}; \boldsymbol{\mu}) & \text{in} \quad \Omega_0 \times \{0\} \end{cases} \tag{9}$$

where $\mathbf{N}$ is the outer normal unit vector and $\alpha, \beta \in \mathbb{R}$. The boundary of the reference domain is partitioned such that $\Gamma_0^D \cup \Gamma_0^N \cup \Gamma_0^R = \Gamma$, with $\Gamma_0^i \cap \Gamma_0^j = \emptyset$ for $i, j \in \{D, N, R\}$. This equation is inherently nonlinear and additional source of nonlinearity is introduced in the material law, i.e. when using a nonlinear strain-energy density function $\mathcal{W} = \mathcal{W}(\mathbf{F})$, which is often the case of engineering applications. For the sake of simplicity, in all test cases, we neglect the body forces $\mathbf{b}_0(\boldsymbol{\mu})$ and consider zero initial conditions $\mathbf{u}_0(\boldsymbol{\mu}) = \dot{\mathbf{u}}_0(\boldsymbol{\mu}) = \mathbf{0}$. Regarding boundary conditions, we consider $\bar{\mathbf{u}}(\boldsymbol{\mu}) = \mathbf{0}$ on the Dirichlet boundary Γ_0^D and always assume $\alpha = \beta = 0$, so that we actually impose homogeneous Neumann conditions on Γ_0^R. Finally, the traction vector is given by

$$\bar{\mathbf{T}}(\mathbf{X}, t, \mathbf{N}; \boldsymbol{\mu}) = -\mathbf{g}(t; \boldsymbol{\mu}) J \mathbf{F}^{-T} \mathbf{N},$$

where $\mathbf{g}(t; \boldsymbol{\mu})$ represents an external load and will be specified according to the application at hand. Finally, the residual in (2) is given by

$$\begin{aligned} \mathbf{R}(\mathbf{u}_h^n(\boldsymbol{\mu}), t^n; \boldsymbol{\mu}) := & \left(\frac{\rho_0}{\Delta t^2} \mathcal{M} + \frac{1}{\Delta t} \mathcal{F}_\beta^{int} + \mathcal{F}_\alpha^{int} \right) \mathbf{u}_h^n(\boldsymbol{\mu}) + \mathcal{S}(\mathbf{u}_h^n(\boldsymbol{\mu})) \\ & - \left(\frac{2\rho_0}{\Delta t^2} \mathcal{M} + \frac{1}{\Delta t} \mathcal{F}_\beta^{int} \right) \mathbf{u}_h^{n-1}(\boldsymbol{\mu}) + \frac{\rho_0}{\Delta t^2} \mathcal{M} \mathbf{u}_h^{n-2}(\boldsymbol{\mu}) - \mathcal{F}^{ext,n}(\boldsymbol{\mu}), \end{aligned}$$

for $n = 1, \ldots, N_t$, where $\mathbf{u}_h^0(\boldsymbol{\mu})$ and $\mathbf{u}_h^{-1}(\boldsymbol{\mu})$ are known for the initial condition, and

$$\begin{aligned} [\mathcal{M}]_{ij} &= \int_{\Omega_0} \boldsymbol{\varphi}_j \cdot \boldsymbol{\varphi}_i d\Omega, \\ [\mathcal{F}_\beta^{int}]_{ij} &= \int_{\Gamma_0^R} \beta\, \boldsymbol{\varphi}_j \cdot \boldsymbol{\varphi}_i d\Gamma, \qquad [\mathcal{F}_\alpha^{int}]_{ij} = \int_{\Gamma_0^R} \alpha\, \boldsymbol{\varphi}_j \cdot \boldsymbol{\varphi}_i d\Gamma, \\ [\mathcal{S}(\mathbf{u}_h^n(\boldsymbol{\mu}))]_i &= \int_{\Omega_0} \mathbf{P}(\mathbf{u}_h^n(\boldsymbol{\mu}))\colon \nabla \boldsymbol{\varphi}_i d\Omega, \\ [\mathcal{F}^{ext,n}(\boldsymbol{\mu})]_i &= \int_{\Gamma_0^N} \bar{\mathbf{T}}^n(\mathbf{N}; \boldsymbol{\mu}) \cdot \boldsymbol{\varphi}_i d\Gamma + \int_{\Omega_0} \mathbf{b}_0^n(\boldsymbol{\mu}) \cdot \boldsymbol{\varphi}_i d\Omega, \end{aligned}$$

for all $i, j = 1, \ldots, N_h$, being $\{\boldsymbol{\varphi}_i\}_{i=1}^{N_h}$ a basis for the FE space.

As a measure of accuracy of the reduced approximations with respect to the FOM solution, we consider time-averaged L^2-errors of the displacement vector, that are defined as follows:

$$\begin{aligned} \epsilon_{abs}(\boldsymbol{\mu}) &= \frac{1}{N_t} \sum_{n=1}^{N_t} \|\mathbf{u}_h(\cdot, t^n; \boldsymbol{\mu}) - \mathbf{V}\mathbf{u}_N(\cdot, t^n; \boldsymbol{\mu})\|_2, \\ \epsilon_{rel}(\boldsymbol{\mu}) &= \frac{1}{N_t} \sum_{n=1}^{N_t} \frac{\|\mathbf{u}_h(\cdot, t^n; \boldsymbol{\mu}) - \mathbf{V}\mathbf{u}_N(\cdot, t^n; \boldsymbol{\mu})\|_2}{\|\mathbf{u}_h(\cdot, t^n; \boldsymbol{\mu})\|_2}. \end{aligned} \tag{10}$$

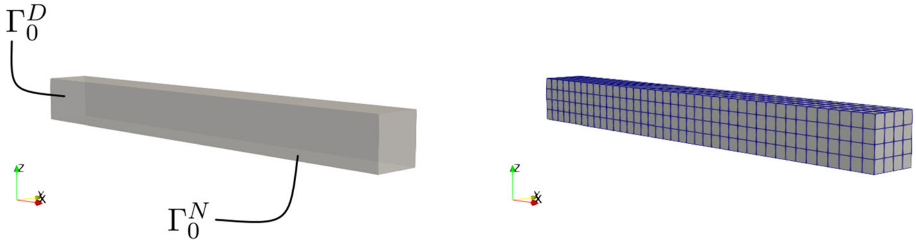


Fig. 3 Rectangular beam geometry (left) and computational grid (right)

The CPU time ratio, that is the ratio between FOM and ROM computational times, is used to measure efficiency, since it represents the speed-up offered by the ROM with respect to the FOM. The code is implemented in Python in our software package pyfe$^{\text{x}}$, a Python binding with the in-house Finite Element library `life`$^{\text{x}}$ (https://lifex.gitlab.io/lifex), a high-performance C++ library based on the `deal.II` (https://www.dealii.org) Finite Element core [3]. Computations have been performed on a PC desktop computer with 3.70GHz Intel Core i5-9600K CPU and 16GB RAM, except for the training of the DNNs and the prediction of the reduced nonlinear operators that have been carried out on a Tesla V100 32GB GPU.

Furthermore, we introduce the following measures to evaluate the performance of the DNNs during the training process:

$$\begin{aligned} \epsilon_{rel,\mathbf{R}_N} &= \frac{1}{n_s' N_t} \sum_{\ell=1}^{n_s'} \sum_{n=1}^{N_t} \mathbf{E}(\mathbf{R}_N, \widetilde{\mathbf{R}}_N; \boldsymbol{\mu}_\ell, t^n), \\ \epsilon_{rel,\mathbf{J}_N} &= \frac{1}{n_s' N_t} \sum_{\ell=1}^{n_s'} \sum_{n=1}^{N_t} \mathbf{E}(vec(\mathbf{J}_N), vec(\widetilde{\mathbf{J}}_N); \boldsymbol{\mu}_\ell, t^n), \end{aligned} \tag{11}$$

for residual vectors and Jacobian matrices, respectively, where

$$\mathbf{E}(\mathbf{w}, \widetilde{\mathbf{w}}; \boldsymbol{\mu}_\ell, t^n) = \left(\frac{\sqrt{\sum_{k=1}^{N_{k_\ell}} \|\mathbf{w}(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}_\ell), t^n; \boldsymbol{\mu}_\ell) - \widetilde{\mathbf{w}}(\boldsymbol{\mu}_\ell, t^n, k; \boldsymbol{\theta}_{DF}, \boldsymbol{\theta}_D)\|^2}}{\sqrt{\sum_{k=1}^{N_{k_\ell}} \|\mathbf{w}(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}_\ell), t^n; \boldsymbol{\mu}_\ell)\|^2}} \right).$$

4.2 Deformation of a Clamped Rectangular Beam

The first series of test cases represents a typical structural mechanical problem, with reference geometry $\bar{\Omega}_0 = [0, 10^{-2}] \times [0, 10^{-3}] \times [0, 10^{-3}]$ m^3, reported in Fig. 3.

We consider a nearly-incompressible neo-Hookean material, which is characterized by the following strain density energy function

$$\mathcal{W}(\mathbf{F}) = \frac{G}{2}(\mathcal{I}_1 - 3) + \frac{K}{4}((J-1)^2 + \ln^2(J)),$$

where $G > 0$ is the shear modulus, $\mathcal{I}_1 = J^{-\frac{2}{3}} \det(\mathbf{C})$ and the latter term is needed to enforce incompressibility, being the bulk modulus $K > 0$ the penalization factor. This choice leads to the following first Piola-Kirchhoff stress tensor, characterized by a nonpolynomial nonlinearity,

$$\mathbf{P}(\mathbf{F}) = GJ^{-\frac{2}{3}} \left(\mathbf{F} - \frac{1}{3}\mathcal{I}_1 \mathbf{F}^T \right) + \frac{K}{2} J \left(J - 1 + \frac{1}{J} \ln(J) \right) \mathbf{F}^T.$$

The beam is clamped at the left-hand side, that is, Dirichlet boundary conditions are imposed on the left face $x = 0$, whilst a pressure load changing with the deformed surface orientation is applied to the entire bottom face $z = 0$ (i.e. Γ_0^N). Homogeneous Neumann conditions are applied on the remaining boundaries (i.e. Γ_0^R with $\alpha = \beta = 0$). As possible functions for the external load $\mathbf{g}(t; \boldsymbol{\mu})$, we choose

1. A linear function $\mathbf{g}(t; \boldsymbol{\mu}) = \widetilde{p}\, t/T$;
2. A triangular or hat function $\mathbf{g}(t; \boldsymbol{\mu}) = \widetilde{p} \left(2t\, \chi_{\left(0, \frac{T}{2}\right]}(t) + 2(T - t)\, \chi_{\left(\frac{T}{2}, T\right]}(t) \right)$;
3. A step function $\mathbf{g}(t; \boldsymbol{\mu}) = \widetilde{p}\, \chi_{\left(0, \frac{T}{3}\right]}(t)$, so that the presence of the inertial term is not negligible.

