Politecnico di Torino
Master's Degree in Mathematical Engineering

Model Order Reduction and Machine Learning
Nonlinear Elliptic Problem

Lucia Ghezzi s325951
Elisabetta Roviera s328422

Contents

1 Nonlinear Elliptic Problem

1.1 Problem definition and project objective

Problem definition Let us consider the two-dimensional spatial domain $\Omega = (0,1)^2$. We want to solve the following parametrized problem: given $\mu = (\mu_0, \mu_1) \in \mathcal{P} = [0.1, 1]^2$, find $u(\mu)$ such that

$$-\Delta u(\mu) + \frac{\mu_0}{\mu_1}(e^{\mu_1 u(\mu)} - 1) = g(x;\mu)$$

with homogeneous Dirichlet condition on the boundary, i.e. zero-boundary condition. We will refer to this problem as the Nonlinear Elliptic Problem (NEP). The problem setup is taken from *A graph convolutional autoencoder approach to model order reduction for parametrized PDEs* by Federico Pichi, Beatriz Moya, and Jan S. Hesthaven.

The source term g is defined as

1. For NEP1 :
$$g(x;\mu) = g_1 = 100\sin(2\pi x_0)\cos(2\pi x_1) \quad \forall x = (x_0, x_1) \in \Omega.$$
2. For NEP2:
$$g(x;\mu) = g_2 = 100\sin(2\pi\mu_0 x_0)\cos(2\pi\mu_0 x_1) \quad \forall x = (x_0, x_1) \in \Omega.$$

Project objective The aim of this project is to solve NEP with the two different forcing terms using three methods and to compare the results in terms of accuracy of the solution and computational cost. The three approaches used to solve the problem are: POD-Galerkin method over a Finite Element full order model (POD), parametric Physics Informed Neural Network (PINN) and Proper Orthogonal Decomposition - Neural Network method (POD-NN).

1.2 Exact solution test of the methods

Exact solution problem definition As a preliminary step before the solution of NEP1 and NEP2, we test the performance of the methods solving another elliptic problem, NEP0, of which we know the exact solution. This technique is used to assess what will be the expected behavior of the methods for the problems we are actually interested in; computing the error done by the methods with respect to the known exact solution, we can establish if we expect a good performance for the problems with unknown exact solution.

This is done starting from a random exact solution

$$\tilde{u}(x, y; \mu) = \mu_0 xy(1-x)(1-y)$$

and computing the forcing term $\tilde{g}(x, y; \mu)$ st $-\Delta\tilde{u}(\mu) + \frac{\mu_0}{\mu_1}(e^{\mu_1 \tilde{u}(\mu)} - 1) = \tilde{g}(x, y; \mu)$.
To do so, compute the Laplacian of $\tilde{u}$:

$$\Delta\tilde{u}(\mu) = -2\mu_0(y(1-y) + x(1-x))$$

and substitute $\tilde{u}$ in the non linear term of the PDE :

$$\frac{\mu_0}{\mu_1}(e^{\mu_1 \tilde{u}(\mu)} - 1) = \frac{\mu_0}{\mu_1}(e^{\mu_1 \mu_0 xy(1-x)(1-y)} - 1)$$

Then, the forcing term for NEP0 is

$$\tilde{g}(x; \mu) = g_0 = 2\mu_0(y(1-y) + x(1-x)) + \frac{\mu_0}{\mu_1}(e^{\mu_1 \mu_0 xy(1-x)(1-y)} - 1)$$

1.3 Weak formulation of the problem and Newton scheme

Passage to weak formulation Start from the general NEP with forcing term g and homogeneous Dirichlet conditions: given $\mu \in \mathcal{P}$, find $u(\mu) \in V$ such that

$$-\Delta u(\mu) + \frac{\mu_0}{\mu_1}(e^{\mu_1 u(\mu)} - 1) = g(x; \mu)$$

Integrating on the domain, multiplying times a general function $v \in V$ and recalling the boundary condition, we get the weak formulation of the problem: given $\mu \in \mathcal{P}$, find $u(\mu) \in V$ such that for every $v \in V$

$$\int_\Omega \nabla u \cdot \nabla v \, dx + \int_\Omega \frac{\mu_0}{\mu_1}(e^{\mu_1 u} - 1)v \, dx - \int_\Omega gv \, dx = 0$$

Take $V = H_0^1(\Omega)$ and define the operator $F : V \to V'$ such that for $u \in V$ and $v \in V$:

$$F(u)[v] = \int_\Omega \nabla u \cdot \nabla v \, dx + \int_\Omega \frac{\mu_0}{\mu_1}(e^{\mu_1 u} - 1)v \, dx - \int_\Omega gv \, dx$$

Then the weak formulation of the problem is equivalent to $F(u)[v] = 0$.

