A Discrete Physics-Informed Training for Projection-Based Reduced Order Models with Neural Networks

N. Sibuet , S. Ares de Parga * , J.R. Bravo , R. Rossi

Posted Date: 8 April 2025

doi: 10.20944/preprints202504.0645.v1

Keywords: reduced order model (ROM); Physics Informed Neural Networks (PINNs); artificial neural network (ANN); Projection-Based Model Reduction; proper orthogonal decomposition (POD)

 doi:10.20944/preprints202504.0645.v1

Article

A Discrete Physics-Informed Training for Projection-Based Reduced Order Models with Neural Networks

N. Sibuet [1], S. Ares de Parga [1,2,*], J.R. Bravo [1,2] and R. Rossi [1,2]

[1] Universitat Politècnica de Catalunya, Department of Civil and Environmental Engineering (DECA), Barcelona, Spain
[2] Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), Barcelona, Spain
* Correspondence: sebastian.ares@upc.edu

Abstract: This paper presents a physics-informed training framework for projection-based Reduced Order Models (ROMs). We extend the PROM-ANN architecture [1] by complementing snapshot-based training with a FEM-based, discrete physics-informed residual loss, bridging the gap between traditional projection-based ROMs and physics-informed neural networks (PINNs). Unlike conventional PINNs that rely on analytical PDEs, our approach leverages FEM residuals to guide the learning of the ROM approximation manifold. Key contributions include: (1) a parameter-agnostic, discrete residual loss applicable to non-linear problems, (2) an architectural modification to PROM-ANN improving accuracy for fast-decaying singular values, and (3) an empirical study on the proposed physics informed training process for ROMs. The method is demonstrated on a non-linear hyperelasticity problem, simulating a rubber cantilever under multi-axial loads. The main accomplishment in regards to the proposed residual-based loss is its applicability on non-linear problems by interfacing with FEM software while maintaining reasonable training times. The modified PROM-ANN outperforms POD by orders of magnitude in snapshot reconstruction accuracy, while the original formulation is not able to learn a proper mapping for this use-case. Finally, the application of physics informed training in ANN-PROM modestly narrows the gap between data reconstruction and ROM accuracy, however it highlights the untapped potential of the proposed residual-driven optimization for future ROM development. This work underscores the critical role of FEM residuals in ROM construction and calls for further exploration on architectures beyond PROM-ANN.

Keywords: reduced order model (ROM); Physics Informed Neural Networks (PINNs); artificial neural network (ANN); Projection-Based Model Reduction; proper orthogonal decomposition (POD)

MSC: 65M60; 68T07

1. Introduction

In recent years, the development of increasingly sophisticated high-fidelity models (HFMs) is crucial for simulating complex physical phenomena. One of the main computational challenges in HFMs is the need for very fine spatio-temporal resolutions, which results in extremely high-dimensional problems that can take weeks to solve, even on numerous parallelized computing cores. Therefore, it is vital to create methods that significantly reduce the computational time and resources required to enable the application of these models in time-sensitive scenarios, including real-time control systems, simulation-driven optimization and digital twins [2,3].

One of the most extended approaches to dimensionality reduction is the use of Projection-Based Reduced-Order Models (ROMs) [4], which combine data-driven methods with the physics-based solving process of HFMs. These methods leverage pre-computed results from the HFM to define a low-dimensional latent space that captures the essential dynamics of the system. A common technique for generating this latent space is Proper Orthogonal Decomposition (POD) [5,6], which employs

 doi:10.20944/preprints202504.0645.v1

truncated singular value decomposition (SVD) in sampled data to produce an orthonormal basis that spans the latent space and also serves as the optimal linear mapping from the full-dimensional to latent space and back [7]. The process of collecting samples within a defined parametric space and applying POD constitutes the offline stage—an upfront computational investment. This enables the online stage, where new solutions within the parametric space can be computed at a fraction of the cost of running the HFM. During the online stage, solutions are found directly in the latent space by projecting the full-dimensional optimization problem onto another reduced space of the same dimensions as the latent space. The choice of projection basis depends on the problem's properties, with common options including Galerkin projection [8], Least-Squares Petrov-Galerkin [9–12] projection and other derivates of Petrov-Galerkin [13].

Despite its utility, the reduction capability of POD is often limited by its linear nature, particularly in problems with slow decaying Kolmogorov n-width [14], such as advection-dominated problems. This limitation has encouraged the development of non-linear reduction techniques able to achieve more compact representations. The formal theory for generic non-linear projection-based ROM was introduced in [15], and it was specifically demonstrated by using a convolutional neural network as a mapping between the full and latent spaces. Despite the several limitations of this architecture, i.e. limited scaling capabilities and requirement of structured meshes, it paved the way for further non-linear architectures. These include methods based on quadratic manifolds [16] and the PROM-ANN architecture from [1]; the latter essentially being a non-linear approximation of POD using dense neural networks.

In this paper, we explore a new factor to take into account when constructing non-linear ROM operators. While existing projection-based ROMs construct their projection operators based exclusively on learning the reconstruction of solution snapshots, we propose incorporating the residuals of the HFM into the training of these operators. The reason being that, since the residual is the quantity to be be optimized during ROM inference, its more accurate representation will necessarily improve the solutions of the ROMs.

Our methodology is inspired by Physics-Informed Neural Networks (PINNs) [17] and their variants, e.g. [18,19]. PINNs are models based on neural networks that learn the solution function to a system of partial differential equations (PDEs) in its continuous form. They do so by incorporating the PDEs into their training loss and taking advantage of the automatic differentiation capabilities of common machine learning libraries. However, their application in engineering workflows can be hindered by their difficulties to cope with: irregular or discontinuous domains, complex PDEs due to spectral bias (which hinders their ability to handle high-frequency terms [20]), discontinuities (such as shock waves [21]), and plasticity (requiring significant effort to develop workarounds for these issues [22]).

In contrast, the Finite Element Method (FEM) [23], along with other HFMs like Finite Differences or Finite Volumes Methods, remains the gold standard for solving complex physical behaviors, particularly in engineering applications. This has led to recent efforts to bridge FEM and PINNs. For example, [24,25] proposed discrete PINNs that use FEM to compute residuals and their derivatives for backpropagation in linear problems. These approaches predict nodal values using neural networks that take simulation parameters as the only input. Meanwhile, the spatial discretization and boundary conditions are implicitly integrated via the FEM residual. While [25] focuses exclusively on structural mechanics, [24] generalizes the approach but still only for linear PDEs. Another intermediate approach is presented in [26], which introduces a physics-informed neural operator inference [27] framework that takes a discretized, variational form of the PDE as the training loss. This variational form is based on the energy of the system and is closely related to the FEM one, although not generalizable to the same degree. They remark how traditional PINNs only enforce the strong form of the PDE on the collocation points, resulting in lack of global smoothness, while a variational approach ensures this implicitly.

 doi:10.20944/preprints202504.0645.v1

In this work, we propose integrating concepts from PINNs into the field of intrusive projection-based ROMs. Specifically, we propose a method to incorporate physics information into the ROM non-linear approximation manifold by using the FEM residual as the training loss. This requires two key components: (1) a flexible ROM architecture capable of accommodating custom loss functions for the definition of the projection operators, and (2) a framework to integrate FEM residuals into the training process. For the architecture, we adapt the PROM-ANN framework from [1,28], which uses neural networks in a scalable manner. We modify this architecture to suit our needs and develop a framework to integrate FEM residuals into a neural network's loss, along with their Jacobians for the back-propagation. Our approach assumes the user has access to a fully functional FEM software capable of providing these quantities —a reasonable requirement given the need for an HFM to generate training data and perform intrusive ROM.

Our residual-based loss function differs from previous studies in its generality and flexibility. Instead of minimizing the FEM residual norm directly, we compute the difference between the obtained residual and a ground truth, thereby learning the behavior of the residual in non-converged conditions. We also provide an optimized implementation for computing the gradient of the residual loss, enabling efficient backpropagation.

To validate our methodology, we apply it to a steady-state structural mechanics problem involving a cantilever modeled by an unstructured mesh. The FEM software of choice is KratosMultiphysics [29], which we adapted for this purpose. Our results show that the modified PROM-ANN architecture, combined with the residual-based loss, achieves slightly but consistently improved accuracy in ROM simulations. We do acknowledge that training with the residual loss is computationally expensive compared to the traditional snapshot-based loss. To address this, we propose using residual training only as a fine-tuning step after initial snapshot-based training, significantly reducing overall training time.

While adapting the architecture of PROM-ANN to suit our specific needs, we also identified opportunities to enhance the original design, enabling it to learn effectively across a broader range of scenarios, even without incorporating the residual-based loss. These improvements are presented as additional contributions in this paper.

In summary, our contributions are threefold:

1. Enhanced PROM-ANN Architecture: We modify the PROM-ANN framework to improve its learning capabilities, particularly in cases with fast singular value decay.
2. Residual-Based Loss Function: We introduce a flexible, residual-based loss function that allows users to utilize their already-existing FEM infrastructure to perform physics-informed training of neural networks. We also specify an efficient implementation strategy for it.
3. Exploration of the effects of physics-informed manifold on projection-based ROM: We explore the impact of training the decoder using a residual-based loss function, both in terms of pure data reconstruction and of online ROM simulation. We obtain encouraging results, particularly for the latter one.

Our findings suggest that focusing on residual behavior could unlock further potential in non-linear ROMs, particularly when combined with architectures specifically designed for this purpose.

The rest of the paper is organized as follows. Section 2 provides a review of the main methods that form the basis for this work, i.e. PINNs and projection-based ROM with an emphasis on the original PROM-ANN architecture. In Section 3 we propose a discrete, FEM-based residual loss and develop an adequate implementation strategy for it. Then, Section 4 provides a series of modifications to the original PROM-ANN architecture and loss that make it compatible with problems with fast-decaying singular values. Section 5 effectively merges the developments presented in the two previous sections to enable physics-informed non-linear ROM. Further on, Section 6 explains the specific FEM problem in which the developments will be tested, as well as the software of choice and the neural-network training strategy. Section 7 shows the results of applying the methods developed throughout the paper onto our specific use-case. Section 8 further discusses the implications of the results from the previous

 doi:10.20944/preprints202504.0645.v1

section and proposes further research directions for future improvements. Finally, 9 closes the paper with the most significant conclusions.

2. Background

2.1. Physics-Informed Neural Networks

In this section we introduce PINNs, highlighting the most relevant aspects to our case. Our focus is on PINNs for forward parametric problems. Comprehensive reviews on the numerous variations of PINNs can be found in e.g. [18,30]

Consider a physical problem in the following form

$$\begin{cases} \mathcal{R}(\boldsymbol{u}(\boldsymbol{z};\boldsymbol{\mu}),\boldsymbol{\mu}) = \mathbf{0} & \boldsymbol{z} \in \Omega \times (0,T]\,, \\ \mathcal{B}(\boldsymbol{u}(\boldsymbol{z};\boldsymbol{\mu}),\boldsymbol{z}) = g(\boldsymbol{z},\boldsymbol{\mu}) & \boldsymbol{z} \in \partial\Omega \times (0,T]\,, \\ \mathcal{I}(\boldsymbol{u}(\boldsymbol{z};\boldsymbol{\mu}),\boldsymbol{z}) = f(\boldsymbol{z},\boldsymbol{\mu}) & \boldsymbol{z} \in \Omega \times \{0\}\,, \end{cases} \tag{1}$$

where $\Omega \subset \mathbb{R}^d$ represents the spatial domain with boundary $\partial\Omega$, $\boldsymbol{u} : \Omega \times \mathbb{R}_+ \times \mathbb{R}^p \to \mathbb{R}^{N_{\text{dof}}}$ is the unknown solution, $\boldsymbol{z}$ is a spatio-temporal coordinates vector, $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^p$ is the parameters vector encapsulating geometrical, material, and/or initial and boundary conditions. $\mathcal{R}(\cdot)$ is the differential operator in residual form encapsulating the physics. $\mathcal{B}(\cdot)$ is an operator indicating boundary conditions, and $\mathcal{I}(\cdot)$ is an operator indicating initial conditions.

In PINNs one uses a neural network for representing the solution function, as

$$\boldsymbol{u} \approx \mathcal{N}(\boldsymbol{z},\boldsymbol{\mu};\boldsymbol{\Theta})\,, \tag{2}$$

where $\mathcal{N}$ denotes the neural network, and $\boldsymbol{\Theta}$ encapsulates its learnable parameters. The optimal parameters for the neural network are obtained by minimizing a loss function:

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta})\,, \tag{3}$$

where the loss function $\mathcal{L}(\boldsymbol{\Theta})$ can be splitted into

$$\mathcal{L}(\boldsymbol{\Theta}) = \omega_{\mathcal{R}}\mathcal{L}_{\mathcal{R}}(\boldsymbol{\Theta}) + \omega_d\mathcal{L}_d(\boldsymbol{\Theta})\,. \tag{4}$$

Here, ω_d and $\omega_{\mathcal{R}}$ are weighting factors that balance the contribution of the data-driven loss $\mathcal{L}_d$ and the residual loss $\mathcal{L}_{\mathcal{R}}$, respectively, to the overall loss function. The data-driven loss $\mathcal{L}_d$ is related to the mismatch between mN_d ground-truth data points $\boldsymbol{u}^*_{ij}$, and the output of the PINN for the same spatio-temporal and parametric coordinates. This data-driven loss $\mathcal{L}_d$ reads

$$\mathcal{L}_d = \text{MSE}_d = \frac{1}{mN_d}\sum_{j=1}^{m}\sum_{i=1}^{N_d}\left\|\mathcal{N}(\boldsymbol{z}_i,\boldsymbol{\mu}_j;\boldsymbol{\Theta}) - \boldsymbol{u}^*_{ij}\right\|^2\,, \tag{5}$$

where N_d stands for the number of collocation points considered in the spatio-temporal domain. Moreover, m represents the number of points considered in the parametric space $\mathcal{P}$.