Here, $\widetilde{p} > 0$ is a parameter controlling the maximum load. The FOM is built on a hexahedral mesh with 640 elements and 1025 vertices, resulting in a high-fidelity dimension $N_h = 3075$ (since $\mathbb{Q}_1$-FE are employed). The resulting computational mesh in the reference configuration is reported in Fig. 3.

The following sections are organized as follows: first, we analyze the accuracy and the efficiency of the POD-Galerkin ROM without hyper-reduction with respect to the POD tolerance ε_{POD}, thus resulting in reduced subspaces of different dimensions $N \in \mathbb{N}$. Then, for a fixed basis $\mathbf{V} \in \mathbb{R}^{N_h \times N}$, POD-Galerkin-DEIM approximation capabilities are investigated for different sizes of the reduced mesh, associated with different values of the DEIM tolerance ε_{DEIM} for the computation of the residual basis $\boldsymbol{\Phi}_{\mathcal{R}} \in \mathbb{R}^{N_h \times m}$. Finally, the performances of Deep-HyROMnet are assessed and compared to those of DEIM-based hyper-ROMs.

4.2.1 Test Case 1: Linear Function for the Pressure Load

Let us consider the parametrized linear function

$$\mathbf{g}(t; \boldsymbol{\mu}) = \widetilde{p}\, t/T,$$

for the pressure load, describing a situation in which a structure is progressively loaded. We choose a time interval $t \in [0, 0.25]$ s and employ a uniform time step $\Delta t = 5 \times 10^{-3}$ s for the time discretization scheme, resulting in a total number of 50 time iterations. As parameters, we consider:

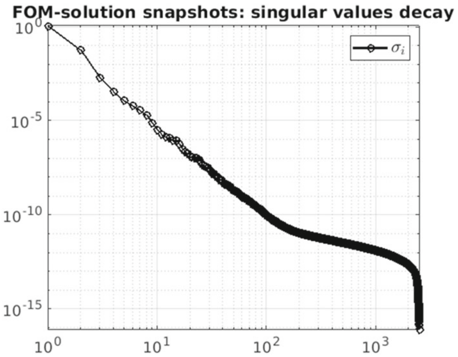
- The shear modulus $G \in [0.5 \times 10^4, 1.5 \times 10^4]$ Pa;
- The bulk modulus $K \in [2.5 \times 10^4, 7.5 \times 10^4]$ Pa;
- The external load parameter $\widetilde{p} \in [2, 6]$ Pa.

Given a training set of $n_s = 50$ points generated from the three-dimensional parameter space $\mathcal{P}$ through latin hypercube sampling (LHS), we compute the reduced basis $\mathbf{V} \in \mathbb{R}^{N_h \times N}$ using the POD method with tolerance

$$\varepsilon_{POD} \in \{10^{-3}, 5 \times 10^{-4}, 10^{-4}, 5 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6}, 10^{-6}\}.$$

The corresponding reduced dimensions are $N = 3, 4, 5, 6, 8, 9$ and 15, respectively. In Fig. 4 we show the singular values of the snapshot matrix related to the FOM displacement $\mathbf{u}_h$, where a rapid decay of the plotted quantity means that a small number of RB functions is needed to correctly approximate the FOM solution.

Fig. 4 Test case 1. Decay of the singular values of the FOM solution snapshots matrix



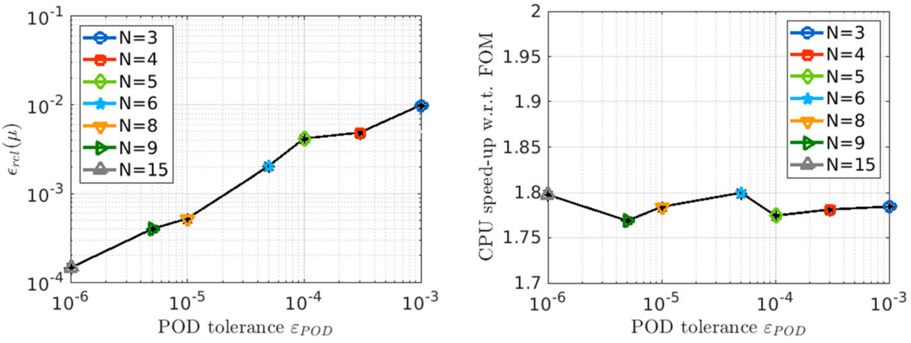


Fig. 5 Test case 1. Average over 50 testing parameters of relative error ϵ_{rel} (left) and average speed-up (right) of ROM without hyper-reduction

The average relative error ϵ_{rel} between the FOM and the POD-Galerkin ROM solutions computed over a testing set of 50 randomly chosen parameters, different from the ones used to compute the solution snapshots, is reported in Fig. 5, together with the CPU time ratio. The approximation error decreases up to an order of magnitude when reducing the POD tolerance ε_{POD} from 10^{-3} to 10^{-6}, corresponding to an increase of the RB dimension from $N = 3$ to $N = 15$. Despite the low dimension of the POD space, the computational speed-up achieved by the reduced model is negligible. This is due to the fact that the ROM still depends on the FOM dimension N_h during the online stage. For this reason, we need to rely on suitable hyper-reduction techniques.

For the construction of the hyper-reduced models, i.e. POD-Galerkin-DEIM ROMs and Deep-HyROMnet, we first need to compute snapshots from the ROM solutions for given parameter values and time instants, in order to build either the DEIM residual basis $\mathbf{\Phi}_{\mathcal{R}}$ or train the DNNs. To this goal, we choose a POD-Galerkin ROM with dimension $N = 4$, yielding a good balance between accuracy and computational effort for the test case at hand, and perform ROM simulations for a given set of $n'_s = 200$ parameter samples to collect residual and Jacobian data. To investigate the impact of hyper-reduction on the ROM solution error, we compute the DEIM basis $\mathbf{\Phi}_{\mathcal{R}}$ for approximating the residual using different DEIM tolerances,

$$\varepsilon_{DEIM} \in \{10^{-3}, 5 \times 10^{-4}, 10^{-4}, 5 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6}, 10^{-6}\},$$

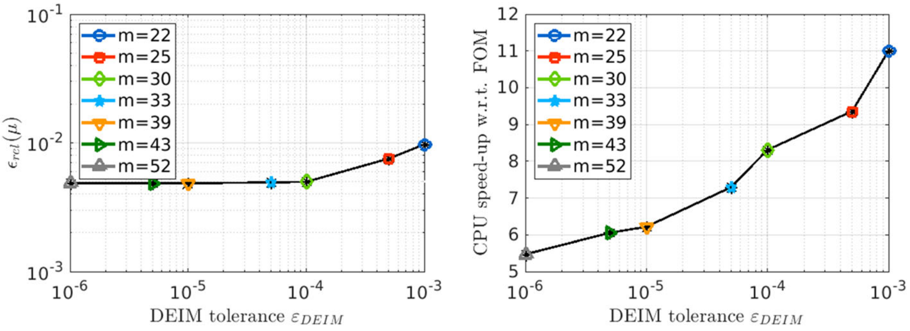


Fig. 6 Test case 1. Average over 50 testing parameters of relative error ϵ_{rel} (left) and average speed-up (right) of POD-Galerkin-DEIM with $N = 4$

Table 3 Test case 1. Computational data related to POD-Galerkin-DEIM with $N = 4$ and different values of m

DEIM tolerance ε_{DEIM}	5×10^{-4}	5×10^{-5}	5×10^{-6}
DEIM interpolation dofs m	25	33	43
Reduced mesh elements (total: 640)	86	115	168
Online CPU time	2.8 s	3.6 s	4.3 s
System construction [$*$]	78%	83%	88%
System solution	0.16%	0.13%	0.09%
[$*$] System construction for each Newton iteration	0.02 s	0.02 s	0.03 s
Residual assembling	89%	87%	88%
Jacobian computing through AD	0.6%	0.4%	0.5%
Computational speed-up	× 9.4	× 7.3	× 6.0
Time-averaged $L^2(\Omega_0)$-absolute error	3×10^{-5}	2×10^{-5}	2×10^{-5}
Time-averaged $L^2(\Omega_0)$-relative error	8×10^{-3}	5×10^{-3}	5×10^{-3}

corresponding to $m = 22, 25, 30, 33, 39, 43, 51$, respectively, DEIM basis functions. Larger tolerances were not sufficient to ensure the convergence of the Newton method for all considered combinations of parameters, so that higher speed-ups cannot be achieved by decreasing the basis dimension m. The average relative error ϵ_{rel} is evaluated over the testing set and plotted in Fig. 6, as well as the CPU time ratio. To compute the high-fidelity solutions, 26 s are required in average, while a POD-Galerkin-DEIM ROM, with $N = 4$ and $m = 22$, requires only 2.4 s, thus yielding a speed-up about 11 times compared to the FOM.

Data related to the performances of the POD-Galerkin-DEIM ROMs for $N = 4$ and different values of m are shown in Table 3. The number of elements of the reduced mesh represents a small percentage of the one forming the original grid, so that the cost related to the residual assembling is remarkably alleviated. Nonetheless, it is evident how the main computational bottleneck is the construction of the reduced system at each Newton iteration, and in particular the assembling of the residual vector on the reduced mesh, which requires between 78% and 88% of the total (online) CPU time. In particular, almost 90% of this computational time is required for assembling the residual $\mathbf{R}(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}), t^n; \boldsymbol{\mu})$ on the reduced mesh, while computing the associated Jacobian matrix using the AD tool takes less than 1%.