Newton method for non linear problems Since the equation is non-linear, to solve it we use Newton method, which is based on the linearization of the problem by means of the Taylor expansion of order 1. At every iteration k, we solve for δu the linear system

$$DF(u_k)[\delta u] = -F(u_k)$$

and update the current approximation of the solution u_k with the increment δu

$$u_{k+1} = u_k + \delta u$$

$DF(u_k)[\delta u]$ is the Fréchet Derivative of F along direction δu evaluated at u_k :

$$DF(u)[\delta u][v] = \lim_{\epsilon \to 0} \frac{F(u + \epsilon\delta u)[v] - F(u)[v]}{\epsilon}$$

Being F composed of three terms $F = F_1 + F_2 + F_3$, the derivatives can be computed separately:

- $F_1(u)[v] = \int_\Omega \nabla u \cdot \nabla v \, dx$ is linear in u, therefore:

$$DF_1(u)[\delta u][v] = F_1(\delta u)[v] = \int_\Omega \nabla \delta u \cdot \nabla v \, dx$$

- $F_2(u)[v] = \int_\Omega \frac{\mu_0}{\mu_1}(e^{\mu_1 u} - 1)v \, dx$ is non linear in u, therefore:

$$DF_2(u)[\delta u][v] = \lim_{\epsilon \to 0} \frac{F_2(u + \epsilon\delta u)[v] - F_2(u)[v]}{\epsilon} = \lim_{\epsilon \to 0} \int_\Omega \frac{\mu_0}{\mu_1} e^{\mu_1 u} \frac{(e^{\mu_1 \epsilon \delta u} - 1)}{\epsilon} v \, dx = \int_\Omega \mu_0 e^{\mu_1 u} \delta u v \, dx$$

- $F_3(u)[v] = -\int_\Omega gv \, dx$ does not depend on u, therefore :

$$DF_3(u)[\delta u][v] = 0$$

Then, at every iteration of the Newton algorithm, we solve for δu

$$\left(\int_\Omega \nabla \delta u \cdot \nabla v \, dx + \int_\Omega \mu_0 e^{\mu_1 u_k} \delta u v \, dx\right) \delta u = -\left(\int_\Omega \nabla u_k \cdot \nabla v \, dx + \int_\Omega \frac{\mu_0}{\mu_1}(e^{\mu_1 u_k} - 1)v \, dx - \int_\Omega gv \, dx\right)$$

and compute the new approximation of the solution $u_{k+1} = u_k + \delta u$.

Newton implementation details Concerning the practical implementation of the Newton algorithm, the stopping criterion is based on the relative residual error

$$rel\_residual = \frac{||\delta u||_{L^2}}{||u_{k+1}||_{L^2}} = \frac{||u_{k+1} - u_k||_{L^2}}{||u_{k+1}||_{L^2}}$$

when $rel\_residual < Newton\_tol$ or when the maximum number of iterations is reached, we stop the loop. In particular, we choose the following parameters:

- Newton tolerance on the relative residual error: $Newton\_tol = 1e^{-8}$.
- Maximum number of iterations: $max\_iter = 100$.

1.4 Preliminary domain analysis

Theoretical results on the mesh size value As a first step to solve the problem numerically, we conducted a mesh size analysis on the exact solution problem NEP0, in order to evaluate the error of the high fidelity (HF) approximation wrt the know exact solution. First, we evaluate how a decrement in the mesh size leads to a smaller error. From the theory we know that the approximation error should decrease exponentially with decreasing mesh size. In particular, denote $u_\delta \in V_\delta \subset V$ the high fidelity approximation of the exact solution $u \in V$, h the mesh size (i.e. the edge of the mesh polygon - triangle for us) and s the order of the mesh. Then, for the L^2 and H^1 errors we know :

$$Err_{L^2}(h) = Err_{L^2}(h_0)\left(\frac{h}{h_0}\right)^{s+1}$$

$$Err_{H^1}(h) = Err_{H^1}(h_0)\left(\frac{h}{h_0}\right)^{s}$$

where h_0 is the starting mesh size and $Err_{L^2}(h) = \frac{||u_\delta - u||_{L^2}}{||u||_{L^2}}$, i.e. the relative error of the approximation computed with mesh size h wrt the exact solution. Similarly, $Err_{H^1}(h) = \frac{||u_\delta - u||_{H^1}}{||u||_{H^1}}$.

Implementation of mesh size analysis We check if the expected behavior is also observed experimentally in order to assess the quality of the method implementation. In particular, fixing the mesh size h, for a sample of 100 parameters $\mu = (\mu_0, \mu_1) \in \mathcal{P}$, we compute the Newton approximation of the HF solution and we evaluate its error wrt the known exact solution. We then average the errors over the 100 parameter samples for that mesh size. We repeat this operation for decreasing values of the mesh size (starting from 0.1 and halving it at every iteration), until the order of 10^{-4}. The observed results (see Figure 1) reflect the expected theoretical behavior.

Figure 1: Evaluation of the decreasing behavior of the error with decreasing mesh size values : experimental results reflect those expecetd from the theoretical knowledge.

Moreover, analyzing the resulting mean errors of the high fidelity approximation wrt the mesh size values, we can asses which is the most suitable mesh size value.

Results We find the two most suitable mesh size values to be the greatest and smallest ones providing an error of at least 10^{-3}. For these values, average errors and run time per snapshot computation are reported in Table 1.

Table 1: Performance metrics for different mesh sizes

Metric	**Mesh Size = 0.00312**	**Mesh Size = 0.00019**
Average snapshot computation time (s)	0.5948	11.2261
Average relative error (L2 Norm)	0.0089	0.0005
Average relative error (H1 Norm)	0.0937	0.0224

The value $h = 0.00312$, demonstrates a significantly lower computational cost while still yielding acceptable accuracy, particularly concerning the L^2 error, while for $h = 0.00019$ we get a smaller error but need almost $\times 19$ more computational time. This suggests that for applications

where computational efficiency is a primary concern, the larger mesh size presents a favorable trade-off between performance and accuracy. Following this reasoning, we adopted the mesh size value $h = 0.00312$ for the discretization of the domain. Using this mesh size, the domain is discretized into 214 degrees of freedom (DOFs).