The residual loss $\mathcal{L}_{\mathcal{R}}$ relates to the substitution in the residual operator, as defined in Eq. (1), of the unknown solution with its neural network approximation as defined in Eq. (2), and reads

$$\mathcal{L}_{\mathcal{R}} = \text{MSE}_{\mathcal{R}} = \frac{1}{mN_{\mathcal{R}}}\sum_{j=1}^{m}\sum_{i=1}^{N_{\mathcal{R}}}\left\|\mathcal{R}(\mathcal{N}(\boldsymbol{z}_i,\boldsymbol{\mu}_j;\boldsymbol{\Theta}),\boldsymbol{\mu}_j)\right\|^2\,, \tag{6}$$

where $N_{\mathcal{R}}$ represents the number of residual evaluation points, which might be different from the collocation points used in the data-driven loss. Admittedly, from the substitution of Eq. (2) into Eq. (1), two more terms could be considered, corresponding to the initial and boundary conditions. However,

 doi:10.20944/preprints202504.0645.v1

here we assume that the initial and boundary conditions have been embedded into the PINN, as in [31,32], therefore ensuring their compliance, and not entering in the loss function.

The fact that the PINN itself is a differentiable mapping from spatio-temporal coordinates to the corresponding solution means that gradients of the solution and temporal derivatives of it are directly computable within the deep learning software via auto-differentiation. This allows for the internal computation of the loss and its gradient.

2.2. Projection-Based Reduced Order Models

2.2.1. The Full Order Model (FOM)

In the initial formulation of our problem, we represented the physics using a continuous residual form, as shown in Eq. (1), revisited here for clarity:

$$\mathcal{R}(\boldsymbol{u}(\boldsymbol{z};\boldsymbol{\mu});\boldsymbol{\mu}) = \mathbf{0} \qquad \boldsymbol{z} \in \Omega \times (0, T] \,. \tag{7}$$

To adapt this continuous form for computational analysis, we discretize the domain using FEM. By applying FEM discretization to Eq. (7) and incorporating initial and boundary conditions, we arrive at a set of governing equations in the form of a $\boldsymbol{\mu}$-parametric operator $\mathbf{R} : \mathbb{R}^N \times \mathcal{P} \to \mathbb{R}^N$. This results in the following discrete representation:

$$\mathbf{R}(\mathbf{u};\boldsymbol{\mu}) = \mathbf{0}\,, \tag{8}$$

where $\mathbf{u} \in \mathbb{R}^N$ is the FOM solution vector containing the value of the solution on every degree of freedom of the spatial discretization, and $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^p$ is the parameters vector encapsulating e.g. geometrical variations, material properties or boundary conditions.

Given an initial guess of the FOM solution at a step t, symbolically represented as $\mathbf{u}_t^0(\boldsymbol{\mu})$[1], the actual solution corresponding to the step can be obtained via an iterative method (e.g. Newton's) like

$$-\mathbf{J}(\mathbf{u}_t^k(\boldsymbol{\mu});\boldsymbol{\mu})\delta\mathbf{u}_t^k(\boldsymbol{\mu}) = \mathbf{R}(\mathbf{u}_t^k(\boldsymbol{\mu});\boldsymbol{\mu}) \tag{9a}$$

$$\mathbf{u}_t^{k+1}(\boldsymbol{\mu}) = \mathbf{u}_t^k(\boldsymbol{\mu}) + \delta\mathbf{u}_t^k(\boldsymbol{\mu}) \tag{9b}$$

where $\mathbf{J} = \frac{\partial \mathbf{R}}{\partial \mathbf{u}}$ is a jacobian matrix (or an approximation thereof), k is the current non-linear iteration, and $\delta\mathbf{u}_t^k \in \mathbb{R}^N$ is the solution state increment.

In practice, the FEM discretization can yield millions of degrees of freedom, making each nonlinear iteration prohibitively expensive. This challenge grows acute in scenarios demanding rapid solutions (e.g., design optimization, digital twins). To address this, various surrogate models have emerged, ranging from intrusive methods that directly manipulate the governing equations to non-intrusive black-box approaches. This work builds upon an intrusive, projection-based reduced-order model.

2.3. Manifold Projection-Based ROMs

Following the standard procedure introduced in [15] for non-linear projection-based ROMs (also known as manifold-based ROMs), we begin by approximating the FOM's solution state variable $\mathbf{u}^*(\boldsymbol{\mu})$ through a general decoder function defined as:

$$\mathbf{u}^*(\boldsymbol{\mu}) \approx \mathbf{u}(\boldsymbol{\mu}) = D_u(\mathbf{q}(\boldsymbol{\mu})), \tag{10}$$

where $D_u : \mathbb{R}^n \to \mathbb{R}^N$ maps a reduced solution $\mathbf{q}(\boldsymbol{\mu})$ in a latent space to a solution $\mathbf{u}(\boldsymbol{\mu})$ in the full space, and $n \ll N$.

Correspondingly, we introduce its encoder:

$$E_u : \mathbb{R}^N \to \mathbb{R}^n, \quad \mathbf{u}(\boldsymbol{\mu}) \mapsto \mathbf{q}(\boldsymbol{\mu}). \tag{11}$$

[1] For our current discussion can be either a time step, or a loading step following a predefined trajectory

 doi:10.20944/preprints202504.0645.v1

When Eq. (10) is substituted into Eq. (8), we obtain a residual characterized by:

$$\mathbf{R}(\mathbf{u}(\boldsymbol{\mu});\boldsymbol{\mu}) = \mathbf{R}(D_u(\mathbf{q}(\boldsymbol{\mu}));\boldsymbol{\mu}). \tag{12}$$

Finding the best reduced solution $\boldsymbol{q}$ involves solving the following minimization problem:

$$\min_{\mathbf{q}(\boldsymbol{\mu})\in\mathbb{R}^n} \frac{1}{2}\|\mathbf{R}(D_u(\mathbf{q}(\boldsymbol{\mu}));\boldsymbol{\mu})\|^2_{\mathbf{G}}\,. \tag{13}$$

Particular choices of the norm-defining matrix $\mathbf{G} \in \mathbb{R}^{N\times N}$ lead to different methods. In particular setting $\boldsymbol{G} := \boldsymbol{J}^{-T}$ leads to the Galerkin method, which we utilize in this work. For the derivation of the Galerkin method and other methods by particular choices of $\boldsymbol{G}$, the interested reader is directed to [13].

In this way, given an initial guess for the ROM solution, symbolically represented as $\mathbf{u}^0(\boldsymbol{\mu}) \in \mathbb{R}^N$, the ROM solution can be obtained using the Galerkin method by solving iteratively the following system of equations

$$\left(\frac{\partial D_u(\mathbf{q}^k(\boldsymbol{\mu}))}{\partial \mathbf{q}}\right)^T \mathbf{J}(\mathbf{u}^k(\boldsymbol{\mu});\boldsymbol{\mu})\frac{\partial D_u(\mathbf{q}^k(\boldsymbol{\mu}))}{\partial \mathbf{q}}\delta\mathbf{q}^k(\boldsymbol{\mu}) = -\left(\frac{\partial D_u(\mathbf{q}^k(\boldsymbol{\mu}))}{\partial \mathbf{q}}\right)^T \mathbf{R}(\mathbf{u}^k(\boldsymbol{\mu});\boldsymbol{\mu}) \tag{14a}$$

$$\mathbf{q}^{k+1}(\boldsymbol{\mu}) = \mathbf{q}^k(\boldsymbol{\mu}) + \delta\mathbf{q}^k(\boldsymbol{\mu}) \tag{14b}$$

$$\mathbf{q}^0(\boldsymbol{\mu}) = E_u(\mathbf{u}^0(\boldsymbol{\mu})) \tag{14c}$$

2.4. Neural Network-Augmented Projection-Based ROM (PROM-ANN)

In this work, we focus on the PROM-ANN proposed by Barnett et al.[1]. This methodology constructs a nonlinear approximation manifold by augmenting a traditional linear reduced-order basis ROB with a nonlinear correction obtained via an artificial neural network (ANN). It builds upon earlier developments in quadratic approximation manifolds[16], addressing limitations inherent to purely linear ROM approximations, especially regarding problems exhibiting slowly decaying Kolmogorov n-width.

2.4.1. Construction of the Nonlinear Approximation Manifold

The PROM-ANN methodology constructs the solution manifold by leveraging a low-dimensional primary basis and enhancing it with a nonlinear mapping learned from data. Specifically, POD is employed by factorizing a snapshot matrix $\mathbf{S}_u \in \mathbb{R}^{N\times m}$ (containing m high-fidelity solutions) via a truncated singular value decomposition:

$$\mathbf{S}_u = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T, \tag{15}$$

where $\mathbf{U} \in \mathbb{R}^{N\times(n+\bar{n})}$ contains the left singular vectors, and $\boldsymbol{\Sigma}$ is the diagonal matrix of singular values σ_i, ordered by magnitude. From this decomposition, we construct the primary ROB $\boldsymbol{\Phi} \in \mathbb{R}^{N\times n}$ using the first n dominant modes, and a secondary ROB $\bar{\boldsymbol{\Phi}} \in \mathbb{R}^{N\times\bar{n}}$ using the next $\bar{n}$ subdominant modes. Given a full-order solution vector $\mathbf{u}^*(\boldsymbol{\mu}) \in \mathbb{R}^N$, we approximate it with a nonlinear manifold representation defined as follows:

$$\mathbf{u}^*(\boldsymbol{\mu}) \approx \mathbf{u}(\boldsymbol{\mu}) = D_u(\mathbf{q}(\boldsymbol{\mu})) = \mathbf{u}_{\text{ref}} + \boldsymbol{\Phi}\mathbf{q}(\boldsymbol{\mu}) + \bar{\boldsymbol{\Phi}}\mathcal{N}(\mathbf{q}(\boldsymbol{\mu});\boldsymbol{\Theta}) \tag{16a}$$

$$\mathbf{q}(\boldsymbol{\mu}) = E_u(\mathbf{u}^*(\boldsymbol{\mu})) = \boldsymbol{\Phi}^T(\mathbf{u}^*(\boldsymbol{\mu}) - \mathbf{u}_{\text{ref}}), \tag{16b}$$

where $\mathbf{u}_{\text{ref}} \in \mathbb{R}^N$ is a suitable reference solution, typically chosen as the mean of the snapshots, $\mathbf{q}(\boldsymbol{\mu}) \in \mathbb{R}^n$ are the primary reduced coordinates, encoding the dominant features of the solution, $\mathcal{N}(\mathbf{q};\boldsymbol{\Theta}) : \mathbb{R}^n \to \mathbb{R}^{\bar{n}}$ is a neural network with parameters $\boldsymbol{\Theta}$, creating a nonlinear mapping from the primary to the secondary reduced coordinates, thus providing a nonlinear correction to the linear subspace approximation.

 doi:10.20944/preprints202504.0645.v1

The ANN acts as a nonlinear map, capturing correlations between dominant (primary) and subdominant (secondary) mode coefficients. Rather than explicitly reconstructing truncated information, the ANN provides a nonlinear relationship that significantly improves the compactness and accuracy of the reduced-order representation.

To solve the reduced-order system defined in Eq. (14), it is necessary to explicitly compute the derivative of the decoder function with respect to the primary reduced coordinates. For the PROM-ANN architecture defined in Eq. (16), this derivative reads:

$$\frac{\partial D_u(\mathbf{q}(\boldsymbol{\mu}))}{\partial \mathbf{q}} = \boldsymbol{\Phi} + \bar{\boldsymbol{\Phi}} \frac{\partial \mathcal{N}(\mathbf{q}(\boldsymbol{\mu}); \boldsymbol{\Theta})}{\partial \mathbf{q}}. \tag{17}$$

The derivative of the neural network term $\frac{\partial \mathcal{N}(\mathbf{q}(\boldsymbol{\mu}); \boldsymbol{\Theta})}{\partial \mathbf{q}}$ can be computed efficiently using automatic differentiation frameworks typically employed in neural network training.

2.4.2. Training the Neural Network

The ANN is trained using precomputed solution snapshots from the high-fidelity model. For each training snapshot indexed by $j = 1, \ldots, m$, we first compute the primary and secondary reduced coordinates by projection onto their respective ROBs:

$$\mathbf{q}_j(\boldsymbol{\mu}_j) = \boldsymbol{\Phi}^T(\mathbf{u}_j^*(\boldsymbol{\mu}_j) - \mathbf{u}_{\text{ref}}), \tag{18a}$$

$$\bar{\mathbf{q}}_j(\boldsymbol{\mu}_j) = \bar{\boldsymbol{\Phi}}^T(\mathbf{u}_j^*(\boldsymbol{\mu}_j) - \mathbf{u}_{\text{ref}}). \tag{18b}$$

The original PROM-ANN approach [1] defines the training loss function directly on the discrepancy between the secondary coordinates $\bar{\mathbf{q}}_j$ and the ANN prediction $\mathcal{N}(\mathbf{q}_j; \boldsymbol{\Theta})$:

$$\mathcal{L}_{\text{PROM-ANN}} = \frac{1}{m} \sum_{j=1}^{m} \|\bar{\mathbf{q}}_j(\boldsymbol{\mu}_j) - \mathcal{N}(\mathbf{q}_j(\boldsymbol{\mu}_j); \boldsymbol{\Theta})\|^2. \tag{19}$$

While very computationally efficient, this approach lacks a proper normalization strategy in order to guarantee that the neural network can be effectively trained. We will study this issue in Section 4.

3. Discrete PINN-like Loss

Our non-linear ROM architecture aims to incorporate the physics of the problem into the approximation manifold construction. The equivalent effect is accomplished in traditional PINNs by auto-differentiating the neural network's output with regard to a given point in continuous space and time, using the strong form of the PDE system. In contrast, a discrete approach like ours, requires a numerical approximation by discretization of the PDE, which can be done using a variety of techniques such as the finite element, or finite volume methods. The integration of these discretized residuals into neural networks places our approach within the category of informed machine learning, as detailed by [33].