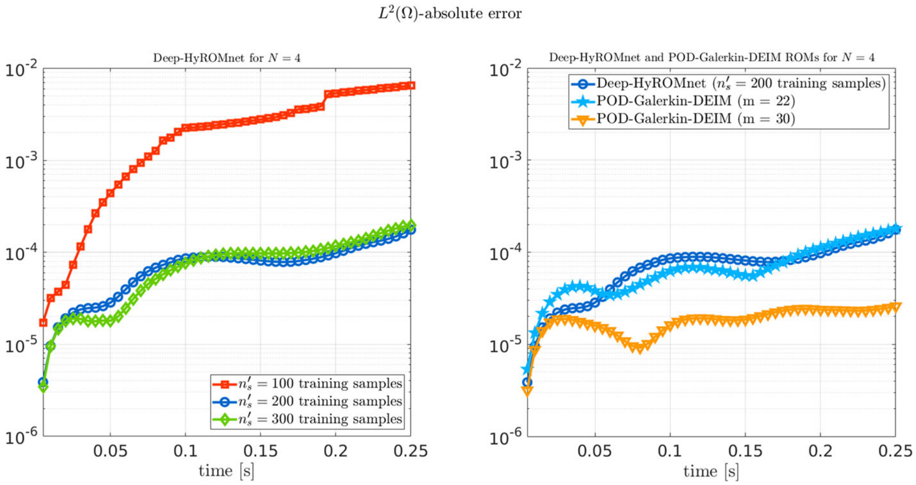


Fig. 7 Test case 1. Evolution in time of the average $L^2(\Omega_0)$-absolute error for $N = 4$ computed using Deep-HyROMnet for different sizes of the training set (left) and that obtained using POD-Galerkin-DEIM and Deep-HyROMnet (for $n'_s = 200$) hyper-ROMs (right)

The performances of Deep-HyROMnet are first analyzed by considering different sizes of the training set, and then compared in terms of both accuracy and efficiency with those of different POD-Galerkin-DEIM ROMs.

From the evolution of the $L^2(\Omega)$-absolute error displayed in Fig. 7 (left), averaged over 50 testing parameters values, we observe that relying on a training set of $n'_s = 200$ ROM simulations results in an approximation accuracy up to two orders of magnitude higher than the one obtained for $n'_s = 100$. On the other hand, further increasing the dimension of the dataset, e.g. $n'_s = 300$, does not affect the results due to the fact that the NN architecture is kept fixed and, consequently, the additional information becomes redundant. For this reason, we focus on the results obtained with $n'_s = 200$, corresponding to the same training parameters used for the construction of the DEIM bases.

The average of the absolute error ϵ_{abs}, the relative error ϵ_{rel} and the CPU time ratio are reported in Table 4. Moreover, in Table 5 we report the relative errors (11) for the approximation of the reduced nonlinear arrays by means of the DNNs, computed on both training and testing sets, showing that the approximation of the ROM residual is the most challenging task due to the higher variability of the data. In terms of efficiency, Deep-HyROMnets outperform DEIM-based hyper-ROMs substantially, being almost 100 times faster than the POD-Galerkin-DEIM ROM exploiting $m = 22$ DEIM basis functions, whilst achieving the same accuracy for $n'_s = 200$. In particular, our Deep-HyROMnet approach allows us to compute the reduced solutions in less than 0.02 s, thus yielding an overall speed-up of order $\mathcal{O}(10^3)$ compared to the FOM.

The evolution of the $L^2(\Omega_0)$-absolute error, averaged over the testing parameters, is reported in Fig. 7 for both the reduction approaches considered. The final accuracy of the hyper-ROMs equals that of the ROM without hyper-reduction, i.e. $\epsilon_{rel} \approx 10^{-2}$, meaning that the projection error dominates over the nonlinear operators approximation error. The difference between the FOM and Deep-HyROMnet solutions at time $T = 0.25$ s is shown in Fig. 8 in two scenarios.

Table 4 Test case 1. Computational data related to POD-Galerkin-DEIM ROMs and Deep-HyROMnet, for $N = 4$

	POD-Galerkin-DEIM (m =22)	POD-Galerkin-DEIM (m =30)	Deep-HyROMnet
Computational speed-up	× 11	× 8	× 1300
Avg. CPU time	2 s	3 s	0.02 s
Time-avg. $L^2(\Omega_0)$-abs. err.	7.4×10^{-5}	1.9×10^{-5}	7.7×10^{-5}
Time-avg. $L^2(\Omega_0)$-rel. err.	9.7×10^{-3}	5.0×10^{-3}	8.3×10^{-3}

Table 5 Test case 1. Accuracy of the DNNs predictions on training and test sets

	Training set error	Test set error
$\epsilon_{rel,\mathbf{R}_N}$	0.00835	0.00925
$\epsilon_{rel,\mathbf{J}_N}$	0.00043	0.00093

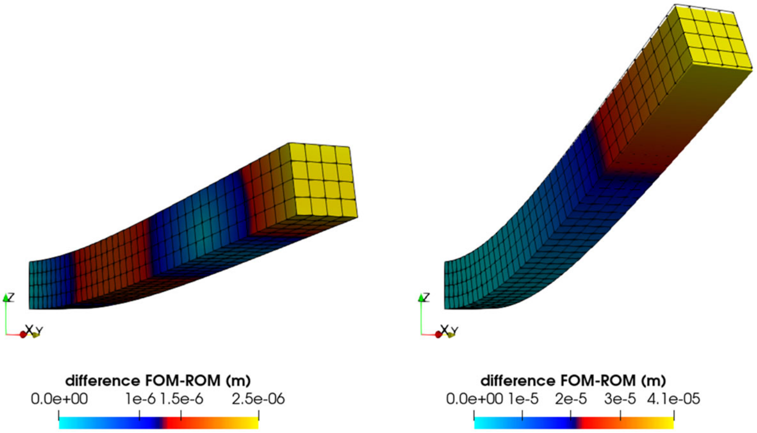


Fig. 8 Test case 1. FOM (wireframe) and Deep-HyROMnet (colored) solutions at time $T = 0.25$ s for $\boldsymbol{\mu} = [1.3225 \times 10^4$ Pa, 3.9875×10^4 Pa, 3.43 Pa] (left) and $\boldsymbol{\mu} = [0.6625 \times 10^4$ Pa, 5.8625×10^4 Pa, 4.89 Pa] (right) (Color figure online)

Table 6 collects the offline CPU time required for building the POD basis and collecting the ROM nonlinear operators, as well as the offline GPU time needed for training the DNN. Moreover, we report the total offline computational time—assuming that the training stages are run sequentially—and the *break even point*, that is the number of online FOM simulations that make the construction of the ROM affordable. We recall that FOM and POD-Galerkin ROM simulations are run in serial on a PC desktop computer with 3.70GHz Intel Core i5-9600K CPU and 16GB RAM, whereas the DNN are trained on a Tesla V100 32GB GPU.

In order to improve the accuracy of the reduced solution, we should consider higher values of the RB dimension N. Nonetheless, by increasing the number of the POD basis functions, thus the size of the reduced nonlinear data, the task of the DNNs becomes more complex. As a matter of fact, from the results reported in Fig. 9, we observe that more training samples and a larger size of the neural networks are required when N increases, in order to achieve comparable results with the ones obtained for smaller values of the POD dimension. In

Table 6 Test case 1. Offline time required for the construction of the ROMs (for $N = 4$) and the equivalence in terms of FOM simulations

	CPU time × n_s (or n'_s)
FOM	$26\,\text{s} \times 50$
POD-Galerkin ROM	$14.5\,\text{s} \times 200$
	GPU time × #epochs
DNN for $\mathbf{R}_N$	$2.7\,\text{s} \times 5426$
DNN for $\mathbf{J}_N$	$2.8\,\text{s} \times 8785$
Offline time	12 h
break even point	1661 FOM simulations

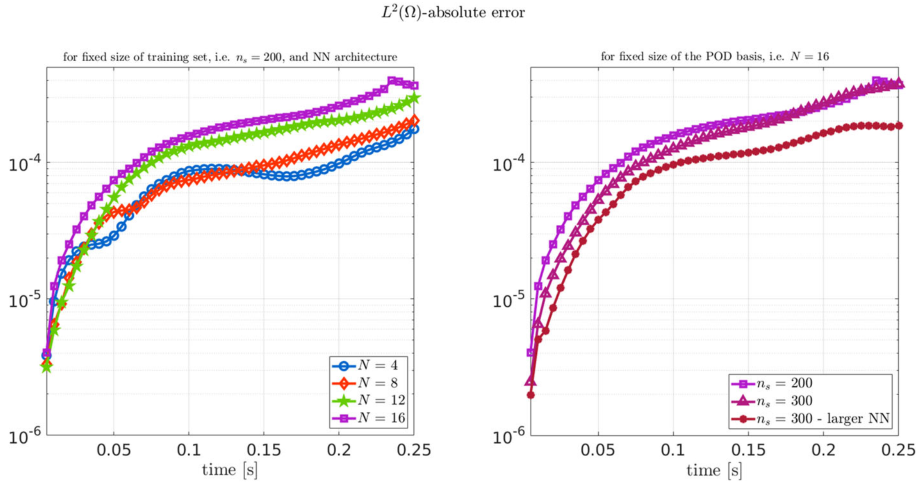


Fig. 9 Test case 1. Evolution in time of the average $L^2(\Omega_0)$-absolute error computed using Deep-HyROMnet for different RB dimensions N (left) and different features of the DNNs, i.e. amount of training data and architecture size (right)

particular, we have included an additional hidden layer in both the encoder and the decoder of the CAE, and considered a larger numbers of neurons in the dense layers. To conclude, we report in Table 7 the computational data associated with POD-Galerkin-DEIM and Deep-HyROMnet hyper-ROMs when $N = 8$, but the same number of training snapshots and the same DNN architectures of previous case (i.e. $N = 4$) are employed. As expected, the POD-Galerkin-DEIM method is able to provide more accurate approximations of the FOM solution by increasing the size m of the residual basis, albeit reducing the online speed-up substantially.