2 Solution of Nonlinear Elliptic Problem

2.1 Error computation

We evaluate the performance of the solutions computing the error wrt to the high fidelity solution, evaluated on the quadrature points of the mesh. In particular, denote the N_{qp} quadrature points $QP = \{(x_q, y_q), q = 1, ..., N_{qp}\}$ and their weights $W = \{w_q, q = 1, ..., N_{qp}\}$, and

- For the high fidelity solution, denote $u_\delta(\mu) \in \mathbb{R}^{N_\delta}$ the high fidelity snapshot, $u_{\delta,x}(\mu) \in \mathbb{R}^{N_\delta}$ its derivative wrt x, $u_{\delta,y}(\mu) \in \mathbb{R}^{N_\delta}$ its derivative wrt y, for the parameter $\mu \in \mathcal{P}$ evaluated on the quadrature points QP.

- For the solution approximation obtained with one of the methods (output of model for PINN, projection of the reduced solution in the high fidelity space V_δ for POD and POD-NN), denote $\tilde{u}(\mu) \in \mathbb{R}^{N_\delta}$, $\tilde{u}_x(\mu) \in \mathbb{R}^{N_\delta}$ its derivative wrt x, $\tilde{u}_y(\mu) \in \mathbb{R}^{N_\delta}$ its derivative wrt y, for the parameter $\mu \in \mathcal{P}$.

Then the L^2 and H^1 absolute errors of approximation wrt the high fidelity solution evaluated on quadrature points are defined as

$$||u_\delta - \tilde{u}||_{L^2,abs} = (\sum_{(x_q,y_q)\in QP} (u_\delta(x_q, y_q) - \tilde{u}(x_q, y_q))^2 w_q)^{\frac{1}{2}}$$

$$||u_\delta - \tilde{u}||_{H^1,abs} = (\sum_{(x_q,y_q)\in QP} [(u_\delta(x_q, y_q) - \tilde{u}(x_q, y_q))^2 + (u_{\delta,x}(x_q, y_q) - \tilde{u}_x(x_q, y_q))^2 + (u_{\delta,y}(x_q, y_q) - \tilde{u}_y(x_q, y_q))^2] w_q)^{\frac{1}{2}}$$

and the relative errors

$$||u_\delta - \tilde{u}||_{L^2,rel} = \frac{||u_\delta - \tilde{u}||_{L^2,abs}}{(\sum_{(x_q,y_q)\in QP} u_\delta(x_q, y_q)^2 w_q)^{\frac{1}{2}}}$$

$$||u_\delta - \tilde{u}||_{H^1,rel} = \frac{||u_\delta - \tilde{u}||_{H^1 abs}}{(\sum_{(x_q,y_q)\in QP} [u_\delta(x_q, y_q)^2 + u_{\delta,x}(x_q, y_q)^2 + u_{\delta,y}(x_q, y_q)^2] w_q)^{\frac{1}{2}}}$$

2.2 Methods evaluation for exact solution problem

To assess the performance of the methods used to solve NEP1 and NEP2, we first use them to solve NEP0, for which we know the exact solution and thus are able to compute the real accuracy of the solutions obtained with the methods.

Results In Table 2 we report the errors of the high fidelity solutions, of the reduced solutions projected in the high dimensional space obtained with POD and with POD-NN, and the solution output of PINN. All method solutions are compared to the exact solution and are evaluated on the quadrature points (see error definition in Section 2.1). To have a better understanding of the performances of the methods on NEP1 and NEP2, we also report the error wrt the high fidelity snapshots. Since the problem is non linear, the solutions are computed using the Newton schema, presented in Section 1.3. The implemetation details can be found, respectively, in Sections 2.3, 2.5, 2.4.

Table 2: Solution of NEP0 and comparison of the different methods with focus on accuracy and compuattaional cost

NEP0 summary table		High fidelity	POD (N=1)	PINN	PODNN
Error w.r.t. exact solution	L2 relative	8.9364×10^{-3}	8.9355×10^{-3}	2.2172×10^{-3}	8.7427×10^{-3}
	H1 relative	9.1437×10^{-2}	9.1437×10^{-2}	2.5999×10^{-3}	9.1441×10^{-2}
Error w.r.t. HF solution	L2 relative	-	7.8979×10^{-6}	1.0230×10^{-2}	5.5511×10^{-4}
	H1 relative	-	1.6823×10^{-5}	9.2040×10^{-2}	5.5593×10^{-4}
Execution time	Average evaluation time (s)	7.17×10^{-3}	3.774×10^{-4}	1.0783×10^{-3}	2.19×10^{-4}
	Average speed-up wrt HF	-	19.541	7.003	36.467
Training loop	Number of training iterations	-	-	5	17800
	Number of switch L-BFGS optimizer	-	-	4	-
	Training time on CPU (s)	-	-	4.55	18.822

Observations

- **Error wrt the exact solution**: the high fidelity, POD and POD-NN approximations provide very similar results of the order of 10^{-3} for the L^2 error and of 10^{-2} for the H^1 error. This is explained by the simplicity of the exact solution, which is smooth and low-dimensional; even a reduced basis of size $N = 1$ suffices to capture its behavior accurately. Interestingly, the PINN method achieves the best accuracy in this case, with errors of order 10^{-3} in both norms, outperforming the other methods despite its unsupervised nature.