The proposal to use discrete approximation for substituting the auto-differentiation in PINNs for residual minimization is introduced almost simultaneously in [24,25]. Both of these approaches introduce NN architectures for solving forward problems of linear, steady-state simulation and [24] additionally presents a model for backwards problems of the same nature. Their conceptualization doesn't differ too much from classical PINNs in terms of inputs, outputs, and loss definition: they take the parameters vector for the desired simulation as input, and return the results of the nodal variables of the system as output. The spatial information that is typically an input in PINNs is intrinsically defined in the FEM solver at the time of the residual computation.

 doi:10.20944/preprints202504.0645.v1

In terms of the training loss, both approaches aim to minimize the mean squared L2-norm of the residual, as determined by the FEM solver. This minimization considers the predicted snapshot (nodal variables) and the relevant simulation parameters:

$$\mathcal{L}_{\mathbf{R}} = \frac{1}{mN} \sum_{j=1}^{m} \left\| \mathbf{R}(\mathcal{N}(\boldsymbol{\mu}_j; \boldsymbol{\Theta}); \boldsymbol{\mu}_j) \right\|^2 . \tag{20}$$

where $\boldsymbol{\Theta}$ are the trainable parameters of the neural network conforming the PINN, index j specifies the different samples to be used within the batch, m is the number of samples in the batch and N is the number of degrees of freedom in our system (and therefore the size of the residual). An important remark about this loss is that it requires the removal of contributions in the residual $\mathbf{R}(\mathcal{N}(\boldsymbol{\mu}_j; \boldsymbol{\Theta}); \boldsymbol{\mu}_j)$ at the degrees of freedom with Dirichlet conditions, otherwise the norm will not approach zero.

The exact implementation of this loss differs in both approaches. The one in [25] is limited to static linear cases of structural mechanics and proposes a specific loss scaling strategy for these cases. They also propose an implementation strategy to perform the loss computation in batches. Meanwhile, the approach in [24] is more general, but still only applicable to linear PDEs. This limitation to linear cases significantly simplifies the loss implementation, as the residual becomes of the form $\mathbf{R}(\mathbf{u}; \boldsymbol{\mu}) = \mathbf{K}(\boldsymbol{\mu})\mathbf{u} - \mathbf{F}(\boldsymbol{\mu})$. So both the stiffness matrix $\mathbf{K}$ and the vector of external contributions $\mathbf{F}$ are independent of the current snapshot. Accordingly, the loss becomes:

$$\mathcal{L}_{\mathbf{R}} = \frac{1}{mN} \sum_{j=1}^{m} \left\| \mathbf{K}(\boldsymbol{\mu}_j)\mathcal{N}(\boldsymbol{\mu}_j; \boldsymbol{\Theta}) - \mathbf{F}(\boldsymbol{\mu}_j) \right\|^2 \tag{21}$$

where one could have precomputed all $\mathbf{K}(\boldsymbol{\mu}_j)$ and $\mathbf{F}(\boldsymbol{\mu}_j)$, thus allowing the whole optimization to be done via auto-differentiation, without calling the FEM software.

Even the approaches in more recent papers like [34,35] are still designed only for linear problems, with the difference being their specific use cases, the architectures they use for the surrogate model and the loss being just the norm of the residual, instead of the square of it.

In contrast, our approach differs in two key aspects: (1) the loss function is formulated in a parameter-agnostic manner, and (2) it is designed to handle both linear and non-linear cases, thereby enabling a much broader range of applications. As with PINNs, our method also allows for a combination of physics-based and data-driven losses. The following subsections discuss these three developments.

3.1. Parameter-Agnostic Loss

One aspect in which our methodology diverges significantly from the original idea of PINNs is that we are not training a self-contained surrogate model (typically in the form of a single neural network $\mathcal{N}(\boldsymbol{\mu}_j)$) but instead, the approximation manifold defined by a trainable encoder-decoder pair $D_u(E_u(\mathbf{u}_j^*))$. As we can see, the latter case does not take the simulation parameters $\boldsymbol{\mu}_j$ as an input, so it is agnostic to them. This is because the FEM-based ROM software itself will be in charge of conditioning the simulation with those parameters and then finding the most appropriate solution within our latent space.

But not only that. Traditionally, for a loss like the one in Eq. (20), we need to specify the exact parameters $\boldsymbol{\mu}_j$ for the solution we are seeking, as that is the one that will yield a residual of zero and therefore enable the learning by minimization. Instead, we design a loss function in which the parameters applied in the FEM software can be arbitrary. In this new loss we are not merely minimizing the properly parameterized residual of the predicted quantity $D_u(E_u(\mathbf{u}_j^*))$, but the difference between this quantity and the residual of the target solution $\mathbf{u}_j^*$, both with constant parameter $\boldsymbol{\mu}$:

$$\mathcal{L}_{\mathbf{R}} = \frac{1}{mN} \sum_{j=1}^{m} \left\| \mathbf{R}(D_u(E_u(\mathbf{u}_j^*)); \boldsymbol{\mu}) - \mathbf{R}(\mathbf{u}_j^*; \boldsymbol{\mu}) \right\|^2 . \tag{22}$$

 doi:10.20944/preprints202504.0645.v1

In this case, the trainable parameters $\boldsymbol{\Theta}$ are contained within the encoder and/or decoder of our ROM, and their specific form depends on the chosen architecture. Strictly speaking, we should write $D_u(E_u(\mathbf{u}_j^*;\boldsymbol{\Theta});\boldsymbol{\Theta})$. However, for clarity and conciseness, we will omit $\boldsymbol{\Theta}$ from the notation in what follows.

Our approach aims to minimize the discrepancy in residual behavior when it is non-zero, ensuring the ROM approximation manifold captures not only the converged solution but also the behavior of the residual outside of convergence situations. In our approach, the decoder will be readily integrated into the Newton iterative procedure. Thus, we aim for it to enhance its residual representation near the convergence space, aiding in achieving an optimal converged solution. On the same line, we no longer have the restriction that we mentioned for Eq. (20) on the components of the residual associated to Dirichlet conditions. Meaning that we can leave those components in order to learn them too. The main caveat of using this approach is that it still requires the original HFM's snapshots samples $\mathbf{u}_j^*$ even when training via the residual. In our specific case, the encoder-decoder architecture of choice will need these snapshots regardless, so it is not a major inconvenient.

Something important to consider is that Dirichlet conditions must be imposed strongly in the solutions vector whenever we compute the residual. That means that whatever architecture we use as encoder-decoder pair, it needs to operate on and predict only the degrees of freedom unaffected by Dirichlet conditions. The fixed ones are given their corresponding value forcefully.

3.2. In-Training Integration of FEM Software

The obstacle impeding going from linear to non-linear cases in other FEM-based residual losses [24, 25] is not a theoretical or mathematical difficulty in generating the residuals themselves or their derivatives. We have extensive FEM software at our disposal that already does precisely this. The difficulty is in dynamically integrating this information in an effective way during the training of the decoder.

Let us study the loss we proposed in Eq. (22) in more detail to identify our needs during training. The loss itself needs the values for two residuals: $\mathbf{R}(D_u(E_u(\mathbf{u}_j^*));\boldsymbol{\mu})$ and $\mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu})$. The latter can be pre-computed, as it will be constant for sample j during the whole training. The first one, however, has to be computed within the FEM software with an updated snapshot value at each training step.

Now, the actual training of the model happens during the backpropagation of the loss, which necessitates calculating the gradient of the loss with respect to $\boldsymbol{\Theta}$. This is typically done automatically thanks to the auto-differentiation capabilities of deep learning frameworks. However, as we take part of the loss computation onto external software, we are unable to do this and we have to code the gradient manually. The derivative of the loss function $\mathcal{L}_{\mathbf{R}}$ can be decomposed using the chain rule:

$$\frac{\partial \mathcal{L}_{\mathbf{R}}}{\partial \boldsymbol{\Theta}} = \frac{2}{mN}\sum_{j=1}^{m}\Big(\mathbf{R}(\mathbf{u}_j;\boldsymbol{\mu}) - \mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu})\Big)^T \frac{\partial \mathbf{R}(\mathbf{u}_j;\boldsymbol{\mu})}{\partial \mathbf{u}_j}\frac{\partial \mathbf{u}_j}{\partial \boldsymbol{\Theta}}. \tag{23}$$

where $\mathbf{u}_j = D_u(E_u(\mathbf{u}_j^*))$ is the current prediction for the nodal values of the solution.

The factor $\frac{\partial \mathbf{R}(\mathbf{u}_j;\boldsymbol{\mu})}{\partial \mathbf{u}_j}$ is the Jacobian matrix of the residual function with respect to the predicted solution vector $\mathbf{u}_j$. This Jacobian is computed and assembled entirely in the FEM software, given the current snapshot and arbitrary simulation parameter. Within the context of FEM, it is expressed as:

$$\mathbf{J}_F(\mathbf{u}_j;\boldsymbol{\mu}) = \frac{\partial \mathbf{R}(\mathbf{u}_j;\boldsymbol{\mu})}{\partial \mathbf{u}_j}\,. \tag{24}$$

The final factor $\frac{\partial \mathbf{u}_j}{\partial \boldsymbol{\Theta}}$ represents another Jacobian, this time of the predicted snapshot with respect to the trainable parameters. We can express it as:

$$\mathbf{J}_D(\mathbf{u}_j;\boldsymbol{\mu}) = \frac{\partial \mathbf{u}_j}{\partial \boldsymbol{\Theta}}\,. \tag{25}$$

 doi:10.20944/preprints202504.0645.v1

This latter derivative is applied only on the computations performed in the trainable encoder-decoder pair $D_u(E_u(\mathbf{u}_j^*))$. As such, this factor is self-contained within the deep learning framework and does not require additional external data. Such a Jacobian could be obtained by using the `tf.GradientTape.batch_jacobian()` method in TensorFlow, or equivalent methodologies in other deep learning frameworks.

To compute the loss gradient as it is defined explicitly in Eq. (23), we just need to collect the factors coming from the FEM software into the deep learning framework. However, this computation would be unnecessarily inefficient because of two main reasons described next. For each of these, we propose a corresponding implementation strategy:

1. First, take into account that $\mathbf{J}_F(\mathbf{u}_j;\boldsymbol{\mu})$ is typically a high-dimensional and highly sparse matrix. So, moving it directly from one software to another could be costly, especially if it cannot be treated as a sparse structure in all computations. FEM software is typically designed to cope with these kinds of matrices and to perform sparse vector-matrix multiplications. Therefore, for each snapshot we precompute the quantities:

$$\begin{aligned} \mathbf{e}_{\mathbf{R},j} &\coloneqq \Big(\mathbf{R}(\mathbf{u}_j;\boldsymbol{\mu}) - \mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu})\Big)^T \\ \mathbf{v}_{\mathbf{R},j} &\coloneqq \mathbf{e}_{\mathbf{R},j}\mathbf{J}_F(\mathbf{u}_j;\boldsymbol{\mu}) \end{aligned} \tag{26}$$

   within the FEM software. Now both of these are vectorial quantities, making the transfer to the deep learning framework more efficient. $\mathbf{e}_{\mathbf{R},j}$ is the error used to compute the loss, as $\mathcal{L}_\mathbf{R} = \frac{1}{mN}\sum_{j=1}^m \left\|\mathbf{e}_{\mathbf{R},j}\right\|^2$. And $\mathbf{v}_{\mathbf{R},j}$ will be used to compute the gradient of the loss.
2. Then, the main bottleneck comes from needing to compute m full Jacobian matrices $\{\mathbf{J}_D(\mathbf{u}_j;\boldsymbol{\mu})\}_{j=1}^m$ via auto-differentiation, instead of just the gradient of a single scalar $\mathcal{L}_\mathbf{R}$. Avoiding this is straightforward when reformulating the gradient of the loss with vectors $\mathbf{v}_{\mathbf{R},j}$. We realize that, to the deep learning framework, externally-computed $\mathbf{v}_{\mathbf{R},j}$ is no longer seen as a function depending on the model parameters $\boldsymbol{\Theta}$, but as a constant instead. This means that we can treat it as such during the computation and get:

$$\frac{\partial \mathcal{L}_\mathbf{R}}{\partial \boldsymbol{\Theta}} = \frac{2}{mN}\sum_{j=1}^m \mathbf{v}_{\mathbf{R},j}\frac{\partial \mathbf{u}_j}{\partial \boldsymbol{\Theta}} = \frac{\partial}{\partial \boldsymbol{\Theta}}\left(\frac{2}{mN}\sum_{j=1}^m \mathbf{v}_{\mathbf{R},j}\mathbf{u}_j\right) \tag{27}$$

   Therefore, we compute a single gradient of a scalar value for the whole batch, via auto-differentiation.

Particularly in our case, we take KratosMultiphysics as our choice of FEM software. As it is open-source, it enables us to develop all the custom functionalities needed to take a given set of snapshots $\{\mathbf{u}_j\}_{j=1}^m$ (those of the current batch), and for each one apply it as the current solution to then output $\mathbf{e}_{\mathbf{R},j}$ and $\mathbf{v}_{\mathbf{R},j}$.

3.3. Data-Based Loss Term

Even when training with the residual loss, it might be beneficial to introduce a data-related loss simultaneously. This is common in traditional PINNs in two ways: specifically on the boundary and initial conditions, to try to enforce Dirichlet conditions [17] (unless a special approach is used to establish them strongly [32]), and on collocation points sampled inside the simulation space, as a regularization that can help the overall loss converge faster [18].