4.2.2 Test Case 2: Hat Function for the Pressure Load

Let us now consider a piecewise linear pressure load given by

$$\mathbf{g}(t; \boldsymbol{\mu}) = \widetilde{p}\left(2\frac{t}{T}\,\chi_{\left(0,\frac{T}{2}\right]}(t) + 2\frac{T-t}{T}\,\chi_{\left(\frac{T}{2},T\right]}(t)\right),$$

describing the case in which a structure is increasingly loaded until a maximum pressure is reached, and then linearly unloaded in order to recover the initial resting state. For the case

Table 7 Test case 1. Computational data related to POD-Galerkin-DEIM ROMs and Deep-HyROMnet, for $N = 8$

	POD-Galerkin-DEIM (m =29)	POD-Galerkin-DEIM (m =51)	Deep-HyROMnet
Computational speed-up	× 8	× 5	× **949**
Avg. CPU time	3 s	5 s	0.027 s
Time-avg. $L^2(\Omega_0)$-abs. err.	2.6×10^{-5}	3.9×10^{-6}	9.0×10^{-5}
Time-avg. $L^2(\Omega_0)$-rel. err.	1.1×10^{-2}	6.0×10^{-4}	8.1×10^{-3}

Significance values are in bold

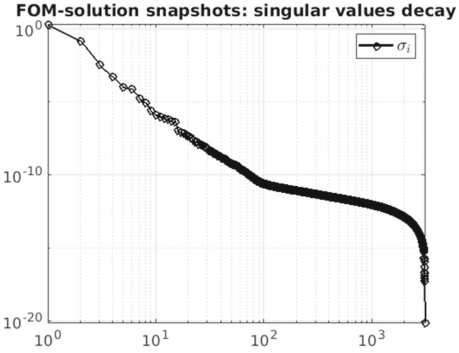


Fig. 10 Test case 2. Decay of the singular values of the FOM solution

at hand, we choose $t \in [0, 0.35]$ s and $\Delta t = 5 \times 10^{-3}$ s, resulting in a total number of 70 time steps. As parameter, we consider the external load parameter $\widetilde{p} \in [2, 12]$ Pa; the shear modulus G and the bulk modulus K are fixed to 10^4 Pa and 5×10^4 Pa, respectively. Let us consider a training set of $n_s = 50$ points generated from $\mathcal{P} = [2, 12]$ Pa through LHS and build the RB basis $\mathbf{V} \in \mathbb{R}^{N_h \times N}$ with $N = 4$, corresponding to $\varepsilon_{POD} = 10^{-4}$. The singular values of the solution snapshots matrix are reported in Fig. 10.

Given the Galerkin-ROM nonlinear data collected for $n'_s = 300$ sampled parameters, different DEIM residual bases $\mathbf{\Phi}_{\mathcal{R}}$ are computed using different tolerances

$$\varepsilon_{DEIM} \in \{10^{-3}, 5 \times 10^{-4}, 10^{-4}, 5 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6}, 10^{-6}\},$$

corresponding to $m = 14, 16, 22, 24, 29, 31, 37$ DEIM basis functions, respectively. Tolerances ε_{DEIM} larger than the values reported above were not sufficient to ensure convergence of the Newton method for all the considered parameters.

The relative error ϵ_{rel}, evaluated over a testing set of 50 parameters, is about 10^{-2} when using $m = 14$ residual basis functions, and can be further reduced of one order of magnitude when increasing the DEIM dimension to $m = 29$, albeit highly decreasing the CPU time ratio, as shown in Fig. 11.

Table 8 shows the comparison between the POD-Galerkin-DEIM (with $m = 14$ and $m = 29$) and the Deep-HyROMnet hyper-reduced models on a testing set of 50 parameter instances. As observed in the previous test case, Deep-HyROMnet is able to achieve good results in terms of accuracy, comparable with the fastest POD-Galerkin-DEIM ROM ($m = 14$), at a greatly reduced cost. Also in this case, the speed-up achieved by our Deep-HyROMnet is of order $\mathcal{O}(10^3)$ with respect to the FOM, since less than 0.04 s are needed to

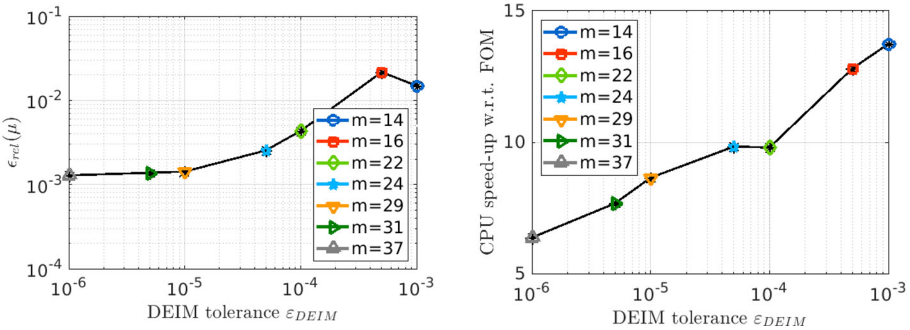


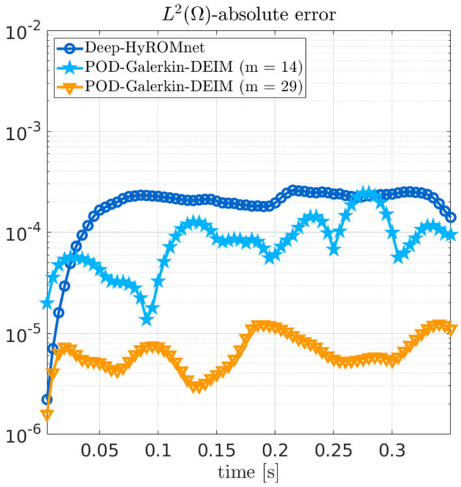
Fig. 11 Test case 2. Average over 50 testing parameters of relative error ϵ_{rel} (left) and average speed-up (right) of POD-Galerkin-DEIM with $N = 4$

Table 8 Test case 2. Computational data related to POD-Galerkin-DEIM ROMs and Deep-HyROMnet, for $N = 4$

	POD-Galerkin-DEIM ($m = 14$)	POD-Galerkin-DEIM ($m = 29$)	Deep-HyROMnet
Computational speed-up	× 14	× 9	× 1153
Avg. CPU time	3 s	5 s	0.035 s
Time-avg. $L^2(\Omega_0)$-abs. err.	9.0×10^{-5}	6.8×10^{-6}	2.0×10^{-4}
Time-avg. $L^2(\Omega_0)$-rel. err.	1.5×10^{-2}	1.4×10^{-3}	1.7×10^{-2}

Significance values are in bold

Fig. 12 Test case 2. Evolution in time of the average $L^2(\Omega_0)$-absolute error for $N = 4$ computed using POD-Galerkin-DEIM and Deep-HyROMnet



compute the reduced solution for each new instance of the parameter, against a time of about 40 s required by the FOM, and of (at least) 3 s required by POD-Galerkin-DEIM ROMs.

The evolution in time of the average $L^2(\Omega_0)$-absolute error for the POD-Galerkin-DEIM and the Deep-HyROMnet models is shown in Fig. 12. The accuracy obtained using Deep-HyROMnet, although slightly lower than the ones achieved using a DEIM-based approximation, is satisfying in all the considered scenarios. Figure 13 shows the FOM and

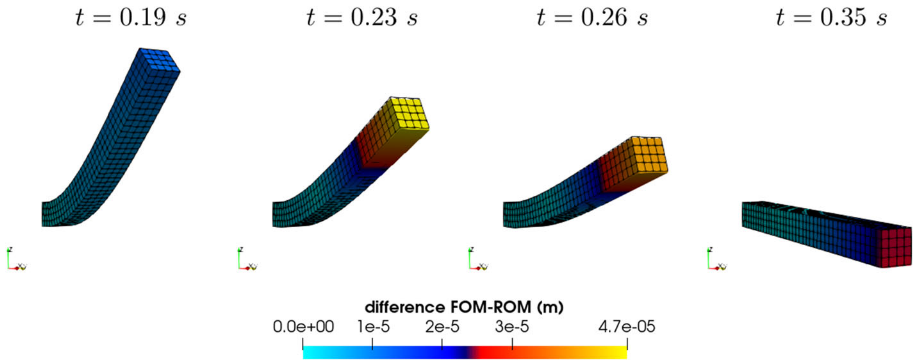


Fig. 13 Test case 2. FOM (wireframe) and Deep-HyROMnet (colored) solutions computed at different times for $\boldsymbol{\mu} = [10.7375\ \text{Pa}]$ (Color figure online)

Table 9 Test case 2. Offline time required for the construction of the ROMs and the equivalence in terms of FOM simulations

	CPU time × n_s (or n'_s)
FOM	40 s × 50
POD-Galerkin ROM	22 s × 300
	GPU time × #epochs
DNN for $\mathbf{R}_N$	6.2 s × 6367
DNN for $\mathbf{J}_N$	6.5 s × 4899
Offline time	13 h 21 min
Break even point	1202

the Deep-HyROMnet displacements at different time instances obtained for a given testing parameter.

We finally report in Table 9 the offline CPU and GPU times required for the test case at hand.

4.2.3 Test Case 3: Step Function for the Pressure Load

As last test case for the beam geometry, we consider a pressure load acting on the bottom surface area for only a third of the whole simulation time, that is

$$\mathbf{g}(t; \boldsymbol{\mu}) = \widetilde{p}\ \chi_{\left(0, \frac{T}{3}\right]}(t),$$

such that the resulting deformation features oscillations. This case is of particular interest in nonlinear elastodynamics, since the inertial term cannot be neglected, as it has a crucial impact on the deformation of the body. For the case at hand, we choose $t \in [0, 0.27]$ s and a uniform time step $\Delta t = 3.6 \times 10^{-3}$ s, resulting in a total number of 75 time iterations. Concerning the input parameters, we vary the external load $\widetilde{p} \in [2, 12]$ Pa and consider $G = 10^4$ Pa and $K = 5 \times 10^4$ Pa fixed.