- **Error wrt the high fidelity solution**: POD errors are extremely small ($\approx 10^{-6}$ to 10^{-5}), as we expected from the very low dimension of the reduced space that hints a very simple task. POD-NN also provides accurate approximations ($\approx 10^{-4}$), though it is not quite as precise as POD. PINN shows larger discrepancies ($\approx 10^{-2}$ to 10^{-1}), which confirms that its solution differs more significantly from the high-fidelity reference than the reduced-order approaches. Thus, we can deduce that the trend of the performances wrt high fidelity reflects the behavior of the methods wrt the exact solution.

- **Computational time**: POD-NN has the fastest evaluation, followed by POD, as expected from MOR techniques. PINN, despite a slower evaluation, is much faster to train (4.55 s vs 18.8 s) and requires very few iterations. This confirms that PINNs can be efficient when the problem is simple.

2.3 Proper Orthogonal Decomposition

Proper Orthogonal Decomposition (POD) is a model reduction technique that identifies the most energetic modes of a system from high-fidelity simulation data. By extracting optimal basis functions through a snapshot analysis, POD enables the projection of the original high-dimensional problem onto a low-dimensional subspace. This significantly reduces the computational cost while preserving the dominant dynamics of the system.

Implementation details of POD We compute $N_\delta = 100$ high fidelity snapshots, relative to a random sample of parameters $\mathcal{P}_\delta = \{\mu_i, i = 1, ..., 100\} \subset \mathcal{P}$. Snapshots are computed using Newton method (see details in Section 1.3). We choose as tolerance for the minimum relative retained energy for the reduced space dimension $rel\_energy\_tol = 1 - 10^{-7}$.

Results Concerning the computation of the high fidelity snapshots, the $Newton\_tol = 1 \times 10^{-8}$ is reached in a very few iterations (max number of iterations before stopping is 6 for NEP1 and 9 for NEP2, and it never stops for max_iter stopping criterion), thus we compute high fidelity solutions of satisfying accuracy. The eigenvalues of the covariance matrix of the snapshot matrix are then used to find the highest dimension N for the reduced space $V_N \subset V_\delta$ that captures at least $rel\_energy\_tol = 1-10^{-7}$ energy. For NEP1 the so found reduced dimension is $N = 3$, while for NEP2 we obtain $N = 9$. The N eigenvectors associated to the N largest eigenvalues are then used to construct the change of basis matrix that will be used to go from the low fidelity space V_N to the high fidelity space V_δ. Average relative errors (as defined in Section 2.1) of the low fidelity solutions projected in the full dimensional space (both in L^2 and H^1) wrt the high fidelity snapshots are reported in Table 3. The table also shows the average computational time necessary to compute one fom and rom solution: the considered time is the cumulated time necessary to solve the Newton linear systems for δu - respectively in the N_δ and N dimensional spaces - until the stopping criterion is satisfied, for each parameter. Finally, the $\mu-$averaged fom-rom speed-up is reported, where $speed\_up(\mu_i) = \frac{time\_fom(\mu_i)}{time\_rom(\mu_i)}$ for every $\mu_i \in \mathcal{P}_\delta$.

Table 3: Comparison of errors of POD solution with respect to the high fidelity solution for NEP1 and NEP2

POD results evaluation	Metric	NEP1 (N=3)	NEP2 (N=9)
Error w.r.t. HF solution	L2 relative	2.772×10^{-5}	5.1432×10^{-4}
	H1 relative	3.0695×10^{-5}	9.8641×10^{-4}
Execution time	Average evaluation time FOM (s)	1.190×10^{-1}	1.190×10^{-1}
	Average evaluation time POD (s)	8.035×10^{-4}	5.5382×10^{-4}
	Average speed-up	15.657	22.327

Observations

- **Accuracy of POD**: the method yields extremely low errors with respect to the high-fidelity solution. For NEP1, using a reduced basis of size $N = 3$, the relative L^2 and H^1 errors are of the order 10^{-5}. For NEP2, which is more challenging due to its parametric forcing term, the reduced basis is larger ($N = 9$) and the errors are slightly higher but still very low ($\approx 10^{-4}$). This increase in the error is consistent with the expectations, given the added complexity of the parametric problem.

- **Computational efficiency of POD**: POD achieves substantial speed-ups over the full-order model. The average evaluation time is reduced from $\approx 10^{-1}$ to 10^{-4} for both forcing terms. This corresponds to speed-ups of approximately $\times 15$ and $\times 22$. Notably, despite the higher dimensionality required for NEP2, the POD method still delivers both high accuracy and greater computational gains, highlighting its effectiveness even in more complex parametric settings.

2.4 Physical-Informed Neural Network

Physics-Informed Neural Networks (PINN) are neural networks that incorporate physical laws (via differential equations) directly into their loss function, allowing them to solve problems in an unsupervised manner, without labeled data.

Architectural variants tested During the experiments, numerous variants of the PINN are implemented with the aim of comparing the performance resulting from different architectures and activation functions. The models tested are listed below.