In our case the Dirichlet conditions are enforced strongly and managed by the FEM software for their effect on the residual. But we could still introduce such a loss term for regularization purposes. Thus, the total loss function would be defined by two components:

$$\mathcal{L} = \omega_\mathbf{R}\mathcal{L}_\mathbf{R} + \omega_d\mathcal{L}_d\ . \tag{28}$$

 doi:10.20944/preprints202504.0645.v1

where $\omega_{\mathbf{R}}$ and ω_d are tunable hyper-parameters to balance both terms as needed, and the data-related loss $\mathcal{L}_d$ is defined simply as the mean squared error between the currently predicted snapshot $\mathbf{u}_j = D_u(E_u(\mathbf{u}_j^*))$ and the ground-truth one $\mathbf{u}_j^*$:

$$\mathcal{L}_d = \frac{1}{mN} \sum_{j=1}^{m} \left\| \mathbf{u}_j - \mathbf{u}_j^* \right\|^2 . \tag{29}$$

Then, the implementation of the total loss $\mathcal{L}$ and its gradient starts by defining the following vectors as constants:

$$\begin{aligned} \mathbf{e}_{d,j} = \mathbf{v}_{d,j} &\coloneqq (\mathbf{u}_j - \mathbf{u}_j^*)^T , \\ \mathbf{e}_{\mathbf{R},j} &\coloneqq \left(\mathbf{R}(\mathbf{u}_j; \boldsymbol{\mu}) - \mathbf{R}(\mathbf{u}_j^*; \boldsymbol{\mu}) \right)^T , \\ \mathbf{v}_{\mathbf{R},j} &\coloneqq \mathbf{e}_{\mathbf{R},j} \mathbf{J}_F(\mathbf{u}_j; \boldsymbol{\mu}) , \end{aligned} \tag{30}$$

and then using these to compute the loss as

$$\mathcal{L} = \frac{1}{mN} \sum_{j=1}^{m} \left(\omega_{\mathbf{R}} \left\| \mathbf{e}_{\mathbf{R},j} \right\|^2 + \omega_d \left\| \mathbf{e}_{d,j} \right\|^2 \right) \tag{31}$$

And finally its gradient via auto-differentiation:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Theta}} = \frac{\partial}{\partial \boldsymbol{\Theta}} \left(\frac{2}{mN} \sum_{j=1}^{m} \left(\omega_{\mathbf{R}} \mathbf{v}_{\mathbf{R},j} + \omega_d \mathbf{v}_{d,j} \right) \mathbf{u}_j \right) . \tag{32}$$

4. Modifications to the PROM-ANN Architecture

We adopt the PROM-ANN architecture from [1] as the foundation for developing our physics-informed ROM model, which introduces the use of neural networks while ensuring scalability and avoiding the requirement of a structured mesh for the underlying simulation. The architecture was introduced in Section 2.4, where we noted that it can be interpreted as a non-linear extension of classical POD that introduces additional effective modes without increasing the size of the latent space.

However, we find that the architecture, as originally proposed, is difficult to train—at least in our case, where the underlying problem exhibits a rapidly decaying singular value spectrum. To address this, we first revise both the architecture and the data-driven loss formulation to improve training capability. Later, in the following section, we will then incorporate our residual-based loss into the framework.

The modifications we propose in this section are as follows: (1) scaling of reduced coefficients, (2) changing the data-based loss for one that takes the full snapshot into account and (3) properly scaling the loss

4.1. Scaling of Reducedcoefficients

The main concern for us from the original architecture is that there is no normalization or scaling of the reduced coefficients prior to using them in the neural network. Ideally, neural networks are designed to operate on inputs that are approximately independent and identically distributed (i.i.d.), as this assumption facilitates more effective learning and optimization [36,37]. This can rarely be guaranteed, but it is still common practice to perform some sort of normalization or scaling on the inputs so that they all operate in a similar range of values. Otherwise, serious issues may arise during training, mainly because of some of the inputs being ignored.

In the case of PROM-ANN, the inputs and outputs of the neural network are the coefficients for a given snapshot (remember from Section 2.4 that we train a network such that $\mathcal{N}(\boldsymbol{\Phi}^T \mathbf{u}_j^*) \approx \bar{\boldsymbol{\Phi}}^T \mathbf{u}_j^*$). These coefficients $\mathbf{q}_j^* = \boldsymbol{\Phi}^T \mathbf{u}_j^*$ and $\bar{\mathbf{q}}_j^* = \bar{\boldsymbol{\Phi}}^T \mathbf{u}_j^*$ will scale similarly to the singular value decay observed when performing SVD on the training data. Essentially, for a fast-decaying singular value profile, each

 doi:10.20944/preprints202504.0645.v1

coefficient will result in a considerably smaller range of coefficients than the previous one, causing the neural network to learn only from a few of the first ones.

In order to correct this issue, we modify the architecture of the encoder and the decoder themselves to include a pair of scaling matrices:

$$\begin{aligned} E_u(\mathbf{u}^*) &= \mathbf{q} = \boldsymbol{\Xi}^{-1}\boldsymbol{\Phi}^T\mathbf{u}^* \\ D_u(\mathbf{q}) &= \boldsymbol{\Phi}\boldsymbol{\Xi}\mathbf{q} + \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\Xi}}\mathcal{N}(\mathbf{q}) \end{aligned} \tag{33}$$

where both the projection matrices $\boldsymbol{\Phi}$, $\bar{\boldsymbol{\Phi}}$ and the scaling matrices $\boldsymbol{\Xi}$, $\bar{\boldsymbol{\Xi}}$ come from the SVD decomposition of the snapshots matrix:

$$\begin{aligned} &SVD(\mathbf{S_u}) = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \\ &\boldsymbol{\Phi} = [\mathbf{U}_1|...|\mathbf{U}_n], \quad \bar{\boldsymbol{\Phi}} = [\mathbf{U}_{n+1}|...|\mathbf{U}_{\bar{n}}] \\ &\boldsymbol{\Xi} = \frac{1}{\sqrt{M}}\begin{bmatrix}\sigma_1 & & & \\ & \sigma_2 & & \\ & & ... & \\ & & & \sigma_n\end{bmatrix}, \quad \bar{\boldsymbol{\Xi}} = \frac{1}{\sqrt{M}}\begin{bmatrix}\sigma_{n+1} & & & \\ & \sigma_{n+2} & & \\ & & ... & \\ & & & \sigma_{\bar{n}}\end{bmatrix} \end{aligned} \tag{34}$$

and σ_i are the singular values found in the diagonal of the $\boldsymbol{\Sigma}$ matrix. Finally, M is the total number of samples in $\mathbf{S_u}$.

This procedure effectively re-scales each coefficient to have a roughly equivalent range. In the ideal case where all rows of the training snapshot matrix $\mathbf{S}_u$ have zero mean, multiplying by the matrix $\boldsymbol{\Xi}^{-1}$ scales the quantity $\mathbf{Q} = \boldsymbol{\Xi}^{-1}\mathbf{U}^T\mathbf{S}_u$ such that the covariance matrix between its rows becomes the identity. In other words, the modes in $\mathbf{Q}$ are uncorrelated—already ensured by the projection onto $\boldsymbol{\Phi}$—and have unit variance.

In our case, the condition that all row means in $\mathbf{S}_u$ are exactly zero does not hold. Nevertheless, as we will show in Section 6, the resulting scaling still leads to reduced coefficients with similar magnitudes in our particular use case.

We intentionally avoid enforcing zero-mean values, as we do not wish to apply any offset to the projected values. This design choice is crucial because it enables a neural network architecture without biases, thereby ensuring that $D_u(E_u(\mathbf{0})) = \mathbf{0}$ holds. Additionally, the proposed scaling procedure preserves the orthogonality of the projection matrices, so we still have $E_u(D_u(\mathbf{q})) = \mathbf{q}$.

Finally, no additional computation is required to obtain the scaling factors, since the SVD is already performed as part of the original methodology. In this sense, the approach is highly efficient.

4.2. Corrected Data-Based Loss

Now we have an architecture that provides appropriately-scaled features for the neural network. But at the same time, this scaling may not be the best if we want to apply the training loss as in the original paper[1]. That is, measuring the error on the predicted $\tilde{\mathbf{q}}_j$ coefficients themselves. This loss would translate like this to our architecture:

$$\mathcal{L}'_d = \frac{1}{mN}\sum_{j=1}^{m}\left\|\mathcal{N}(\mathbf{q}_j^*) - \bar{\boldsymbol{\Xi}}^{-1}\bar{\boldsymbol{\Phi}}^T\mathbf{u}_j^*\right\|^2 = \frac{1}{mN}\sum_{l=1}^{m}\left\|\mathcal{N}(\boldsymbol{\Xi}^{-1}\boldsymbol{\Phi}^T\mathbf{u}_j^*) - \bar{\boldsymbol{\Xi}}^{-1}\bar{\boldsymbol{\Phi}}^T\mathbf{u}_j^*\right\|^2 \tag{35}$$

The rationale behind dividing by N is to ensure consistency with the loss formulations introduced in Section 3.

Such a loss will give the same importance to all of the output features, or reduced coefficients in this case. We know from the nature of POD that this is not desirable, as lower modes should be given more importance than higher ones. In fact what we have to do is to reverse, within the loss, the scaling

 doi:10.20944/preprints202504.0645.v1

that we applied to the reduced coefficients. We can do so by applying $\bar{\boldsymbol{\Xi}}$ on the contents of the norm in Eq. (35):

$$\mathcal{L}_{d,\text{compact}} = \frac{1}{mN}\sum_{l=1}^{m}\left\|\bar{\boldsymbol{\Xi}}\left(\mathcal{N}(\mathbf{q}^*) - \bar{\boldsymbol{\Xi}}^{-1}\bar{\boldsymbol{\Phi}}^T\mathbf{u}^*\right)\right\|^2 \tag{36}$$

But we could also achieve the same effect by just enforcing the whole reconstructed full-order snapshot to approximate the ground-truth one. That is, using exactly the same data-based loss that we had defined in Section 3.3:

$$\mathcal{L}_d = \frac{1}{mN}\sum_{j=1}^{m}\left\|\mathbf{u}_j - \mathbf{u}_j^*\right\|^2 = \frac{1}{mN}\sum_{j=1}^{m}\left\|\boldsymbol{\Phi}\boldsymbol{\Xi}\mathbf{q}^* + \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\Xi}}\mathcal{N}(\mathbf{q}^*) - \mathbf{u}_j^*\right\|^2 \tag{37}$$

This second loss is more intuitive, as we make it explicit to achieve the final goal for our ROM approximation manifold: learning the full snapshot itself. However, it is not difficult to prove that both Eq. (36) and Eq. (37) are equivalent up to a constant offset. That is, assuming that all snapshots $\{\mathbf{u}_j^*\}_{j=1}^m$ were included in the set $\mathbf{S_u}$ to which we performed the SVD.

Proof. By developing from $\mathcal{L}_d$:

$$\mathcal{L}_d = \frac{1}{mN}\sum_{j=1}^{m}\left\|\mathbf{u}_j - \mathbf{u}_j^*\right\|^2 = \frac{1}{mN}\sum_{j=1}^{m}\left\|\boldsymbol{\Phi}\boldsymbol{\Xi}\mathbf{q}^* + \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\Xi}}\mathcal{N}(\mathbf{q}^*) - \mathbf{u}_j^*\right\|^2$$

By developing $\mathbf{u}_j^* = (\boldsymbol{\Phi}\boldsymbol{\Xi}\boldsymbol{\Xi}^{-1}\boldsymbol{\Phi}^T + \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\Xi}}\bar{\boldsymbol{\Xi}}^{-1}\bar{\boldsymbol{\Phi}}^T + \hat{\boldsymbol{\Phi}}\hat{\boldsymbol{\Xi}}\hat{\boldsymbol{\Xi}}^{-1}\hat{\boldsymbol{\Phi}}^T)\mathbf{u}_j^*$. Where $\hat{\boldsymbol{\Phi}}$ is the orthonormal basis containing the singular vectors from $SVD(\mathbf{S_u})$ that were not included in neither $\boldsymbol{\Phi}$ or $\bar{\boldsymbol{\Phi}}$, and $\hat{\boldsymbol{\Xi}}$ is analog but for the singular values matrix. Then applying it:

$$\begin{aligned}\mathcal{L}_d &= \frac{1}{mN}\sum_{j=1}^{m}\left\|\boldsymbol{\Phi}\boldsymbol{\Xi}\mathbf{q}^* + \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\Xi}}\mathcal{N}(\mathbf{q}^*) - \left(\boldsymbol{\Phi}\boldsymbol{\Xi}\mathbf{q}^* + \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\Xi}}\bar{\boldsymbol{\Xi}}^{-1}\bar{\boldsymbol{\Phi}}^T\mathbf{u}_j^* + \hat{\boldsymbol{\Phi}}\hat{\boldsymbol{\Xi}}\hat{\boldsymbol{\Xi}}^{-1}\hat{\boldsymbol{\Phi}}^T\mathbf{u}_j^*\right)\right\|^2 \\ &= \frac{1}{mN}\sum_{j=1}^{m}\left\|\bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\Xi}}\left(\mathcal{N}(\mathbf{q}^*) - \bar{\boldsymbol{\Xi}}^{-1}\bar{\boldsymbol{\Phi}}^T\mathbf{u}_j^*\right) - \hat{\boldsymbol{\Phi}}\hat{\boldsymbol{\Phi}}^T\mathbf{u}_j^*\right\|^2\end{aligned}$$

By acknowledging that $\bar{\boldsymbol{\Phi}}$ and $\hat{\boldsymbol{\Phi}}$ are orthogonal to each other:

$$\mathcal{L}_d = \frac{1}{mN}\sum_{j=1}^{m}\left(\left\|\bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\Xi}}\left(\mathcal{N}(\mathbf{q}^*) - \bar{\boldsymbol{\Xi}}^{-1}\bar{\boldsymbol{\Phi}}^T\mathbf{u}_j^*\right)\right\|^2 + \left\|\hat{\boldsymbol{\Phi}}\hat{\boldsymbol{\Phi}}^T\mathbf{u}_j^*\right\|^2\right)$$

By acknowledging that the L^2 norm is invariant to right-multiplication of an orthonormal matrix and that the terms $\left\|\hat{\boldsymbol{\Phi}}\hat{\boldsymbol{\Phi}}^T\mathbf{u}_j^*\right\|$ do not depend on trainable parameters $\boldsymbol{\Theta}$:

$$\mathcal{L}_d = \frac{1}{mN}\sum_{j=1}^{m}\left\|\bar{\boldsymbol{\Xi}}\left(\mathcal{N}(\mathbf{q}^*) - \bar{\boldsymbol{\Xi}}^{-1}\bar{\boldsymbol{\Phi}}^T\mathbf{u}_j^*\right)\right\|^2 + C = \mathcal{L}_{d,\text{compact}} + C$$

□

If the user does not intend on applying physics-informed training, then choosing $\mathcal{L}_{d,\text{compact}}$ would make most sense because of its reduced complexity. However, the fact that $\mathcal{L}_d$ takes the snapshots to the full space within the computation makes it most appropriate to be paired with a physics-based loss, which will require the full vector of nodal solutions anyways. In any of the two cases, the ideal would be to pre-compute $\mathbf{q}_j^*$ for all samples to improve efficiency.

 doi:10.20944/preprints202504.0645.v1

4.3. Scaling of the Data-Based Loss

As written in Eq. (37), $\mathcal{L}_d$ would exhibit high variability depending on the number of modes included in the primary ROB. That is because, by including more modes in $\mathbf{\Phi}$ while following their importance order, we are heavily limiting the possible error $\left\|\mathbf{u}_j - \mathbf{u}_j^*\right\|^2$ that we can have [2].