We build the POD basis $\mathbf{V} \in \mathbb{R}^{N_h \times N}$ from a training set of $n_s = 50$ FOM solutions using $\varepsilon_{POD} = 10^{-3}$, obtaining also in this case a reduced dimension of $N = 4$, and perform POD-Galerkin ROM simulations for a given set of $n'_s = 300$ parameter samples to collect the

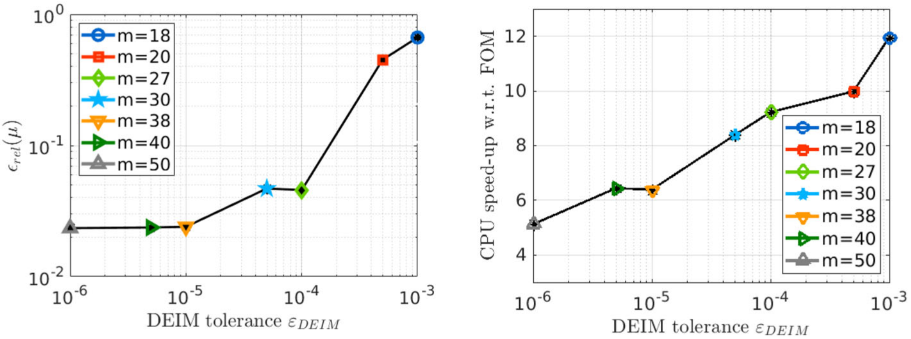


Fig. 14 Test case 3. Average over 50 testing parameters of relative error ϵ_{rel} (left) and average speed-up (right) of POD-Galerkin-DEIM with $N = 4$

Table 10 Test case 3. Computational data related to POD-Galerkin-DEIM ROMs and Deep-HyROMnet, for $N = 4$

	POD-Galerkin-DEIM ($m = 18$)	POD-Galerkin-DEIM ($m = 38$)	Deep-HyROMnet
Computational speed-up	× 12	× 6	× 1350
Avg. CPU time	4 s	8 s	0.038 s
Time-avg. $L^2(\Omega_0)$-abs. err.	2.4×10^{-3}	1.3×10^{-4}	4.8×10^{-4}
Time-avg. $L^2(\Omega_0)$-rel. err.	6.7×10^{-1}	2.4×10^{-2}	1.0×10^{-1}

Significance values are in bold

nonlinear terms data necessary for the construction of both POD-Galerkin-DEIM and Deep-HyROMnet models. The DEIM basis $\mathbf{\Phi}_{\mathcal{R}}$ for the approximation of the residual is computed using as tolerances

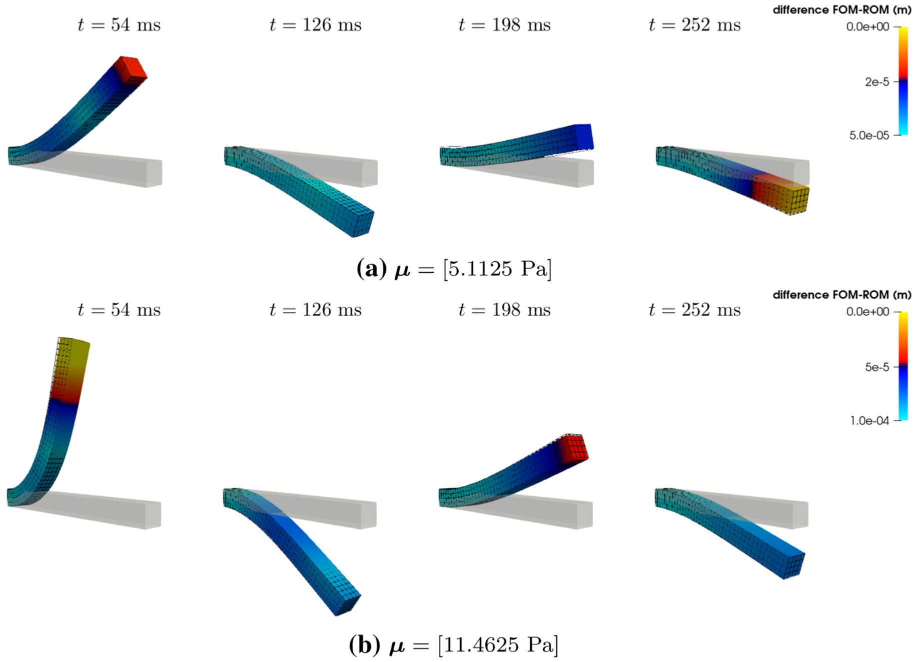
$$\varepsilon_{DEIM} \in \{10^{-3}, 5 \times 10^{-4}, 10^{-4}, 5 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6}, 10^{-6}\},$$

where $\varepsilon_{DEIM} = 10^{-3}$ is the larger tolerance that ensures the convergence of the reduced Newton algorithm for all testing parameters. The corresponding number of bases for $\mathbf{R}$ is $m = 18, 20, 27, 30, 38, 40, 50$, respectively. The results regarding the average relative error ϵ_{rel} and the computational speed-up, evaluated over 50 instances of the parameter, are shown in Fig. 14.

Like for the previous test cases, we compare POD-Galerkin-DEIM and Deep-HyROMnet ROMs, with respect to the displacement error and the CPU time ratio.

As reported in Table 10, DNNs outperform DEIM substantially in terms of efficiency also in this case when handling the nonlinear terms. Indeed, Deep-HyROMnet yields a ROM that is more than 1000 times faster than the FOM (this latter requiring 51 s in average to be solved), still providing satisfactory results in terms of accuracy.

In Fig. 15 the Deep-HyROMnet solution at different time instants for two different values of the parameter is shown, highlighting that our hyper-ROM is able to correctly capture the nonlinear behavior of the continuum body also when the inertial term cannot be neglected. Table 11 reports the offline times required for the construction of the ROMs in the present case.



(a) $\boldsymbol{\mu} = [5.1125 \text{ Pa}]$

(b) $\boldsymbol{\mu} = [11.4625 \text{ Pa}]$

Fig. 15 Test case 3. FOM (wireframe) and Deep-HyROMnet (colored) solutions computed at different times (Color figure online)

Table 11 Test case 3. Offline time required for the construction of the ROMs and the equivalence in terms of FOM simulations

	CPU time × n_s (or n_s')
FOM	51 s × 50
POD-Galerkin ROM	29 s × 300
	GPU time × #epochs
DNN for $\mathbf{R}_N$	8.0 s×4993
DNN for $\mathbf{J}_N$	8.2 s×3441
Offline time	14 h 13 min
Break even point	1004

4.3 Passive Inflation and Active Contraction of an Idealized Left Ventricle

The second problem we are interested in is the inflation and contraction of a prolate spheroid geometry representing an idealized left ventricle (see Fig. 16) where the boundaries Γ_0^R, Γ_0^N and Γ_0^D represent the epicardium, the endocardium and the base of a left ventricle, respectively, the latter being the artificial boundary resulting from truncation of the heart below the valves in a short axis plane. We consider transversely isotropic material properties for the myocardial tissue, adopting a nearly-incompressible formulation of the constitutive law proposed in [31], whose strain-energy density function is given by $\mathcal{W}(\mathbf{F}) = \frac{C}{2}(e^{Q(\mathbf{F})} - 1)$, with the following form for Q to describe three-dimensional transverse isotropy with respect

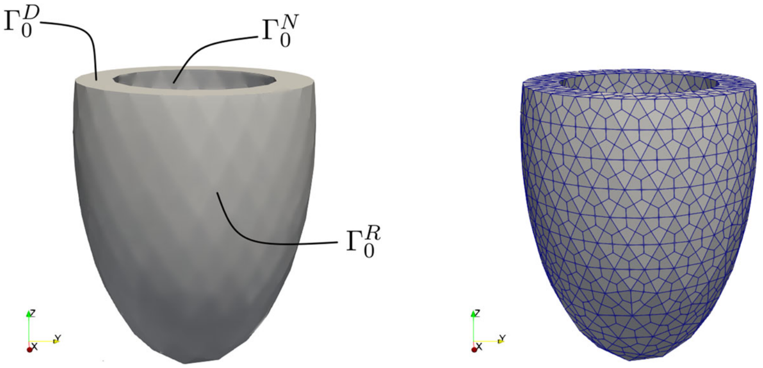


Fig. 16 Passive inflation and active contraction of an idealized left ventricle. Idealized truncated ellipsoid geometry (left) and computational grid (right)

to the fiber coordinate system,

$$Q = b_f E_{ff}^2 + b_s E_{ss}^2 + b_n E_{nn}^2 + b_{fs}(E_{fs}^2 + E_{sf}^2) + b_{fn}(E_{fn}^2 + E_{nf}^2) + b_{sn}(E_{sn}^2 + E_{ns}^2).$$

here E_{ij}, $i, j \in \{f, s, n\}$, are the components of the Green-Lagrange strain tensor $\mathbf{E} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I})$, the material constant $C > 0$ scales the stresses and the coefficients b_f, b_s, b_n are related to the material stiffness in the fiber, sheet and transverse directions, respectively. This leads to a (passive) first Piola-Kirchhoff stress tensor characterized by exponential nonlinearity. In order to enforce the incompressibility constraint, we consider an additional term $\mathcal{W}_{vol}(J)$ in the definition of the strain-energy density function, which must grow as the deformation deviates from being isochoric. A common choice for $\mathcal{W}_{vol}$ is a convex function with null slope in $J = 1$, e.g.,

$$\mathcal{W}_{vol}(J) = \frac{K}{4}((J-1)^2 + \ln^2(J)),$$

where the penalization factor is the bulk modulus $K > 0$. Furthermore, to reproduce the typical twisting motion of the ventricular systole, we need to take into account a varying fiber distribution and contractile forces. The fiber direction is computed using the rule-based method proposed in [51], which depends on parameter angles $\boldsymbol{\alpha}^{epi}$ and $\boldsymbol{\alpha}^{endo}$. Active contraction is modeled through the active stress approach [1], so that we add to the passive first Piola-Kirchoff stress tensor a time-dependent active tension, which is assumed to act only in the fiber direction

$$\mathbf{P} = \left(\frac{\partial \mathcal{W}(\mathbf{F})}{\partial \mathbf{F}} + \frac{\partial \mathcal{W}_{vol}(J)}{\partial \mathbf{F}}\right) + T_a(t)(\mathbf{F}\mathbf{f}_0 \otimes \mathbf{f}_0),$$

where $\mathbf{f}_0 \in \mathbb{R}^3$ denotes the reference unit vector in the fiber direction and T_a is a parametrized function that surrogates the active generation forces. In our case, since we are modeling only the systolic contraction, we define

$$T_a(t) = \widetilde{T}_a\, t/T, \quad t \in (0, T),$$

with $\widetilde{T}_a > 0$. To model blood pressure inside the chamber we assume a linearly increasing external load

$$\mathbf{g}(t; \boldsymbol{\mu}) = \hat{p}\, t/T, \quad t \in (0, T).$$

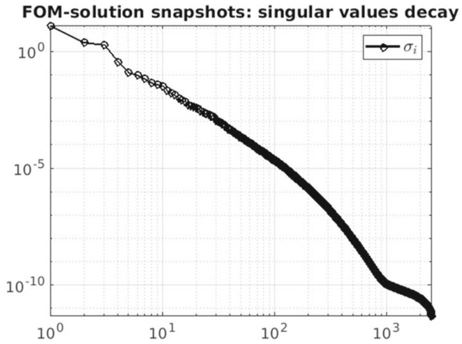


Fig. 17 Passive inflation and active contraction of an idealized left ventricle. Decay of the singular values of the FOM solution snapshots matrix

Since we want to assess the performance of Deep-HyROMnet in enhancing the myocardium contraction problem, we consider as unknown parameters those related to the active components of the strain-energy density function:

- The maximum value of the active tension $\widetilde{T}_a \in [49.5 \times 10^3, 70.5 \times 10^3]$ Pa, and
- The fiber angles $\boldsymbol{\alpha}^{epi} \in [-105.5, -74.5]°$ and $\boldsymbol{\alpha}^{endo} \in [74.5, 105.5]°$.