- **PINN40 / PINN60 / PINN80 / PINN128**. Several PINN are configured with a fully connected architecture and a tanh activation function. These configurations vary in the number of neurons in the four hidden layers (40, 60, 80, and 128, respectively), allowing us to analyze the influence of network size on approximation capability.

- **RELU60**. Network with a structure similar to PINN60, but with a `ReLU` activation function. This choice allows the behavior of a non-smooth activation function to be evaluated.

- **RESNET**. Architecture inspired by ResNet models, with skip connections between layers. This approach is known to improve gradient propagation and potentially accelerate convergence in deep models.

- **PINNTanhDeep**. Variant of PINN40 with seven hidden layers.

- **SILU60**. Network with 60 neurons per layer and `SiLU` activation function, a smooth, self-regulating function that has shown good performance in various learning contexts.

- **SINE60**. Architecture with sinusoidal activation function, based on the SIREN (Sinusoidal Representation Networks) approach, particularly effective in modeling high-frequency functions.

- **ELU60**. Network with `ELU` (Exponential Linear Unit) activation function, which improves gradient stability in the early stages of training.

- **PINNLN60**. Version with layer normalization, for stabilizing training and gradient explosion/vanishing problems.

- **MIXACT**. Network that employs a combination of activation functions within the same model, with the aim of exploiting the complementary characteristics of the different activations.

- **DeepResPINN**. A deeper residual PINN architecture combining the benefits of deep neural networks with residual connections, designed to better capture complex solution structures while alleviating vanishing gradient issues.

- **FourierPINN**. Network incorporating Fourier feature embeddings in the input layer, which helps in representing functions with high-frequency components and improves learning of oscillatory solutions.

- **PINNHardBC**. It is a variant of PINN that exactly satisfies the boundary conditions through a multiplicative ansatz, reducing the space of admissible functions for greater efficiency and accuracy.

Table 4: Performance Comparison of Different PINN Architecture on GPU

Architecture	Relative L^2 Error	Relative H^1 Error	Avg. Epoch Time (s)	Epochs Used	Tot Time (s)
PINN40	3.8105×10^{-1}	4.8245×10^{-1}	0.0153	18275	279.98
PINN60	$\mathbf{4.8981 \times 10^{-2}}$	$\mathbf{2.2007 \times 10^{-1}}$	0.0165	10588	**174.70**
PINN80	4.9559×10^{-2}	2.2051×10^{-1}	0.0875	11197	979.74
PINN128	3.4705×10^{-1}	4.4672×10^{-1}	0.0575	10174	584.91
RELU60	1.0216×10^{0}	1.0000×10^{0}	0.0113	7501	84.76
RESNET	3.5844×10^{-1}	4.8224×10^{-1}	0.0223	39093	872.08
PINNTanhDeep	3.1378×10^{-1}	3.7665×10^{-1}	0.0301	18356	552.45
SILU60	3.5232×10^{-1}	4.8420×10^{-1}	0.0199	17364	345.54
SINE60	3.7523×10^{-1}	5.2273×10^{-1}	0.0141	50000	705.00
ELU60	3.1729×10^{-1}	3.5060×10^{-1}	0.0154	49788	766.74
PINNLN60	3.5723×10^{-1}	4.7960×10^{-1}	0.0411	19173	787.97
MIXACT	1.2343×10^{0}	9.9775×10^{-1}	0.0229	19253	440.99
DeepResPINN	3.7505×10^{-1}	4.7178×10^{-1}	0.0196	12048	236.14
FourierPINN	1.0006×10^{0}	1.0000×10^{0}	0.0161	19576	315.17
PINNHardBC	$\mathbf{4.1897 \times 10^{-2}}$	$\mathbf{2.1803 \times 10^{-1}}$	0.0273	11875	**324.19**

The numerical results in Table 4 clearly show that PINNHardBC achieved the best performance in terms of both relative L^2 and H^1 errors, indicating a significant advantage from hard-enforced boundary conditions and the physics-informed formulation. While PINN60 has a slightly higher relative error, it converges faster and requires the least total training time among the most accurate models, making it preferable for time-sensitive applications where a favorable trade-off between efficiency and precision is desired. PINNHardBC remains the optimal choice for maximum accuracy.

Detailed description of the PINNHardBC architecture The PINNHardBC network precisely enforces boundary conditions through a multiplicative ansatz. This method significantly boosts efficiency and accuracy by restricting the function space during optimization.
As illustrated in Figure 2, the model is a fully connected feedforward neural network characterized by:

- **Input Layer (4 neurons)**: Receives a spatial point (x_1, x_2) and two parameters μ_0, μ_1.

- **Hidden Layers**: Comprises three hidden layers, each with 60 neurons, utilizing a Tanh activation function.

- **Output Layer (1 neuron)**: Produces a single scalar output representing the approximated solution function value $\hat{u}(\mathbf{x}, \mu)$.

The network's output is scaled by a **boundary factor** to analytically enforce Dirichlet boundary conditions. This factor is defined as:

$$\text{boundary\_factor}(\mathbf{x}) = x_1(1 - x_1)x_2(1 - x_2)$$

This function evaluates to zero along all boundaries of the square computational domain $[0, 1]^2$. Consequently, the overall network approximates a solution $u(\mathbf{x}, \mu)$ of the form:

$$u(\mathbf{x}, \mu) = \underbrace{x_1(1 - x_1)x_2(1 - x_2)}_{\text{Boundary factor}} \cdot \underbrace{\mathcal{N}(\mathbf{x}, \mu; \theta)}_{\text{Neural network}}$$

where $\mathcal{N}(\mathbf{x}, \mu; \theta)$ represents the parameterized neural network. This construction inherently ensures that the approximate solution satisfies homogeneous Dirichlet conditions on the domain boundary.