In order to correct this issue, we propose to use a pre-computed scaling factor that will be applied globally to the loss:

$$\mathcal{L}_d = \frac{1}{mN}\frac{1}{e_{\text{POD},d}}\sum_{j=1}^{m}\left\|\mathbf{u}_j - \mathbf{u}_j^*\right\|^2. \tag{38}$$

With the new factor $e_{\text{POD},d}$ defined as:

$$e_{\text{POD},d} \coloneqq \frac{1}{MN}\sum_{j=1}^{M}\left\|\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{u}_j^* - \mathbf{u}_j^*\right\|^2. \tag{39}$$

This quantity corresponds to the mean squared reconstruction error of the standard POD approach when only the primary modes are retained. It is averaged over all the samples in the training set, not only those in the batch, in order to make it representative for any choice of samples during the training, and thus, only having to compute it once at the beginning.

This choice of scaling also gives a much more interpretable value to the loss, as it will become an indicator to how much better the current model is compared to just using POD for the same size of latent space. In this sense, it is helpful when troubleshooting, as a value over 1 would clearly indicate that we are losing accuracy compared to POD.

4.4. Online Phase of Non-Linear ROM

In order to perform the online ROM simulation, we need to adapt the nonlinear iteration problem to the new architecture. Essentially, we substitute our decoder into the Galerkin-based non-linear iteration formulation for non-linear manifolds (as seen in Eq. (14)). The result would be:

$$\left(\mathbf{\Phi\Xi} + \bar{\mathbf{\Phi}}\bar{\mathbf{\Xi}}\frac{\partial\mathcal{N}(\mathbf{q}^k)}{\partial\mathbf{q}}\right)^T\mathbf{J}(\mathbf{u}(\boldsymbol{\mu});\boldsymbol{\mu})\left(\mathbf{\Phi\Xi} + \bar{\mathbf{\Phi}}\bar{\mathbf{\Xi}}\frac{\partial\mathcal{N}(\mathbf{q}^k)}{\partial\mathbf{q}}\right)\delta\mathbf{q}^k(\boldsymbol{\mu}) = -\left(\mathbf{\Phi\Xi} + \bar{\mathbf{\Phi}}\bar{\mathbf{\Xi}}\frac{\partial\mathcal{N}(\mathbf{q}^k)}{\partial\mathbf{q}}\right)^T\mathbf{R}(\mathbf{u}(\boldsymbol{\mu});\boldsymbol{\mu})$$
$$\mathbf{q}^{k+1}(\boldsymbol{\mu}) = \mathbf{q}^k(\boldsymbol{\mu}) + \delta\mathbf{q}^k(\boldsymbol{\mu}) \tag{40}$$

Meaning that at each non-linear iteration we have to recompute $\bar{\mathbf{\Phi}}\bar{\mathbf{\Xi}}\frac{\partial\mathcal{N}(\mathbf{q}^k)}{\partial\mathbf{q}}$.

The online ROM simulation is done entirely within the FEM software, KratosMultiphysics, as going back between this and the deep learning software would be unnecessarily expensive. Therefore, we made custom methods within KratosMultiphysics to compute both the forward pass of the neural network, $\mathcal{N}(\mathbf{q})$, and the jacobian of its outputs with respect to the inputs, $\frac{\partial\mathcal{N}(\mathbf{q})}{\partial\mathbf{q}}$. The other quantities are defined at initialization from the training results and are kept constant.

5. Integration of Physics-Based Loss into PROM-ANN

Here we take the developments done in the two last sections and join them in order to finally obtain a method to perform physics-informed non-linear projection-based ROM. Essentially, we will take the total $\mathcal{L}$ loss developed in Section 3 and apply it on the PROM-ANN formulation developed in Section 4, while keeping in mind the adaptations to the loss that we also performed in this latter section.

The full discrete PINN-like loss $\mathcal{L}'$, as formulated in Section 3 is:

$$\mathcal{L}' = \omega_{\mathbf{R}}\mathcal{L}'_{\mathbf{R}} + \omega_d\mathcal{L}'_d = \frac{\omega_{\mathbf{R}}}{mN}\sum_{j=1}^{m}\left\|\mathbf{R}(\mathbf{u}_j;\boldsymbol{\mu}) - \mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu})\right\|^2 + \frac{\omega_d}{mN}\sum_{j=1}^{m}\left\|\mathbf{u}_j - \mathbf{u}_j^*\right\|^2. \tag{41}$$

[2] This is true for snapshots within the set $\mathbf{S}_\mathbf{u}$, and should reflect in new samples if they are properly represented by the POD

 doi:10.20944/preprints202504.0645.v1

In Section 4.3 we already determined that a full-snapshot data-based loss like $\mathcal{L}'_d$ in Eq. (41) is appropriate to use once scaled by factor $1/e_{\text{POD},d}$. So that is the only modification we have to do on $\mathcal{L}'_d$.

But now, in order to still be able to use the residual-based loss term $\mathcal{L}'_{\mathbf{R}}$ appropriately, we should also scale it in a similar way to $\mathcal{L}'_d$. We propose modifying it as follows:

$$\mathcal{L}_{\mathbf{R}} = \frac{1}{mN}\frac{1}{e_{\text{POD},\mathbf{R}}}\sum_{j=1}^{m}\left\|\mathbf{R}(\mathbf{u}_j;\boldsymbol{\mu}) - \mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu})\right\|^2. \tag{42}$$

With $e_{\text{POD},\mathbf{R}}$ defined as the mean square error compared to the traditional POD, but this time in terms of the residual and not the snapshot itself:

$$e_{\text{POD},\mathbf{R}} = \frac{1}{MN}\sum_{j=1}^{M}\left\|\mathbf{R}(\boldsymbol{\Phi}\boldsymbol{\Phi}^T\mathbf{u}_j^*;\boldsymbol{\mu}) - \mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu})\right\|^2. \tag{43}$$

With this small modification, the full loss becomes:

$$\mathcal{L} = \omega_{\mathbf{R}}\mathcal{L}_{\mathbf{R}} + \omega_d\mathcal{L}_d = \frac{\omega_{\mathbf{R}}}{mN}\frac{1}{e_{\text{POD},\mathbf{R}}}\sum_{j=1}^{m}\left\|\mathbf{R}(\mathbf{u}_j;\boldsymbol{\mu}) - \mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu})\right\|^2 + \frac{\omega_d}{mN}\frac{1}{e_{\text{POD},d}}\sum_{j=1}^{m}\left\|\mathbf{u}_j - \mathbf{u}_j^*\right\|^2. \tag{44}$$

And the practical implementation of the loss itself and its gradient is:

$$\begin{aligned}\mathcal{L} &= \frac{1}{mN}\sum_{j=1}^{m}\left(\frac{\omega_{\mathbf{R}}}{e_{\text{POD},\mathbf{R}}}\left\|\mathbf{e}_{\mathbf{R},j}\right\|^2 + \frac{\omega_d}{e_{\text{POD},d}}\left\|\mathbf{e}_{d,j}\right\|^2\right),\\ \frac{\partial\mathcal{L}}{\partial\boldsymbol{\Theta}} &= \frac{\partial}{\partial\boldsymbol{\Theta}}\left(\frac{2}{mN}\sum_{j=1}^{m}\left(\frac{\omega_{\mathbf{R}}}{e_{\text{POD},\mathbf{R}}}\mathbf{v}_{\mathbf{R},j} + \frac{\omega_d}{e_{\text{POD},d}}\mathbf{v}_{d,j}\right)\mathbf{u}_j\right),\end{aligned} \tag{45}$$

where all vector quantities $\mathbf{e}_{\mathbf{R},j}$, $\mathbf{e}_{d,j}$, $\mathbf{v}_{\mathbf{R},j}$ and $\mathbf{v}_{d,j}$ are exactly as defined in Eq. (30) and treated as constants. And the definitions of the encoder and decoder are taken from Eq. (33).

As we only modified the loss for the offline stage of ROM, the online stage stays untouched. Therefore, once the decoder is trained, we would proceed exactly as in Section 4.4 with new cases to simulate.

6. Use-Case and Evaluation Methodology

The evaluation of our method is done on a quasi-static structural mechanics case, specifically a non-linear case of hyperelasticity. It simulates the deformation in a 2D rubber cantilever that is fixed at its left wall and which has two different and perpendicularly-oriented line loads, P_x and P_y, applied to its right end. The cantilever is defined by an unstructured mesh with 797 nodes, and the nodal variables to compute are the displacements in components X and Y. Therefore, our FOM system is of dimension $\mathcal{N} = 1594$. Figure 1 shows a schematic of the described setup.

We define the parametric space as the range of loads that can be applied in each direction: $\mathcal{P} = [-3000, 3000] \times [-3000, 3000]$ N/m. And the displacements are computed in reference to the unperturbed position of each node $\mathbf{u}(P_x = 0, P_y = 0)$. Figure 2 shows the deformation on the cantilever for a random set of lineload combinations within the parametric space.

In order to build the snapshot matrix for the training, we perform the ROM simulations of 5000 different cases with parameters defined by a 2D Halton pseudo-random number generator. From each of these simulations, we store both the full snapshot $\mathbf{u}_j^* \in \mathbb{R}^N$ and the corresponding residual $\mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu}) \in \mathbb{R}^N$. For the validation set we generate 1250 different samples using the same strategy, and we generate 300 more for the test set. We perform all evaluations over the test dataset; the training and the validation ones being used exclusively during the training process.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 April 2025 doi:10.20944/preprints202504.0645.v1

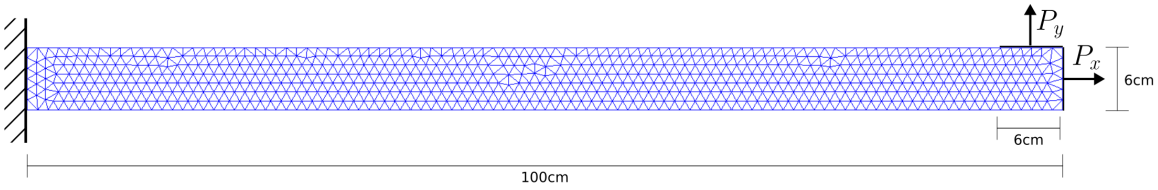


Figure 1. Mesh of the cantilever with line loads

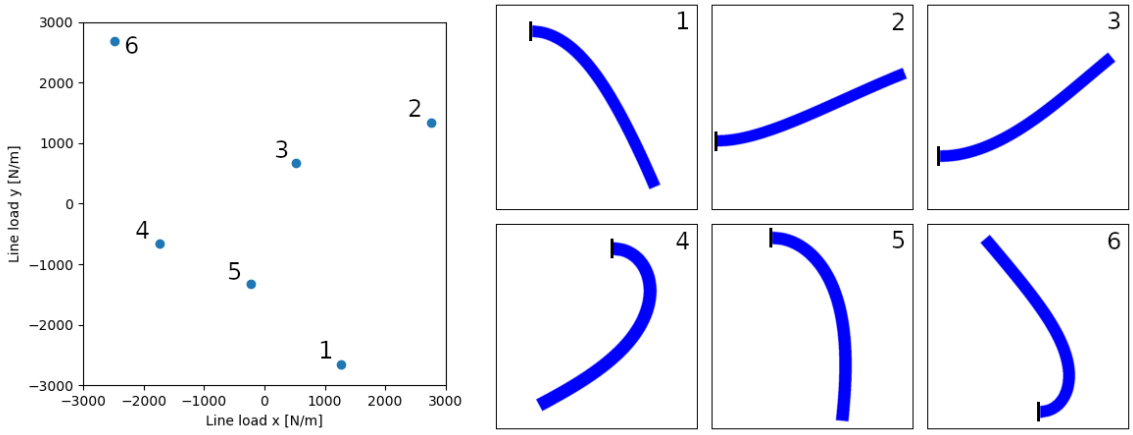


Figure 2. Examples of deformation in the cantilever for six different random combinations of line loads

$\mathbf{\Phi}$, $\bar{\mathbf{\Phi}}$, $\mathbf{\Xi}$ and $\bar{\mathbf{\Xi}}$ are obtained from the SVD of the snapshots dataset for training. Figure 3 shows the decay of the singular values' energy for the first 200 modes of the SVD. The number of modes in each projection matrix is chosen by taking into account the accuracy obtained via traditional POD-based ROM using the same amount of modes. We establish two limit cases: one in which we select a value of $n = 6$, that would lead to a $\sim 0.1\%$ error on the displacements snapshot using POD, and one with $n = 20$, which would achieve a relative error of $\sim 10^{-6}$. Then, we take a series of n values in between these to compare in our experiments. The secondary ROM basis contains a variable amount of modes so that $n + \bar{n} = 60$ always.