All other parameters are fixed to the reference values taken from [38], namely $b_f = 8$, $b_s = b_n = b_{sn} = 2$, $b_{fs} = b_{fn} = 4$, $C = 2 \times 10^3$ Pa, $K = 50 \times 10^3$ Pa and $\widetilde{p} = 15 \times 10^3$. Regarding time discretization, we choose $t \in [0, 0.25]$ s and a uniform time step $\Delta t = 5 \times 10^{-3}$ s, resulting in a total number of 50 time iterations. The FOM is built on a hexahedral mesh with 4804 elements and 6455 vertices, depicted in Fig. 16, corresponding to a high-fidelity dimension $N_h = 19365$, since $\mathbb{Q}_1$-FE (that is, linear FE on a hexahedral mesh) are used. In this case, the FOM requires almost 360 s to compute the solution dynamics for each parameter instance.

Given $n_s = 50$ points obtained by sampling the parameter space $\mathcal{P}$, we construct the corresponding solution snapshots matrix $\mathbf{S}_u$ and compute the POD basis $\mathbf{V} \in \mathbb{R}^{N_h \times N}$ using the POD method with tolerance

$$\varepsilon_{POD} \in \{10^{-3}, 5 \times 10^{-4}, 10^{-4}, 5 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6}, 10^{-6}\}.$$

From Fig. 17, we observe a slower decay of the singular values of $\mathbf{S}_u$ with respect to the structural problems of Sect. 4.2. In fact, we obtain larger reduced basis dimensions $N = 16$, 22, 39, 50, 87, 109 and 178, respectively.

The error and the CPU speed-ups averaged over a testing set of 20 parameters are both shown in Fig. 18, as functions of the POD tolerance ε_{POD}.

Also in this case, the speed-up achieved by the ROM is negligible, since at each Newton iteration without hyper-reduction the ROM still depends on the high-fidelity dimension N_h. For what concerns the approximation error, we observe a reduction of almost two orders of magnitude when going from $N = 16$ to $N = 178$. The fact that greater POD dimension with respect to the previous test cases are now obtained, allows us to assess the performances of the Deep-HyROMnet approach both on small and large values of N.

Given the reduced basis $\mathbf{V} \in \mathbb{R}^{N_h \times N}$ with $N = 16$, we construct the POD-Galerkin-DEIM approximation by considering $n'_s = 200$ parameter samples. Fig. 19 shows the decay of the singular values of $\mathbf{S}_R$, that is, the snapshots matrix of the residual vectors $\mathbf{R}(\mathbf{V}\mathbf{u}_N^{n,(k)}(\boldsymbol{\mu}_{\ell'}), t^n; \boldsymbol{\mu}_{\ell'})$. We observe that the reported curve decreases very slowly, so that we expect that a large number of basis functions is required to correctly approximate the nonlinear operators.

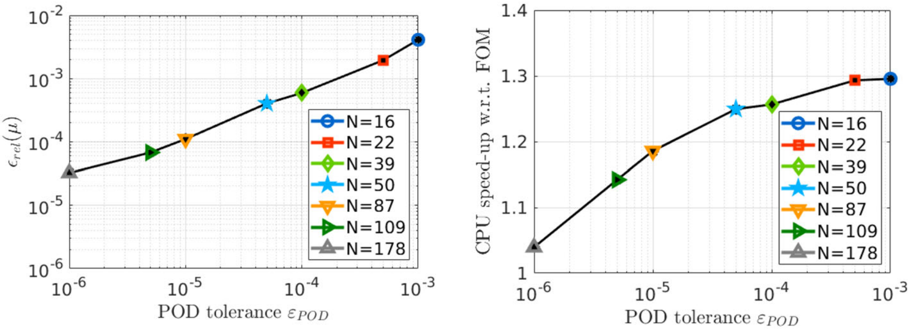
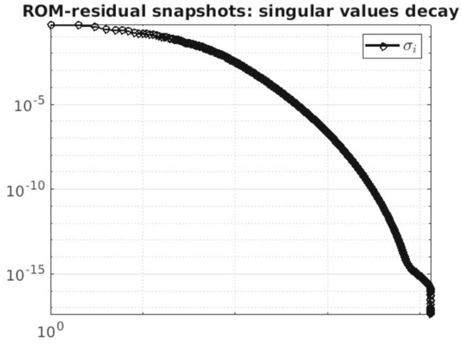


Fig. 18 Passive inflation and active contraction of an idealized left ventricle. Average over 20 testing parameters of relative error ϵ_{rel} (left) and average speed-up (right) of ROM without hyper-reduction

Fig. 19 Passive inflation and active contraction of an idealized left ventricle. Decay of the singular values of the ROM residual snapshots matrix



In fact, by computing $\mathbf{\Phi}_{\mathcal{R}} \in \mathbb{R}^{N_h \times m}$ using the following DEIM tolerances,

$$\varepsilon_{DEIM} \in \{5 \times 10^{-4}, 10^{-4}, 5 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6}, 10^{-6}\},$$

we obtain $m = 303$, 456, 543, 776, 902 and 1233, respectively. Higher values of ε_{DEIM} (related to smaller dimensions m) were not sufficient to guarantee the convergence of the reduced Newton problem for all testing parameters. The average relative error over a set of 20 parameters and the computational speed-up are both reported in Fig. 20. In particular, we observe that the relative error is between 4×10^{-3} and 8×10^{-3}, as we could expect from the projection error reported in Fig. 18, that is, POD-Galerkin-DEIM ROMs are able to achieve the same accuracy of the ROM without hyper-reduction.

The data reported in Table 12 leads to the same conclusions regarding the computational bottleneck of DEIM technique as those reported in Table 3. In fact, assembling the residual on the reduced mesh requires around 85% of the online CPU time, thus undermining the hyper-ROM efficiency.

Finally, Table 13 reports the computational data of POD-Galerkin-DEIM ROMs (for a number of magic points equals to $m = 303$ and $m = 543$) and Deep-HyROMnet, clearly showing that the latter outperforms the classical reduction strategy regarding the computational speed-up. In fact, Deep-HyROMnet is able to approximate the solution dynamics in 0.1 s, that is even faster than real-time given the timescale of the phenomenon at hand, while a POD-Galerkin-DEIM ROM requires 1 min in average, where the final simulation time T is set equal to 0.25 s. The offline CPU and GPU times are reported in Table 14. Although the Deep-HyROMnet error is slightly larger than the one obtained with a DEIM-based hyper-

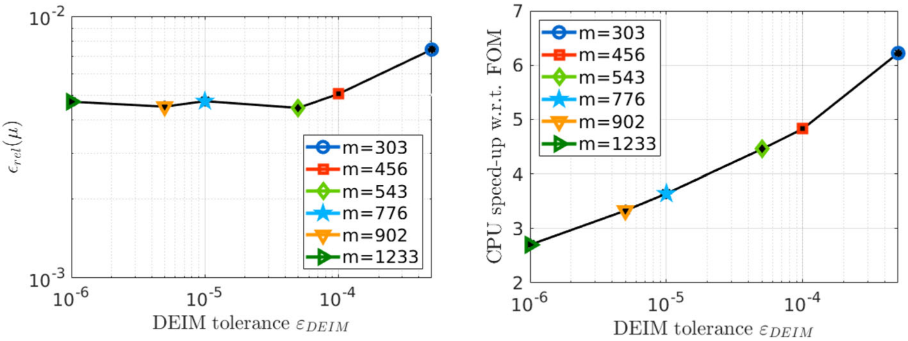


Fig. 20 Passive inflation and active contraction of an idealized left ventricle. Average over 20 testing parameters of relative error ϵ_{rel} (left) and average speed-up (right) of POD-Galerkin-DEIM ROMs with $N = 16$

Table 12 Passive inflation and active contraction of an idealized left ventricle

DEIM tolerance ε_{DEIM}	5×10^{-4}	5×10^{-5}	5×10^{-6}
DEIM interpolation dofs m	303	543	902
Reduced mesh elements (total: 4804)	914	1345	1855
Online CPU time	58 s	81 s	110 s
System construction [$*$]	89%	93%	94%
System solution	0.01%	0.01%	0.01%
[$*$] System construction for each Newton iteration	0.4 s	0.6 s	0.9 s
Residual assembling	94%	94%	94%
Jacobian computing through AD	0.24%	0.24%	0.26%
Computational speed-up	× 6.2	× 4.5	× 3.3
Time-averaged $L^2(\Omega_0)$-absolute error	1×10^{-3}	6×10^{-4}	6×10^{-4}
Time-averaged $L^2(\Omega_0)$-relative error	7×10^{-3}	5×10^{-3}	5×10^{-3}

Computational data related to POD-Galerkin-DEIM with $N = 16$ and different values of m

Table 13 Passive inflation and active contraction of an idealized left ventricle

	POD-Galerkin-DEIM ($m = 303$)	POD-Galerkin-DEIM ($m = 543$)	Deep-HyROMnet
Computational speed-up	× 6	× 5	× 3554
Avg. CPU time	58 s	75 s	0.1 s
Time-avg. $L^2(\Omega_0)$-abs. err.	1.3×10^{-3}	6.6×10^{-4}	1.5×10^{-2}
Time-avg. $L^2(\Omega_0)$-rel. err.	7.5×10^{-3}	5.0×10^{-3}	6.4×10^{-2}

Significance values are in bold
Computational data related to POD-Galerkin-DEIM ROMs and Deep-HyROMnet, for $N = 16$

ROM (see Fig. 21), the results are satisfactory in terms of accuracy. In Fig. 22 the FOM and the Deep-HyROMnet displacements at time $T = 0.25$ s are reported for three different values of the parameters, together with the error between the high-fidelity and the reduced solutions.