The hard imposition of boundary conditions offers several advantages: it enhances accuracy by focusing optimization exclusively on the differential equation within the domain; it improves solution stability near boundaries by mitigating error oscillations; and it boosts computational efficiency by restricting the space of admissible functions, leading to faster convergence.

Training loop The training aims to minimize the PDE residual within the domain while ensuring exact satisfaction of boundary conditions through a physics-informed approach. The total loss function, MSE, is composed of two primary terms: a penalization for the PDE residual and an error term for boundary conditions. Although the latter is inherently zero due to the network's hard enforcement of boundary conditions, it is continuously monitored for verification.

The adopted loss function is defined as: $\text{MSE} = \text{MSE}_b^\mu + \lambda \, \text{MSE}_p^\mu$, where λ is a hyperparameter balancing the contributions. The weight λ in the loss function has initially been varied to explore its impact on the solution quality. However, for the final selected network architecture, which enforces boundary conditions via a hard constraint (Hard Boundary Condition), the boundary loss term MSE_b remains identically zero throughout the training. As a result, the value of λ becomes irrelevant in this configuration. The boundary loss term, MSE_b^μ, and the physical loss term, MSE_p^μ, are defined, respectively, as:

$$\text{MSE}_b^\mu = \frac{1}{N_b} \sum_{k=1}^{N_b} \left| \tilde{w}(x_k^b, \mu_k^b) - u(x_k^b, \mu_k^b) \right|^2, \quad (x_k^b, \mu_k^b) \in \partial\Omega \times \mathcal{P} \qquad \text{MSE}_p^\mu = \frac{1}{N_p} \sum_{k=1}^{N_p} \left| R(\tilde{w}(x_k^p, \mu_k^p)) \right|^2, \quad (x_k^p, \mu_k^p) \in \Omega \times \mathcal{P}$$

with $R(\cdot)$ denoting the residual operator of the PDE. The boundary loss, ideally zero, is still monitored during training. A maximum of 50000 epochs is allowed, but early stopping is used to avoid overfitting, excessive computations, or numerical issues. Total, PDE, and boundary losses are tracked throughout to ensure stability and guide potential manual adjustments.

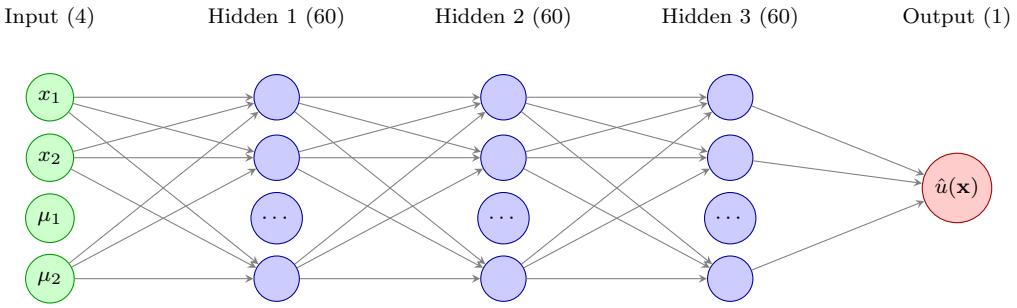


Figure 2: PINNHardBC Architecture Diagram.

Initially, during the first epoch, placement points are sampled uniformly at random across both the internal domain $\Omega = [0, 1]^2$ and its boundary. From the second epoch onward, an adaptive sampling strategy is implemented. This strategy selects points with the highest residual from a larger candidate set, thereby concentrating optimization efforts on regions where the network exhibits the largest errors. This approach enhances solution quality and learning efficiency through more targeted training.

The optimization process employs a hybrid strategy, beginning with the Adam algorithm and transitioning to L-BFGS. Initially, the Adam optimizer is utilized with a learning rate of 10^{-3}. This phase aims to rapidly reduce the loss function. A crucial aspect of this strategy is the criterion for switching optimizers. For NEP1 the transition occurred when both the losses fell below 10^{-3}. However, for the NEP2 model, this threshold proved problematic; the loss function would often stabilize without reaching this stringent criterion, preventing the switch to the L-BFGS optimizer and consequently hindering further improvement. To address this, the transition to the L-BFGS optimizer occurs when the total loss value drops below 5×10^{-1}. The purpose of this optimizer switch is to leverage the strengths of each algorithm. While Adam is effective for initial rapid convergence, L-BFGS is known for its ability to perform highly precise, fine-tuned optimization, particularly beneficial once the loss has plateaued with Adam.

Results Following an initial GPU-based study for rapid architectural comparison and selection, we retrain the chosen PINN architecture on a CPU. This re-execution is crucial for a comprehensive performance evaluation, allowing for direct comparison of results not only in terms of accuracy but also of computational execution times. The outcomes of this CPU-based training are presented in Table 5, which provides a comparative analysis of PINN solution errors and training efficiencies for NEP1 and NEP2. In particular, NEP2 shows lower relative L^2 and H^1 errors, achieving better precision compared to NEP1, although it requires a greater number of epochs and total execution time.