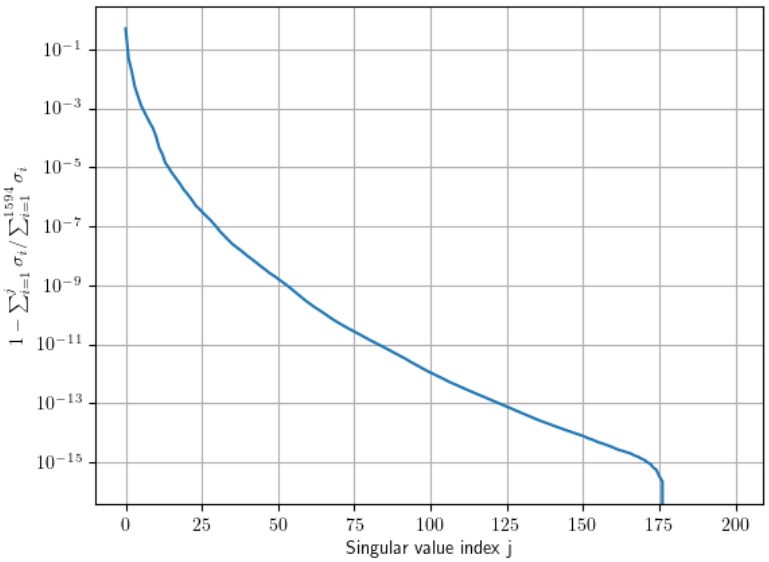


Figure 3. Singular values' energies for the first 200 modes of $\mathbf{S}_u$.

Two different error metrics are defined:

 doi:10.20944/preprints202504.0645.v1

- Relative error on snapshot:

$$e_u = \exp\left(\frac{1}{M}\sum_{j=1}^{M} \ln \frac{\left\|\mathbf{u}_j - \mathbf{u}_j^*\right\|}{\left\|\mathbf{u}_j^*\right\|}\right) \tag{46}$$

  This is the geometric mean of the relative errors of each ROM sample $\mathbf{u}_j$ compared to the FOM one $\mathbf{u}_j^*$.
- Relative error on residual:

$$e_{\mathbf{R}} = \exp\left(\frac{1}{M}\sum_{j=1}^{M} \ln \frac{\left\|\mathbf{R}(\mathbf{u}_j;\boldsymbol{\mu}) - \mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu})\right\|}{\left\|\mathbf{R}(\mathbf{u}_j^*;\boldsymbol{\mu})\right\|}\right) \tag{47}$$

  Again, the geometric mean of the samples' relative errors, but this time comparing the residual.

We will use these metrics in the next section, in order to compare the behavior of our models both in reconstruction and online ROM.

In terms of software, we use KratosMultiphysics [29] as our FEM framework, as it is open-source and lets us implement the required methods and interfaces in Python. For the neural network framework we choose TensorFlow.

All neural networks presented contain two hidden layers of 200 neurons each and with no bias. We use Exponential Linear Unit (ELU) as our activation, and use a learning rate scheduling strategy where the initial learning rate is decreased following a sinusoidal curve until a value of 10^{-6}. We select AdamW as the network optimizer, with the default values from TensorFlow. Finally, all trainings are done with batches of size $m = 16$.

7. Results

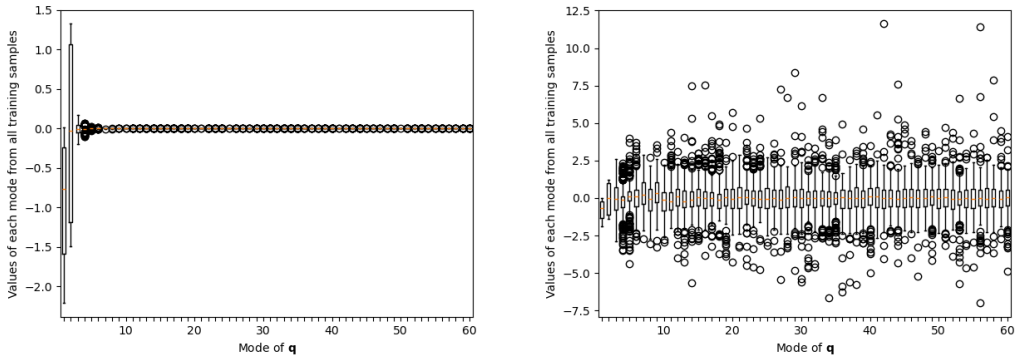
In this section we will go over the evaluation of the methods we have proposed throughout the paper. The first subsection will study the modifications to the original PROM-ANN [1] that were explained in Section 4. After that, we study the effect of training the architecture with the novel residual loss, as developed within Section 5. Finally we do a performance assessment.

7.1. Modifications on the Original PROM-ANN

In this section, we examine the modifications introduced to the original PROM-ANN architecture during Section 4. The modification involved first the incorporation of scaling matrices $\boldsymbol{\Xi}$ and $\bar{\boldsymbol{\Xi}}$ within the encoder-decoder pair and within the loss. Then we added a global, scalar scaling $1/e_{\text{POD},d}$ to the data-based loss. The main effects of these should be (1) achieving a similar range of values for all the neural network's inputs and (2) to make the loss and its gradients invariant to the choice of size for the latent space.

In order to confirm effect (1), we apply the encoder E_u on all snapshots from our training set, with fixed latent size $n = 60$. We do so with the encoder as defined in the original POD-ANN paper [1] (specified in Eq. (16)) and also with ours (specified in Eq (33)). Thus, obtaining $\mathbf{q}_{\text{orig},j} = \boldsymbol{\Phi}^T\mathbf{u}_j^* \in \mathbb{R}^{60}$ and $\mathbf{q}_j = \boldsymbol{\Xi}^{-1}\boldsymbol{\Phi}^T\mathbf{u}_j^* \in \mathbb{R}^{60}$, respectively, for all the snapshots. Then, in Figure 4 we plot the statistics of the obtained values for each component or mode in $\mathbf{q}_{\text{orig},j}$ and $\mathbf{q}_j$, side by side. For the original implementation (Figure 4a) only the first few modes have similarly large coefficients, then they become too small to differentiate. In contrast, by incorporating the singular values matrix, we obtain the result in Figure 4b, where all ranges are similarly large. These are, in fact, the range of values that would go into the neural network in each architecture.

 doi:10.20944/preprints202504.0645.v1



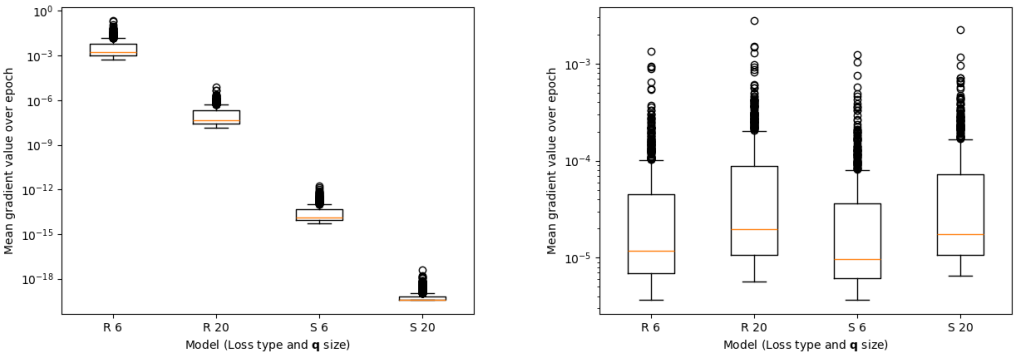
(a) **(b)**

Figure 4. Boxplots showing the spread of the values for each mode of $\mathbf{q}$, for all the samples in our training dataset. (a) shows the case where $\mathbf{q} = \mathbf{\Phi}^T\mathbf{u}^*$ and (b) the case where $\mathbf{q} = \mathbf{\Xi}^{-1}\mathbf{\Phi}^T\mathbf{u}^*$. The ranges are much more uniform in the latter case, which should be beneficial when training the neural network.

Moving on to effect (2), we do a similar study but on the range of the mean of the gradients $\frac{\partial \mathcal{N}(\mathbf{q}_j;\mathbf{\Theta})}{\partial \mathbf{\Theta}}$ obtained during back-propagation in the training routine. We compute the mean gradient for each batch, and then take the mean for all batches in the training epoch. Thus, collecting a single mean gradient value per epoch. We compare the trainings with and without scaling factors $1/e_{\text{POD},d}$ and $1/e_{\text{POD},\mathbf{R}}$ (otherwise as $\mathcal{L}$ in Eq. (44)). And for each of these two cases, we run four different training routines:

- *S6*: 6 primary modes ($n = 6$). Trained only on the data-based loss ($\omega_d = 1$, $\omega_{\mathbf{R}} = 0$).
- *S20*: 20 primary modes ($n = 20$). Trained only on the data-based loss ($\omega_d = 1$, $\omega_{\mathbf{R}} = 0$).
- *R6*: 6 primary modes ($n = 6$). Trained only on the residual loss ($\omega_d = 0$, $\omega_{\mathbf{R}} = 1$).
- *R20*: 20 primary modes ($n = 20$). Trained only on the residual loss ($\omega_d = 0$, $\omega_{\mathbf{R}} = 1$).

all of them for 800 epochs. We can see in Figure 5 the comparison of the statistics of these gradients for the different training modalities without the scalings (Figure 5a) and with the scalings (Figure 5b). While the former presents orders of magnitude in difference for these values depending on the size of the latent space, the latter one provides very uniform results.



(a) **(b)**

Figure 5. Boxplots showing the spread of the mean value per epoch of the applied gradients during training, four different training strategies. (a) Shows the results when no rescaling of the loss is applied. (b) Shows the results when the rescaling factors $e_{\text{POD},d}$ and $e_{\text{POD},\mathcal{R}}$ are applied. The latter case is invariant to the choice of latent space size, contrary to the first one.

The final step in the evaluation is to examine how do these modifications altogether affect the resulting decoders after training. That is, the capacity for the trained decoders to reconstruct the

 doi:10.20944/preprints202504.0645.v1

original snapshots, and also to serve as the approximation manifolds within ROM simulations. For this we compare three architectures and training losses:

- *s-loss*: Using the encoder-decoder pair introduced by us in Eq. (33). Train via the data-based loss $\mathcal{L}_d$ as defined in 38 for 800 epochs. Learning rate starting at 1e-3.
- *q-loss*: Using the original PROM-ANN encoder-decoder pair, as in Eq. (16). Train via the original loss $\mathcal{L}_{\text{PROM-ANN}}$ as defined in Eq. (19) for 800 epochs. Learning rate starting at 1e-3.
- *POD*: Using the fully linear POD encoder-decoder pair. Involves no training apart from the SVD computation.

We compare the relative snapshot error (e_u from Eq. (46)) applied on the reconstructed snapshots $\mathbf{u}_j = D_u(E_u(\mathbf{u}_j^*))$ for the whole test dataset. We do so for the three models specified above, and for all the $\mathbf{q}_j$ sizes specified in Section 6. Then we do the same but taking $\mathbf{u}_j$ as the result of the online ROM simulation with each architecture. The results are illustrated in Figure 6.

For the data reconstruction, the error for our proposed methodology, in *s-loss*, is several orders of magnitude lower than both *POD* and *q-loss*. While the error for *q-loss* is marginally better than *POD* only for the lowest sizes of $\mathbf{q}$. We attribute this to the bad conditioning of the NN input, at least for problems with fast-decaying singular values like our use-case. Without proper training, the secondary term of the PROM-ANN decoder will just add noise to the final snapshot, making it behave even worse than traditional POD.

These observations translate to a similar behavior in the ROM simulations. All the ROM simulations lose some accuracy compared to the data reconstruction, but this gap is even larger for the *q-loss* case.

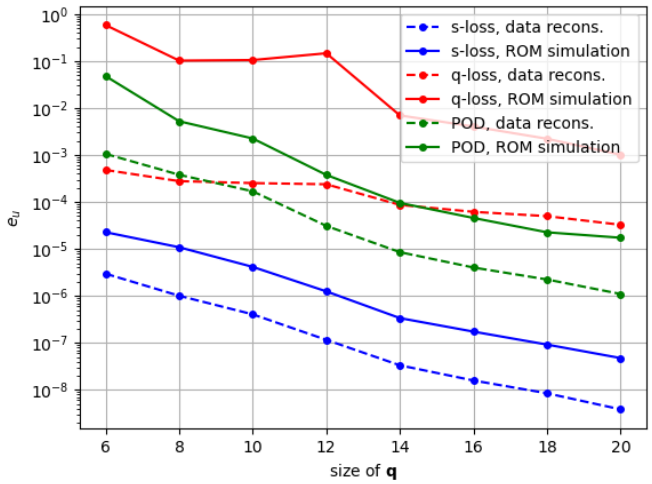


Figure 6. Comparison of the error in the snapshot, e_u, over three different architectures: (1) *s-loss* with our modified PROM-ANN method, (2) *q-loss* with the original PROM-ANN method in [1]and (3) *POD* with the traditional POD method. All of them for different dimensions of latent space. Dotted lines represent error on data reconstruction alone, while full lines represent error on online ROM simulation. *s-loss* consistently performs several orders of magnitude better than *POD*, while *q-loss* is not able to keep up with *POD* in most cases.

7.2. Comparison of Snapshot and Residual Losses

The next step is to make a comparison of the effect of training based purely on data versus training on residuals. Like the subsection before, we compare models with $\mathbf{q}$ size ranging from 6 to 20.

Figure 7 compares errors on snapshots (e_u from Eq. (46)). On the one hand, we check for the encoder-decoder data reconstruction and, on the other hand, for the results of the intrusive ROM execution. Meanwhile, Figure 8 compares the errors on residuals ($e_{\mathbf{R}}$ from Eq. (47)) only for the encoder-decoder reconstruction.