Table 15 reports the accuracy of the prediction on the interpolation functions, showing that the error on the reduced Jacobian matrix $\mathbf{J}_N$ is of the same order of the corresponding

Table 14 Passive inflation and active contraction of an idealized left ventricle. Offline time required for the construction of the ROMs and the equivalence in terms of FOM simulations

	CPU time × n_s (or n_s')
FOM	360 s × 50
POD-Galerkin ROM	240 s × 200
	GPU time × #epochs
DNN for $\mathbf{R}_N$	4.0 s × 10080
DNN for $\mathbf{J}_N$	8.0 s × 8615
Offline time	2 d 40 min
Break even point	487

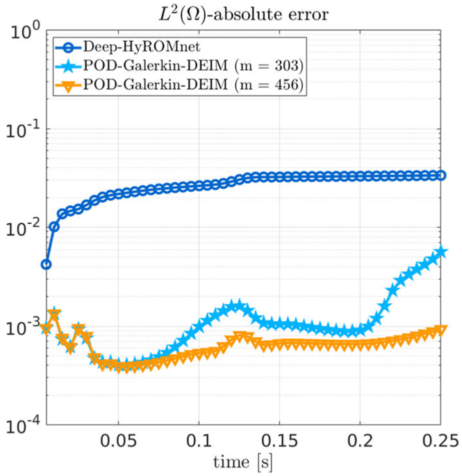


Fig. 21 Passive inflation and active contraction of an idealized left ventricle. Evolution in time of the average $L^2(\Omega_0)$-absolute error computed using POD-Galerkin-DEIM ROMs and Deep-HyROMnet, for $N = 16$

reduced residual vector $\mathbf{R}_N$ and one order of magnitude higher than in test case 4.2. This fact might be due to several issues, such as the larger dimension of the input, i.e., $N^2 = 256$ instead of $N^2 = 16$, the smaller size of the training set, and the overall increased complexity of the problem, either in the constitutive law and the geometry.

To conclude, we assess the performances of Deep-HyROMnet on a problem involving a higher FOM dimension. We still consider the test case described in this Section, however using a finer hexahedral mesh with 9964 elements and 13025 vertices, thus obtaining $N_h = 39075$ as FOM dimension. In this case, about 13 minutes are required to compute the high-fidelity solution. On the other hand, a reduced basis of dimension $N = 16$ is computed for $\varepsilon_{POD} = 10^{-3}$; the computational data, averaged over a testing set of 20 parameter samples, are reported in Table 16. The online CPU time required by Deep-HyROMnet doubles as we double N_h, despite the same POD dimension $N = 16$ has been selected, and the same level of accuracy is obtained. This mild dependence of our hyper-ROM on the FOM dimension N_h is due to the reconstruction of the reduced solutions $\mathbf{V}\mathbf{u}_N^n(\boldsymbol{\mu})$, for $n = 1, \dots, N_t$, whereas the same computational time is required for the online assembling and solution of the reduced Newton systems. Nonetheless, the overall computational speed-up of Deep-HyROMnet increases as N_h grows, while the number N of reduced basis function remains small, so that reduced solutions can be computed extremely fast.

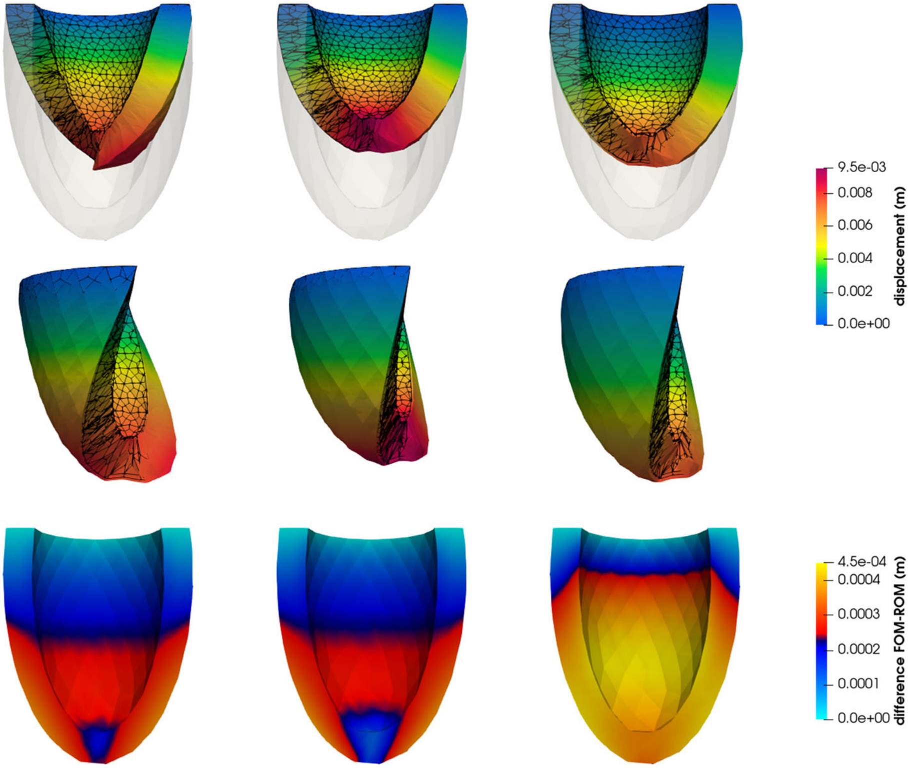


Fig. 22 Passive inflation and active contraction of an idealized left ventricle. FOM (wireframe) and Deep-HyROMnet (colored) displacements (frontal view on top, lateral view in the middle) and corresponding difference (bottom) at time $T = 0.25$ s for $\boldsymbol{\mu} = [61942.5 \text{ Pa}, -77.5225°, 87.9075°]$ (left), $\boldsymbol{\mu} = [59737.5 \text{ Pa}, -102.3225°, 91.1625°]$ (center) and $\boldsymbol{\mu} = [50497.5 \text{ Pa}, -100.9275°, 80.0025°]$ (right) (Color figure online)

Table 15 Passive inflation and active contraction of an idealized left ventricle

	On training set	On test set
$\epsilon_{rel,\mathbf{R}_N}$	0.00342	0.00633
$\epsilon_{rel,\mathbf{J}_N}$	0.00199	0.00331

Accuracy of the DNNs predictions on training and test sets

5 Conclusions

In this work we have addressed the solution to parametrized, nonlinear, time-dependent PDEs arising in elastodynamics, by means of a new projection-based ROM, developed to accurately capture the state solution dynamics at a reduced computational cost with respect to high-fidelity FOMs. We focused on Galerkin-RB methods, characterized by a projection of the differential problem onto a low-dimensional subspace built, e.g., by performing POD on a set of FOM solutions, and by the splitting of the reduction procedure into a costly offline phase and an inexpensive online phase. Numerical experiments showed that, despite their highly nonlinear nature, elastodynamics problems can be reduced by exploiting projection-based strategies in an effective way, with POD-Galerkin ROMs achieving very good accuracy even in presence of a handful of basis functions. However, when dealing with nonlinear problems,

Table 16 Passive inflation and active contraction of an idealized left ventricle

	Deep-HyROMnet	
N_h	19, 365	39, 075
FOM time	5 min 54 s	13 min 01 s
N	16	
Speed-up	× 3554	× 3886
Avg. CPU time	0.1 s	0.2 s
Time-averaged $L^2(\Omega_0)$-abs. err.	1.5×10^{-2}	2.7×10^{-2}
Time-averaged $L^2(\Omega_0)$-rel. err.	6.4×10^{-2}	8.3×10^{-2}

Significance of values are in bold

Computational data related to Deep-HyROMnet for $N_h = 19,365$ and $N_h = 39,075$

a further level of approximation is required to make the online stage independent of the high-fidelity dimension.

Hyper-reduction techniques, such as DEIM, are necessary to efficiently handle the nonlinear operators. However, a serious issue is represented by the assembling (albeit onto a reduced mesh) of the approximated nonlinear operators in this framework. This observation suggested the idea of relying on surrogate models to perform operator approximation, overcoming the need to assemble the nonlinear terms onto the computational mesh. Pursuing this strategy, we have proposed a new projection-based, deep learning-based ROM, *Deep-HyROMnet*, which combines the Galerkin-RB approach with DNNs to assemble the reduced Newton system in an efficient way, thus avoiding the computational burden entailed by classical hyper-reduction strategies. This approach allows to rely on physics-based ROMs retaining the underlying structure of the physical model, as DNNs are employed only for the approximation of the reduced nonlinear operators. Regarding the offline cost of this hybrid reduction strategy, we point out that:

- FOM solutions are only required to build the POD-Galerkin ROM;
- A small number N of reduced basis functions is sufficient to accurately approximate the high-fidelity solution manifold, so that the arising reduced nonlinear systems can be solved efficiently;
- Since data on nonlinear operators are collected during Newton iterations at each time step, a smaller number of ROM simulations—compared to purely data-driven approaches—is sufficient for training the DNNs;
- Since training data are low-dimensional, we can avoid the overwhelming training times and costs that would be required by DNNs if FOM arrays were used.

The Deep-HyROMnet approach has been successfully applied on several test cases in nonlinear solid mechanics, showing remarkable improvements in terms of online CPU time with respect to POD-Galerkin-DEIM ROMs. Our goal in future works is to apply the developed strategy to other classes of nonlinear problems for which traditional hyper-reduction techniques represent a computational bottleneck.

Funding Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 740132, iHEART - An Integrated Heart Model for the simulation of the cardiac function, P.I. Prof. A. Quarteroni) and the Fondazione Cariplo (grant agreement no. 2019-4608, P.I. Prof. A. Manzoni).

Data Availability The code used in this work, as well as the training and testing datasets, will be made available upon request to the authors.

Declarations

Conflict of interest The authors have not disclosed any competing interests.