Table 5: Comparison of errors of PINN solution with respect to the high fidelity solution for NEP1 and NEP2 on CPU

PINN results evaluation	Metric	NEP1	NEP2
Error w.r.t. HF solution	L2 relative	4.1872×10^{-2}	2.4579×10^{-2}
	H1 relative	2.1803×10^{-1}	1.5484×10^{-1}
Train phase	Total Number of epochs	10689	17415
	L-BFGS starting epoch	10685	17412
	Final loss	7.88×10^{-4}	6.24×10^{-2}
	Total execution time (s)	718.11	1068.91
Evaluation phase	Average evaluation time FOM (s)	1.190×10^{-1}	1.190×10^{-1}
	Average evaluation time PINN (s)	1.0967×10^{-3}	1.1493×10^{-3}
	Average speed up	8.4240	10.0273

Observations

- **Accuracy of PINN**: The PINN method yields relative errors of the order of 10^{-2} in norm L^2 and 10^{-1} in norm H^1, which can be considered reasonable given the unsupervised nature of the approach. While the final loss values are relatively small ($\approx 10^{-4}$ for NEP1 and $\approx 10^{-2}$ for NEP2), indicating successful optimization of the training objective, the resulting solution errors highlight a common limitation: low training loss does not necessarily guarantee high accuracy in the PDE solution, especially in more sensitive norms like H^1.

- **Computational efficiency of PINN**: In terms of training, PINNs require a substantial number of epochs (over 10^4) and considerable execution time (from $\approx$ 12 to 18 minutes on CPU). Despite the high training cost, the evaluation phase is fast, with inference times on the order of 10^{-3}, yielding a speed-up of about 8 to 10 compared to the full order model.

2.5 POD-based Neural Network

The POD-based Neural Network (POD-NN) method combines the dimensionality reduction of Proper Orthogonal Decomposition with the approximation capabilities of neural networks. By learning the mapping from parameters to reduced coefficients, POD-NN enables fast and accurate reconstruction of the solution with significantly reduced computational effort.

Implementation details of POD-NN The neural network employed in this work is a fully connected feedforward network designed to map a parametric input $\mu \in \mathcal{P}$ to a reduced-order solution, approximation of the one obtained with POD, $u_N \in \mathbb{R}^N$ (with $N = 3$ for NEP1 and $N = 9$ for NEP2). The net is composed of 4 fully connected layers, each with 40 neurons. Each hidden layer uses the hyperbolic tangent (tanh) activation function, which maps values to the range $[-1, 1]$ and provides smooth nonlinear transformation. Concerning the training set up, we use as loss function the Mean Squared Error (MSE) loss and as optimizer Adam with a learning rate of $lr = 0.001$; The learning rate is manually decayed from 0.001 to 0.0001 after 50.000 epochs for NEP1 and after 100.000 epochs for NEP2, to allow finer updates and improve convergence stability during the later stages of training. We set the maximum number of epochs for the training loop to $max\_epochs = 200.000$ for NEP1 and to $max\_epochs = 500.000$ for NEP2, and the training tolerance to $loss\_tol = 10^{-6}$: we stop the training epoch when either the loss goes below the tolerance or the maximum number of iterations is reached. We train the network using as input the set $\mathcal{P}_{train} = \{\mu_i\}_{i=1}^{100}$ of parameters used to find the high fidelity snapshots (see Section 2.3) and as target the set of high fidelity solutions projected in the reduced space $\mathcal{S}^N_{train} = [u_N(\mu_i)]_{\mu_i \in \mathcal{P}_{train}} \subset \mathbb{R}^N$.

Results The trained net is tested on the set of test parameters $\mathcal{P}_{test} = \{\mu_i\}_{i=1}^{100}$ used to compute the high and low fidelity snapshots to evaluate POD method in Section 2.3. The average errors (as defined in Section 2.1) of the net predictions projected in the fom space wrt the high fidelity solutions (in L^2 and H^1) are reported in Table 6, together with the average execution time of a fom and rom solution computation and with the average fom-rom speed up of PODNN.

Table 6: Comparison of errors of POD-NN solution with respect to the high fidelity solution for NEP1 and NEP2

POD-NN results evaluation	Metric	NEP1	NEP2
Error w.r.t. HF solution	L2 relative	6.0497×10^{-4}	2.8012×10^{-2}
	H1 relative	6.0436×10^{-4}	2.5705×10^{-2}
Execution time	Average evaluation time FOM (s)	1.190×10^{-1}	1.190×10^{-1}
	Average evaluation time POD-NN (s)	2.181×10^{-4}	1.8587×10^{-4}
	Average speed-up	68.617	70.8458
Training loop	Number of training iterations	119274	500000
	Training time (s)	133.364	570.4065 (*)

(*) the loss never goes below the tolerance of 10^{-6}, but setting it to 10^{-5} the train stops after 112474 epochs and the training time (s) is 120.737.

Observations

- **Accuracy of POD-NN** : The POD-NN method delivers highly accurate results for NEP1, with relative errors around 10^{-4}. For the more complex NEP2 problem, the accuracy decreases, with errors of $\approx 10^{-2}$, still remaining within an acceptable range for reduced-order modeling.