In both figures we compare two models trained differently:

- *s-loss*: Train using only the snapshot loss ($\mathcal{L}$ from Eq. (44) with $\omega_d = 1$, $\omega_{\mathbf{R}} = 0$) for 800 epochs. Learning rate starting at 1e-3. (Same as in the previous sub-section)

 doi:10.20944/preprints202504.0645.v1

- *r-loss*: Start weights on the ones resulting from *s-loss*. Then train for 800 epochs with only the residual loss ($\mathcal{L}$ from Eq. (44) with $\omega_d = 0$, $\omega_{\mathbf{R}} = 1$). Learning rate starting at 1e-4.

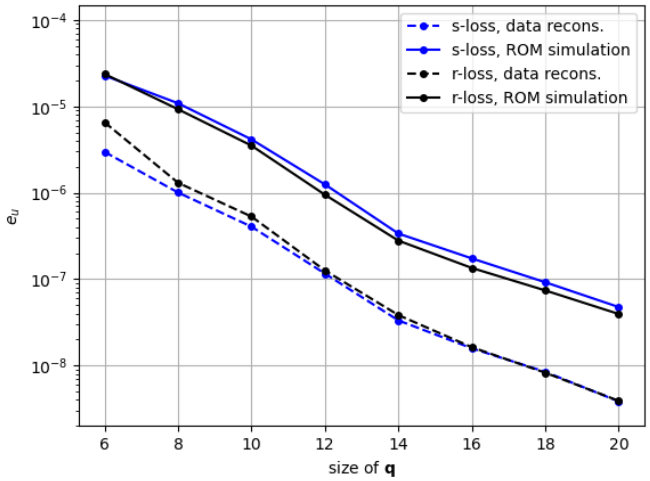


Figure 7. Comparison of the error in the snapshot, e_u, for models trained on loss $\mathcal{L}$ from Eq. (44) with two different configurations: (1) *s-loss* with ($\omega_d = 1$, $\omega_{\mathbf{R}} = 0$), from randomly initialized weights and (2) *r-loss* with ($\omega_d = 0$, $\omega_{\mathbf{R}} = 1$) but starting from the weights of its *s-loss* counterpart. Both of them for different dimensions of latent space. Dotted lines represent error on data reconstruction alone, while full lines represent error on online ROM simulation. *r-loss* consistently gets slightly worse accuracy in the data reconstruction, while its ROM simulation solutions are slightly better than *s-loss*.

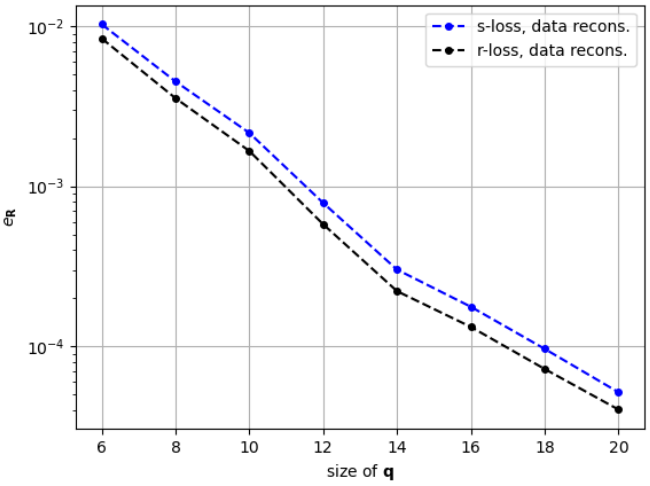


Figure 8. Comparison of the error in the residual, $e_{\mathbf{R}}$, for models *s-loss* and *r-loss* as described in Figure 7. Both of them for different dimensions of latent space. The error is computed only for the case of data reconstruction. *r-loss* consistently gets lower error than *s-loss*.

The behavior represented in Figure 7 is quite interesting. Although the final result of the ROM simulation using the *r-loss* model is not substantially better than its *s-loss* counterpart, it consistently achieves a higher accuracy. This is in total opposition to the results for the encoder-decoder reconstruction of the snapshots, in which *s-loss* gets the highest accuracy for all $\mathbf{q}$ sizes. This seemingly contradictory phenomenon is clarified when looking at the reconstruction residuals in Figure 8 which shows the *r-loss* performing consistently better. These results hint that the behavior of the intrusive ROM may be more influenced by the correctness of the residual representation for the snapshots in our ROM approximation manifold than by the accuracy of the snapshots per se.

7.3. Comparison of Training and Online ROM Runtimes

For the training time, we check the mean time spent to train a single batch, measured over an entire epoch. As stted in Section 6, all cases share the same architecture (two hidden layer of size 200)

 doi:10.20944/preprints202504.0645.v1

and train with batches of size 16. Computation is done on a single thread of an AMD Ryzen 7 5800x3d processor.

The description for the cases to compare is the following:

- Snapshot: Using loss $\mathcal{L}$ from Eq. (44) with $\omega_d = 1, \omega_{\mathbf{R}} = 0$. No interaction with FEM software.
- Residual (optimised): $\mathcal{L}$ from Eq. (44) with $\omega_d = 0$, $\omega_{\mathbf{R}} = 1$. Implementing the gradients computation using the optimizations detailed in Eq. (45).
- Residual (non-optimised): $\mathcal{L}$ from Eq. (44) with $\omega_d = 0, \omega_{\mathbf{R}} = 1$. Implementing the gradients computation in a naive way by computing first $\mathbf{A}_i = (\mathbf{R}(\mathbf{u}_i) - \mathbf{R}(\mathbf{u}_i^*))^T \frac{\partial \mathbf{R}}{\partial \mathbf{u}}$ in KratosMultiphysics and then computing $\mathbf{B}_i = \frac{\partial \mathbf{u}_i}{\partial \boldsymbol{\Theta}}$ via the `tf.GradientTape.batch_jacobian()` method in TensorFlow, then performing matrix multiplication in:

$$\frac{\partial \mathcal{L}_{\mathbf{R}}}{\partial \boldsymbol{\Theta}} = \frac{2}{e_{base,R}} \frac{1}{Nm} \sum_{i=1}^{N} \mathbf{A}_i \mathbf{B}_i \tag{48}$$

The results are printed in Table 1. We see that training via the residual with the naive implementation is not feasible, as the time for a single batch is in the order of seconds. Thanks to the optimizations described in Section 3.2 we avoid computing entire Jacobian matrices, which reduces the time to the order of 10 milliseconds. Still, as we expected, training the neural network via the residual loss takes considerably longer than training via the snapshot loss (around 40x longer). That is why in Section 7.2 we perform a finetuning on the residual loss, instead of training from random weights.

Table 1. Comparison of training time for a single batch, among different losses.

Loss type	Mean batch training time (s)
Snapshot	9.20e-4
Residual (optimised)	3.87e-2
Residual (non-optimised)	7.41

Next, we want to make a performance assessment for the online ROM execution. For that, we measure the time that the FEM software takes to solve the reduced linear system of equations. Specifically, we take the mean resolution time for all the non-linear iterations while solving 9 different cases from our test set. This is performed for two different configurations of the architecture proposed in this paper, two cases of traditional POD that achieve similar accuracies to the former pair, and the FOM simulation. Again, these evaluations have been performed using a single thread from an AMD Ryzen 7 5800x3d processor.

The results for different models can be seen in Table 2.

Table 2. Comparison of computation times for various tasks within the online ROM routine, over a set of different models.

Model	$\mathbf{q}$ size	e_u	System solve time (s)
Modified PROM-ANN (Eq. (33))	6	7.67e-5	1.42e-6
Modified PROM-ANN (Eq. (33))	20	6.12e-8	5.55e-6
POD	18	2.39e-5	4.84-6
POD	40	1.32e-7	1.99e-5
FOM	-	-	2.11e-3

As expected, the main computational advantage of our method (and of the original PROM-ANN [1], as both architectures are equivalent in terms of computational complexity) comes from the achieving smaller linear systems to solve for the same precision. It is noteworthy to point out that the biggest gains in computation time for the online process come with a further step in the

 doi:10.20944/preprints202504.0645.v1

ROM workflow called Hyper-Reduction, in which the amount of elements to take into account for the generation of the system at each step is greatly reduced. The number of reduced elements to use is directly related to the latent space dimension. Therefore, the reduction in modes that we achieve is also beneficial in this regard. This study focuses on the offline part of the ROM process, so there is no further development on the online procedure compared to the original PROM-ANN method. The implementation of Hyper-Reduction is described for this type of architecture in their paper [1], so interested readers are encouraged to check their performance assessments, which should be applicable to our case aswell.

8. Discussion and Future Work

Once we have assessed the performance of our proposed architecture and losses, we will further discuss the implications of these results and the impact of our contributions.

We comment first on the discrete FEM residual-based loss that we developed throughout Section 3, and its proposed implementation strategy. These developments pose a step-up from recent initiatives to design discrete PINN-like losses [24,25,34,35] that limit themselves to linear problems of different natures. By taking advantage of open-source FEM software like KratosMultiphysics [29], we can access all the FEM methodology needed to obtain residuals and Jacobians for non-linear cases in a wide range of state-of-the-art FEM formulations. The cost for this is a loss in time efficiency during training, compared to the classic data-based approach, because of the required dynamic interaction between the FEM software and the neural network framework during training. And also because of the manual computation of the gradient loss. It is in this sense that our implementation proposal makes a huge difference, taking the training time from being prohibitive to being just an order of magnitude higher than the data-based one. These comparisons are shown in Section 7.3. We believe that the main computational bottleneck after our proposed methodology is the fact that the FEM software currently runs the computations in series for each sample in the batch. Some future work on parallelizing these procedures could potentially unlock training times much closer to the data-based ones. Another key characteristic of the proposed loss is being parameter-agnostic. This makes it more versatile in various ways: in terms of efficiency the FEM software does not need to re-configure the simulation for each specific sample, and in terms of use-case it can be applied to cases in which not only the minimization of the residual itself is important, but also its behavior while being non-zero. This latter aspect is key in using this loss for our particular setting of intrusive ROM. One unexplored advantage of this loss formulation would be the possibility to perform partial physics-based learning, where only specific components of the total residual are used for the training. For example one could train only on the steady-state component of the residual of a dynamic case in order to avoid the inconveniences from the dependency on previous time-steps. This is not addressed in this paper, but left as possible future contributions. There are further options that the residual loss could open up and that we haven't fully explored, e.g. the possibility of training on the residual with noisy data in order to achieve data augmentation.

Next, we comment on the modification to the PROM-ANN architecture itself and the data-based loss $\mathcal{L}_d$ with the scaling matrices Ξ, $\bar{\Xi}$ and the global scaling $1/e_{\text{POD},d}$. The scaling matrices are an inexpensive way to normalize the input ranges for the neural network with the direct results from the SVD, so we avoid extra statistical studies of the dataset. The global scaling is a single scalar computed inexpensively via POD, only once for the whole training. The effect of these modifications is apparent in the results in Section 7.1, where both the reconstruction and ROM results from our architecture are several orders of magnitude better than the simpler, original approach described in [1]. Now, there is a very plausible explanation for the lack of performance of the original PROM-ANN, mostly when comparing with the good results that they obtain in their paper. The use case that they use for evaluation is a 2D inviscid Burgers problem, which has a much flatter decay in the SVD's singular values compared to ours. Thus, the range of their inputs to the neural network should naturally be more uniform. Another possibility is that they apply some normalization routine prior to the neural

 doi:10.20944/preprints202504.0645.v1

network without mentioning it explicitly. In any case, we can say that our modification makes the architecture generalizable to any kind of problem in terms of their singular value decay. Additionally, the global scaling $1/e_{\text{POD},d}$ is key in order to stabilize the scale of the loss and the backpropagation gradients, making the neural network optimizer perform equally whatever the choice of latent size. It also has an interpretational purpose, which is to make the loss a direct indicator of how much better the results are relative to the simple POD version. Something to explore in the future would be better methodologies to choose the modes included in the primary and secondary ROBs. Right now this is done in a greedy way, selecting as many modes are needed to achieve a certain accuracy in POD, but it is not clear how different modes are related to one another (especially since they are uncorrelated in the linear sense). The implementation of our version of ANN-PROM (only with the data-based loss) is readily available within the ROM Application of KratosMultiphysics [29].

Finally, we enter the discussion about the effect of training our modified ANN-PROM architecture on the residual loss. Other than the increased duration of the training routine, it was also difficult for us to prevent the model from falling in non-optimal local minima during training with the residual loss. That is why the chosen approach was to train first with the data-based approach and only then finetune with a purely physics-based loss. The intuition for proposing this physics-based training is that the intrusive ROM accuracy is not only given by the snapshot-reconstruction capabilities of the approximation manifold (which would work more like a lower limit in terms of error), but should also be dependant on how well the residuals are represented within these solutions in the manifold. Essentially because the residual is the quantity being optimized during the intrusive ROM simulation. The general discrepancy between ROM and reconstruction is clearly demonstrated within our use-case in Section 7.1, with the error in ROM simulation being approximately one order of magnitude higher than the reconstruction one for both our proposed architecture with the data-based loss and traditional POD. Further on we look at the results in Section 7.2 to specifically understand the effect of the residual-based training. The observations are encouraging, even if not spectacular. We say this in the sense that performing ROM with the model trained on the residual provided slightly but consistently better results than the one trained with the snapshot. It can also be interpreted as achieving a slightly lower discrepancy between reconstruction and ROM simulation, which is what we were aiming for. But the most important insight is that this phenomenon coincides with a consistently lower error in the representation of the residual by the models trained on physics. We are aware that, as it is right now, the significantly higher training time with the residual loss renders the proposed method not attractive, taking into account the marginal increase in ROM accuracy, but we are optimistic that future research can enable more meaningful improvements.

We make the observation, in retrospective, that choosing the ANN-PROM architecture for implementing the residual loss limited the achievable accuracy. That is because, even if the neural network allows us to introduce the loss of our choosing, we still depend entirely on the modes that we gathered via SVD on our snapshots dataset, without any regard for the residual accuracy. The fact that even with this caveat we were able to achieve slightly better results makes us very optimistic for the future where we could explore new architectures (or variants of this one) that put the residual in the focus point from the beginning, or that allow more freedom to correct the residual on top of the modes for the snapshot.