References

1. Ambrosi, D., Pezzuto, S.: Active stress versus active strain in mechanobiology: constitutive issues. J. Elast. **107**(2), 199–212 (2012)
2. Amsallem, D., Zahr, M., Farhat, C.: Nonlinear model order reduction based on local reduced-order bases. Int. J. Numer. Methods Eng. **92**(10), 891–916 (2012)
3. Arndt, D., Bangerth, W., Blais, B., Clevenger, T., Fehling, M., Grayver, A., Heister, T., Heltai, L., Kronbichler, M., Maier, M., Munch, P., Pelteret, J., Rastak, R., Thomas, I., Turcksin, B., Wang, Z., Wells, D.: The deal.II library, version 9.2. J. Numer. Math. **28**(3), 131–146 (2020). https://doi.org/10.1515/jnma-2020-0043
4. Astrid, P., Weiland, S., Willcox, K., Backx, T.: Missing point estimation in models described by proper orthogonal decomposition. IEEE Trans. Autom. Control **53**(10), 2237–2251 (2008)
5. Bai, Z., Peng, L.: Non-intrusive nonlinear model reduction via machine learning approximations to low-dimensional operators. Adv. Model. Simul. Eng. Sci. **8**(1), 1–24 (2021)
6. Barrault, M., Maday, Y., Nguyen, N., Patera, A.: An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. Compt. Rendus Math. **339**(9), 667–672 (2004)
7. Benner, P., Goyal, P., Kramer, B., Peherstorfer, B., Willcox, K.: Operator inference for non-intrusive model reduction of systems with non-polynomial nonlinear terms. Comput. Methods Appl. Mech. Eng. **372**, 113433 (2020)
8. Benner, P., Gugercin, S., Willcox, K.: A survey of projection-based model reduction methods for parametric dynamical systems. SIAM Rev. **57**(4), 483–531 (2015)
9. Benner, P., Ohlberger, M., Patera, A., Rozza, G., Urban (Eds.), K.: Model Reduction of Parametrized Systems. Springer (2017)
10. Bhattacharya, K., Hosseini, B., Kovachki, N., Stuart, A.: Model reduction and neural networks for parametric PDEs. SMAI J. Comput. Math. **7**, 121–157 (2021)
11. Bonomi, D., Manzoni, A., Quarteroni, A.: A matrix DEIM technique for model reduction of nonlinear parametrized problems in cardiac mechanics. Comput. Methods Appl. Mech. Eng. **324**, 300–326 (2017)
12. Broyden, C.: A class of methods for solving nonlinear simultaneous equations. Math. Comput. **19**(92), 577–593 (1965)
13. Carlberg, K., Bou-Mosleh, C., Farhat, C.: Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations. Int. J. Numer. Methods Eng. **86**(2), 155–181 (2011)
14. Chatterjee, A.: An introduction to the proper orthogonal decomposition. Curr. Sci. **78**(7), 808–817 (2000)
15. Chaturantabut, S., Sorensen, D.: Nonlinear model reduction via discrete empirical interpolation. SIAM J. Sci. Comput. **32**(5), 2737–2764 (2010)
16. Chen, T., Chen, H.: Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Trans. Neural Netw. **6**(4), 911–917 (1995)
17. Cicci, L., Fresca, S., Pagani, S., Manzoni, A., Quarteroni, A.: Projection-based reduced order models for parameterized nonlinear time-dependent problems arising in cardiac mechanics. Math. Eng. **5**(2), 1–38 (2023). https://doi.org/10.3934/mine.2023026

18. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 (2015)
19. Drohmann, M., Haasdonk, B., Ohlberger, M.: Reduced basis approximation for nonlinear parametrized evolution equations based on empirical operator interpolation. SIAM J. Sci. Comput. **34**(2), A937–A969 (2012)
20. Farhat, C., Chapman, T., Avery, P.: Structure-preserving, stability, and accuracy properties of the energy-conserving sampling and weighting method for the hyper reduction of nonlinear finite element dynamic models. Int. J. Numer. Meth. Eng. **102**(5), 1077–1110 (2015)
21. Farhat, C., Grimberg, S., Manzoni, A., Quarteroni, A.: Computational bottlenecks for PROMs: precomputation and hyperreduction. In: Benner, P., Grivet-Talocia, S., Quarteroni, A., Rozza, G., Schilders, W., Silveira, L. (eds.) Model Order Reduction. Snapshot-Based Methods and Algorithms, vol. 2, pp. 181–244. De Gruyter, Berlin (2020)
22. Franco, N., Manzoni, A., Zunino, P.: A deep learning approach to reduced order modelling of parameter dependent partial differential equations. arXiv preprint arXiv:2103.06183 (2021)
23. Fresca, S., Dede', L., Manzoni, A.: A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. J. Sci. Comput. **87**(2), 1–36 (2021)
24. Fresca, S., Manzoni, A.: POD-DL-ROM: enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition. Comput. Methods Appl. Mech. Eng. **388**, 114181 (2022)
25. Fresca, S., Manzoni, A., Dede', L., Quarteroni, A.: Deep learning-based reduced order models in cardiac electrophysiology. PloS One **15**(10), e0239416 (2020)
26. Gao, H., Wang, J., Zahr, M.: Non-intrusive model reduction of large-scale, nonlinear dynamical systems using deep learning. Phys. D: Nonlinear Phenom. **412**, 132614 (2020)
27. Ghavamian, F., Tiso, P., Simone, A.: POD-DEIM model order reduction for strain-softening viscoplasticity. Comput. Methods Appl. Mech. Eng. **317**, 458–479 (2017)
28. Gobat, G., Opreni, A., Fresca, S., Manzoni, A., Frangi, A.: Reduced order modeling of nonlinear microstructures through proper orthogonal decomposition. Mech. Syst. Sign. Process. **171**, 108864 (2022). https://doi.org/10.1016/j.ymssp.2022.108864
29. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
30. Grepl, M., Maday, Y., Nguyen, N., Patera, A.: Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations. ESAIM: Math. Model. Numer. Anal. **41**(3), 575–605 (2007)
31. Guccione, J., Costa, K., McCulloch, A.: Finite element stress analysis of left ventricular mechanics in the beating dog heart. J. Biomech. **28**(10), 1167–1177 (1995)
32. Guo, M., Hesthaven, J.: Reduced order modeling for nonlinear structural analysis using gaussian process regression. Comput. Methods Appl. Mech. Eng. **341**, 807–826 (2018)
33. Guo, M., Hesthaven, J.S.: Data-driven reduced order modeling for time-dependent problems. Comput. Methods Appl. Mech. Eng. **345**, 75–99 (2019)
34. Halko, N., Martinsson, P., Tropp, J.: Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. **53**(2), 217–288 (2011)
35. Hesthaven, J., Rozza, G., Stamm, B.: Certified Reduced Basis Methods for Parametrized Partial Differential Equations. Springer (2016)
36. Hesthaven, J., Ubbiali, S.: Non-intrusive reduced order modeling of nonlinear problems using neural networks. J. Comput. Phys. **363**, 55–78 (2018)
37. Kim, Y., Choi, Y., Widemann, D., Zohdi, T.: Efficient nonlinear manifold reduced order model. arXiv preprint arXiv:2011.07727 (2020)
38. Land, S., Gurev, V., Arens, S., Augustin, C., Baron, L., Blake, R., Bradley, C., Castro, S., Crozier, A., Favino, M., Fastl, T., Fritz, T., Gao, H., Gizzi, A., Griffith, B., Hurtado, D., Krause, R., Luo, X., Nash, M., Pezzuto, S., Plank, G., Rossi, S., Ruprecht, D., Seemann, G., Smith, N., Sundnes, J., Rice, J., Trayanova, N., Wang, D., Wang, Z., Niederer, S.: Verification of cardiac mechanics software: benchmark problems and solutions for testing active and passive material behaviour. Proc. R. Soc. A: Math. Phys. Eng. Sci. **471**(2184), 20150641 (2015)
39. Lee, K., Carlberg, K.: Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. J. Comput. Phys. **404**, 108973 (2020)
40. Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.: Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nat. Mach. Intell. **3**(3), 218–229 (2021)
41. Manzoni, A., Bonomi, D., Quarteroni, A.: Reduced order modeling for cardiac electrophysiology and mechanics: New methodologies, challenges and perspectives. In: Boffi, D., Pavarino, L., Rozza, G., Scacchi, S., Vergara, C. (eds.) Mathematical and Numerical Modeling of the Cardiovascular System and Applications, SEMA SIMAI Springer Series, vol. 16, pp. 115–166. Springer, Cham (2018)

42. Negri, F., Manzoni, A., Amsallem, D.: Efficient model reduction of parametrized systems by matrix discrete empirical interpolation. J. Comput. Phys. **303**, 431–454 (2015)
43. Pagani, S., Manzoni, A., Quarteroni, A.: Numerical approximation of parametrized problems in cardiac electrophysiology by a local reduced basis method. Comput. Methods Appl. Mech. Eng. **340**, 530–558 (2018)
44. Peherstorfer, B., Butnaru, D., Willcox, K., Bungartz, H.: Localized discrete empirical interpolation method. SIAM J. Sci. Comput. **36**(1), A168–A192 (2014)
45. Peherstorfer, B., Willcox, K.: Data-driven operator inference for nonintrusive projection-based model reduction. Comput. Methods Appl. Mech. Eng. **306**, 196–215 (2016)
46. Pinkus, A.: n-Widths in Approximation Theory. Springer-Verlag, Berlin-Heidelberg (1985)
47. Qian, E., Kramer, B., Marques, A., Willcox, K.: Transform & learn: a data-driven approach to nonlinear model reduction. In: AIAA Aviation 2019 Forum, p. 3707 (2019)
48. Quarteroni, A.: Numerical Models for Differential Problems, 2nd edn. Springer (2013)
49. Quarteroni, A., Manzoni, A., Negri, F.: Reduced Basis Methods for Partial Differential Equations. Springer International Publishing, An Introduction (2016)
50. Radermacher, A., Reese, S.: POD-based model reduction with empirical interpolation applied to nonlinear elasticity. Int. J. Numer. Methods Eng. **107**(6), 477–495 (2016)
51. Rossi, S., Lassila, T., Ruiz-Baier, R., Sequeira, A., Quarteroni, A.: Thermodynamically consistent orthotropic activation model capturing ventricular systolic wall thickening in cardiac electromechanics. Eur. J. Mech.-A/Solids **48**, 129–142 (2014)
52. Swischuk, R., Mainini, L., Peherstorfer, B., Willcox, K.: Projection-based model reduction: formulations for physics-based machine learning. Comput. Fluids **179**, 704–717 (2019)
53. Tiso, P., Dedden, R., Rixen, D.: A modified discrete empirical interpolation method for reducing nonlinear structural finite element models. In: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 55973, p. V07BT10A043. American Society of Mechanical Engineers (2013)
54. Tiso, P., Rixen, D.: Discrete empirical interpolation method for finite element structural dynamics. In: Topics in Nonlinear Dynamics, Volume 1, pp. 203–212. Springer (2013)
55. Vlachas, K., Tatsis, K., Agathos, K., Brink, A., Chatzi, E.: A local basis approximation approach for nonlinear parametric model order reduction. J. Sound Vib. **502**, 116055 (2021)
56. Wang, S., Wang, H., Perdikaris, P.: Learning the solution operator of parametric partial differential equations with physics-informed deeponets. Sci. Adv. **7**(40), eabi8605 (2021)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.