All in all, the three main contributions of the paper work together to complement each other and obtain a physics-informed intrusive ROM framework, but also hold value by themselves: the residual loss could be applied for other purposes like non-intrusive ROM, the modifications to the ANN-PROM architecture make it more versatile in terms of the types of problems it can handle, and the study of the effect of the residual in intrusive ROM provides a seed for a new path of research in this discipline for us and for the rest of the community in numerical methods.

 doi:10.20944/preprints202504.0645.v1

9. Conclusions

In this paper, we extend the PROM-ANN architecture proposed in [1] by incorporating a training approach based on the finite element method (FEM) residual, rather than relying solely on snapshot data. This establishes a connection between non-linear reduced-order models (ROMs) and physics-informed neural networks (PINNs). While traditional PINNs use analytical partial differential equations (PDEs) to train continuous, non-intrusive models, our approach leverages discrete FEM residuals as the loss function for backpropagation, guiding the learning of the ROM approximation manifold. This development allows us to investigate the impact of improving the residual of the snapshots on the overall performance of projection based ROMs.

The path to achieve this final goal enables us to present three independently significant contributions: (1) a loss based on the FEM residual and which is parameter-agnostic and, most importantly, applicable to non-linear problems; (2) A modification on the original PROM-ANN architecture in [1] that makes it applicable to cases with fast-decaying singular values, and (3) a study on the effect of the residual-based training on ROM simulation. We demonstrate our approach in the context of static structural mechanics with non-linear hyperelasticity, specifically in the deformation of a rubber cantilever subjected to two orthogonal variable loads.

In terms of the residual loss, the fact that it is based on existing FEM software makes it applicable to a high range of problems. The proposed implementation strategy makes the interaction between the neural network and the FEM software viable in reasonable training time, around 40ms per batch. Finally, the enhancement of the resulting residuals via this loss is demonstrated by applying it on our proposed PROM-ANN-based architecture and performing data reconstruction. This results in a consistently lower residual representation compared to the cases trained on snapshot data alone.

About the modifications to the original ANN-PROM architecture, we observe how our method lowers the snapshot error by several orders of magnitude compared to POD, in both the data reconstruction and the ROM simulation results. This improvement is consistent for latent space sizes ranging from $n = 6$ to $n = 20$. In contrast, the original PROM-ANN formulation struggles to train the neural network, resulting in errors higher than POD in most cases. This enhancement from our methodology comes from a proper scaling strategy on both the architecture itself and the loss to be used.

Finally, for the effect of applying the FEM residual loss on the approximation manifold for projection-based ROM, our results show a modest but consistent reduction of the gap between the accuracy for snapshot reconstruction and the accuracy for projection-based ROM. While not being useful in practice as of now, this observation makes us optimistic that better results for projection-based ROM could be unlocked in the future by taking care of the residual representations within the approximation manifold. In the future, alternative non-linear ROM architectures that enable more control over the resulting solutions' residuals can greatly improve the results presented in this work.

Acknowledgments: N. Sibuet acknowledges the Secretariat of Universities and Research of the Department of Research and Universities of the Generalitat of Catalonia, as well as the European Social Plus Fund for their financial support through the predoctoral scholarship AGAUR-FI (2024 FI-3 00065) Joan Oró. S. Ares de Parga and J.R. Bravo acknowledge the Departament de Recerca i Universitats de la Generalitat de Catalunya for the financial support through the FI-SDUR 2020 and FI-SDUR 2021 scholarships. S. Ares de Parga also acknowledges support from the Fulbright Commission Spain through a Fulbright Predoctoral Research Fellowship (2024–2025).

Nomenclature

HFM	high-fidelity model
ROM	reduced order model
POD	proper orthogonal decomposition
SVD	singular value decomposition
PINN	Physics Informed Neural Network

 doi:10.20944/preprints202504.0645.v1

PDE	partial differential equation
FEM	finite elements method
$\mathcal{N} : (\boldsymbol{a}; \boldsymbol{\Theta}) \mapsto \boldsymbol{b}$	a neural Network parametrized by $\boldsymbol{\Theta}$
$\boldsymbol{u}$	nodal solution of FEM problem
$\boldsymbol{q}$	FEM solution representation in a given reduced space
$D : \boldsymbol{q} \mapsto \boldsymbol{u}$	a decoder function
$E : \boldsymbol{u} \mapsto \boldsymbol{q}$	an encoder function
$\boldsymbol{\mu}$	parameters vector for a FEM simulation
$\mathbf{R}(\boldsymbol{u}; \boldsymbol{\mu})$	FEM residual, given nodal solution $\boldsymbol{u}$ and simulation parameters $\boldsymbol{\mu}$
$\mathcal{L}(\boldsymbol{\Theta})$	loss function used to optimize parameters set $\boldsymbol{\Theta}$
$N \in \mathbb{N}$	number of degrees of freedom in the FOM
$n \in \mathbb{N}$	number of degrees of freedom in ROM. Number of POD modes. Latent space dimensions
$M \in \mathbb{N}$	number of samples in the training dataset
$m \in \mathbb{N}$	number of samples in a given batch when training a neural network

References

1. Barnett, J.; Farhat, C.; Maday, Y. Neural-network-augmented projection-based model order reduction for mitigating the Kolmogorov barrier to reducibility. *Journal of Computational Physics* **2023**, *492*, 112420. https://doi.org/https://doi.org/10.1016/j.jcp.2023.112420.
2. Brunton, S.L.; Nathan Kutz, J.; Manohar, K.; Aravkin, A.Y.; Morgansen, K.; Klemisch, J.; Goebel, N.; Buttrick, J.; Poskin, J.; Blom-Schieber, A.W.; et al. Data-driven aerospace engineering: reframing the industry with machine learning. *AIAA Journal* **2021**, *59*, 2820–2847.
3. Jiang, Y.; Yin, S.; Li, K.; Luo, H.; Kaynak, O. Industrial applications of digital twins. *Philosophical Transactions of the Royal Society A* **2021**, *379*, 20200360.
4. Hesthaven, J.S.; Rozza, G.; Stamm, B.; et al. *Certified reduced basis methods for parametrized partial differential equations*; SpringerBriefs in Mathematics, Springer Cham, 2016.
5. Sirovich, L. Turbulence and the dynamics of coherent structures. Part I: Coherent structures. *Quarterly of Applied Mathematics* **1987**, *45*, 561–571.
6. Holmes, P.; Lumley, J.L.; Berkooz, G. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*; Cambridge Monographs on Mechanics, Cambridge University Press, 1996.
7. Eckart, C.; Young, G. The approximation of one matrix by another of lower rank. *Psychometrika* **1936**, *1*, 211–218.
8. Quarteroni, A.; Rozza, G.; et al. *Reduced order methods for modeling and computational reduction*; Vol. 9, *Modeling, Simulation and Applications*, Springer Cham, 2014. https://doi.org/https://doi.org/10.1007/978-3-319-02090-7.
9. Carlberg, K.; Bou-Mosleh, C.; Farhat, C. Efficient non-linear model reduction via a least-squares Petrov-Galerkin projection and compressive tensor approximations. *International Journal for Numerical Methods in Engineering* **2011**, *86*. https://doi.org/10.1002/nme.3050.
10. Carlberg, K.; Farhat, C.; Cortial, J.; Amsallem, D. The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows. *Journal of Computational Physics* **2013**, *242*. https://doi.org/10.1016/j.jcp.2013.02.028.
11. Carlberg, K.; Barone, M.; Antil, H. Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction. *Journal of Computational Physics* **2017**, *330*. https://doi.org/10.1016/j.jcp.2016.10.033.
12. Grimberg, S.; Farhat, C.; Tezaur, R.; Bou-Mosleh, C. Mesh sampling and weighting for the hyperreduction of nonlinear Petrov–Galerkin reduced-order models with local reduced-order bases. *International Journal for Numerical Methods in Engineering* **2021**, *122*, 1846–1874.
13. de Parga, S.A.; Bravo, J.R.; Hernández, J.A.; Zorrilla, R.; Rossi, R. Hyper-reduction for Petrov–Galerkin reduced order models. *Computer Methods in Applied Mechanics and Engineering* **2023**, *416*. https://doi.org/10.1016/j.cma.2023.116298.
14. Pinkus, A. *n-Widths in Approximation Theory*; Springer Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-69894-1.
15. Lee, K.; Carlberg, K.T. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics* **2020**, *404*, 108973.
16. Barnett, J.; Farhat, C. Quadratic approximation manifold for mitigating the Kolmogorov barrier in nonlinear projection-based model order reduction. *Journal of Computational Physics* **2022**, *464*, 111348.

 doi:10.20944/preprints202504.0645.v1

17. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561* **2017**.
18. Cuomo, S.; Di Cola, V.S.; Giampaolo, F.; Rozza, G.; Raissi, M.; Piccialli, F. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing* **2022**, *92*, 88.
19. Li, Z.; Zheng, H.; Kovachki, N.; Jin, D.; Chen, H.; Liu, B.; Azizzadenesheli, K.; Anandkumar, A. Physics-Informed Neural Operator for Learning Partial Differential Equations. *ACM / IMS J. Data Sci.* **2024**, *1*. https://doi.org/10.1145/3648506.
20. Wang, S.; Yu, X.; Perdikaris, P. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics* **2022**, *449*, 110768. https://doi.org/https://doi.org/10.1016/j.jcp.2021.110768.
21. Fuks, O.; Tchelepi, H.A. LIMITATIONS OF PHYSICS INFORMED MACHINE LEARNING FOR NON-LINEAR TWO-PHASE TRANSPORT IN POROUS MEDIA. *Journal of Machine Learning for Modeling and Computing* **2020**, *1*. https://doi.org/10.1615/JMachLearnModelComput.2020033905.
22. Niu, S.; Zhang, E.; Bazilevs, Y.; Srivastava, V. Modeling finite-strain plasticity using physics-informed neural network and assessment of the network performance. *Journal of the Mechanics and Physics of Solids* **2023**, *172*, 105177. https://doi.org/10.1016/j.jmps.2022.105177.
23. Zienkiewicz, O. *The Finite Element Method in Engineering Science*; McGraw-Hill European Publishing Programme, McGraw-Hill, 1971.
24. Meethal, R.E.; Kodakkal, A.; Khalil, M.; Ghantasala, A.; Obst, B.; Bletzinger, K.U.; Wüchner, R. Finite element method-enhanced neural network for forward and inverse problems. *Advanced Modeling and Simulation in Engineering Sciences* **2023**, *10*. https://doi.org/10.1186/s40323-023-00243-1.
25. Le-Duc, T.; Nguyen-Xuan, H.; Lee, J. A finite-element-informed neural network for parametric simulation in structural mechanics. *Finite Elements in Analysis and Design* **2023**, *217*. https://doi.org/10.1016/j.finel.2022.103904.
26. Eshaghi, M.S.; Anitescu, C.; Thombre, M.; Wang, Y.; Zhuang, X.; Rabczuk, T. Variational Physics-informed Neural Operator (VINO) for solving partial differential equations. *Computer Methods in Applied Mechanics and Engineering* **2025**, *437*, 117785. https://doi.org/10.1016/j.cma.2025.117785.
27. Wang, S.; Wang, H.; Perdikaris, P. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances* **2021**, *7*, eabi8605, [https://www.science.org/doi/pdf/10.1126/sciadv.abi8605]. https://doi.org/10.1126/sciadv.abi8605.
28. Chmiel, M.R.; Barnett, J.L.; Farhat, C. Assessment of Projection-Based Model Order Reduction for a Benchmark Hypersonic Flow Problem. In Proceedings of the AIAA SciTech 2024 Forum, 2024, p. 0250.
29. Ferrándiz, V.M.; Bucher, P.; Zorrilla, R.; Warnakulasuriya, S.; Cornejo, A.; Rossi, R.; Roig, C.; jcotela.; Maria, J.; tteschemacher.; et al. KratosMultiphysics/Kratos: v10.1, 2024. URL: https://doi.org/10.5281/zenodo.14185721, https://doi.org/10.5281/zenodo.14185721.
30. Cai, S.; Mao, Z.; Wang, Z.; Yin, M.; Karniadakis, G.E. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica* **2021**, *37*, 1727–1738.
31. Zhu, Y.; Zabaras, N.; Koutsourelakis, P.S.; Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics* **2019**, *394*, 56–81. https://doi.org/https://doi.org/10.1016/j.jcp.2019.05.024.
32. Sun, L.; Gao, H.; Pan, S.; Wang, J.X. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering* **2020**, *361*, 112732. https://doi.org/https://doi.org/10.1016/j.cma.2019.112732.
33. Rueden, L.V.; Mayer, S.; Beckh, K.; Georgiev, B.; Giesselbach, S.; Heese, R.; Kirsch, B.; Pfrommer, J.; Pick, A.; Ramamurthy, R.; et al. Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems. *IEEE Transactions on Knowledge and Data Engineering* **2023**, *35*. https://doi.org/10.1109/TKDE.2021.3079836.
34. Yamazaki, Y.; Harandi, A.; Muramatsu, M.; Viardin, A.; Apel, M.; Brepols, T.; Reese, S.; Rezaei, S. A finite element-based physics-informed operator learning framework for spatiotemporal partial differential equations on arbitrary domains. *Engineering with Computers* **2025**, *41*, 1–29. https://doi.org/10.1007/s00366-024-02033-8.
35. Sunil, P.; Sills, R.B. FE-PINNs: finite-element-based physics-informed neural networks for surrogate modeling. *arXiv preprint arXiv:2412.07126* **2024**.

 doi:10.20944/preprints202504.0645.v1

36. Montavon, G.; Orr, G.; Müller, K.R. *Neural networks: Tricks of the Trade*; Vol. 7700, *Lecture Notes in Computer Science*, Springer Berlin, Heidelberg, 2012.
37. Goodfellow, I.; Bengio, Y.; Courville, A.; Bengio, Y. *Deep learning*; Vol. 1, *Adaptive Computation and Machine Learning*, MIT Press Cambridge, 2016.