- **Computational efficiency of POD-NN**: On the one hand, the evaluation times are extremely low ($\approx 10^{-4}$) resulting in significant speed-ups compared to the full order model: the evaluation is around $\times 70$ faster than the high fidelity snapshot computation. On the other hand, training POD-NN requires a considerable number of iterations, particularly for NEP2, where the default training loop runs for 500000 epochs and the loss does not drop below the default tolerance of 10^{-6}. Relaxing the the stopping criterion to 10^{-5}, the training stops earlier at 112474 epochs, considerably reducing the training time.

3 Comparison of Methods: accuracy vs computational cost

In this section we will compare the performance of the methods for every problem. In order to make the comparison meaningful, we adopt the following strategy:

- We take a random "training set" of parameters $\mathcal{P}_{train} = \{\mu_i\}_{i=1}^{100}$, of which we find the high fidelity snapshots matrix $\mathcal{S}^\delta_{train} = [u_\delta(\mu_i)]_{\mu_i \in \mathcal{P}_{train}}$, that we use to derive the POD basis and to train the POD-NN in a supervised way wrt the high fidelity snapshots.

- We take a "test set" of parameters $\mathcal{P}_{test} = \{\mu_i\}_{i=1}^{100}$, that we use to evaluate the performances : first, we compute the high fidelity snapshot matrix relative to $\mathcal{P}_{test}$, $\mathcal{S}^\delta_{test} = [u_\delta(\mu_i)]_{\mu_i \in \mathcal{P}_{test}}$, then we evaluate the performances of POD, PINN and POD-NN on $\mathcal{P}_{test}$ and compute the error wrt the test high fidelity snapshots on the weighted quadrature points.

3.1 NEP1

In Table 7 is reported a summary of the results obtained with the different methods.

Observations

- **POD** : POD yields the most accurate results, with relative errors around 10^{-5} in both L^2 and H^1 norms. It requires no training and offers a moderate evaluation speed ($\approx 10^{-4}$), achieving a speed-up of about $\times 15.7$ compared to the full-order model.

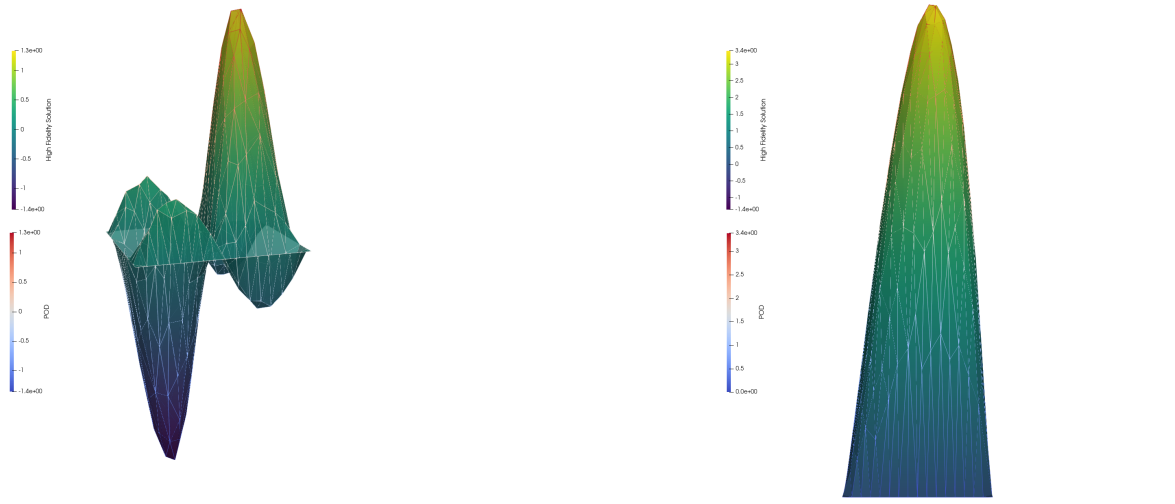
- **PINN** : PINN, although less accurate—with relative errors of the order 10^{-2} to 10^{-1}- has the important advantage of being an unsupervised method, as it does not rely on precomputed high fidelity snapshots for training. This makes it particularly appealing in scenarios where generating such data is costly or infeasible. The supervised reduced-order approaches clearly outperform PINN in both accuracy and efficiency.

- **POD-NN** : POD-NN provides a favorable trade-off between accuracy and computational efficiency. While its errors are slightly higher than POD (on the order of 10^{-4}), it offers a significantly faster evaluation time (around 1/4 of POD time) and a substantial speed-up of nearly $\times 69$ HF solutions. This comes at the cost of a brief training phase of about 2 minutes, which is remarkably fast given the accuracy achieved. Once trained, POD-NN enables highly efficient evaluations.

Table 7: Accuracy vs computational time for NEP1 : comparison of the different methods

NEP1 summary table		POD (N=3)	PINN	PODNN
Error w.r.t. HF solution	L2 relative	2.772×10^{-5}	4.1872×10^{-2}	6.0497×10^{-4}
	H1 relative	3.0695×10^{-5}	2.1803×10^{-1}	6.0436×10^{-4}
Execution time	Average evaluation time (s)	8.035×10^{-4}	1.0967×10^{-3}	2.181×10^{-4}
	Average speed-up wrt HF	15.657	8.4240	68.617
Training loop	Number of training iterations	-	10689	119274
	Training time (s)	-	718.11	133.364

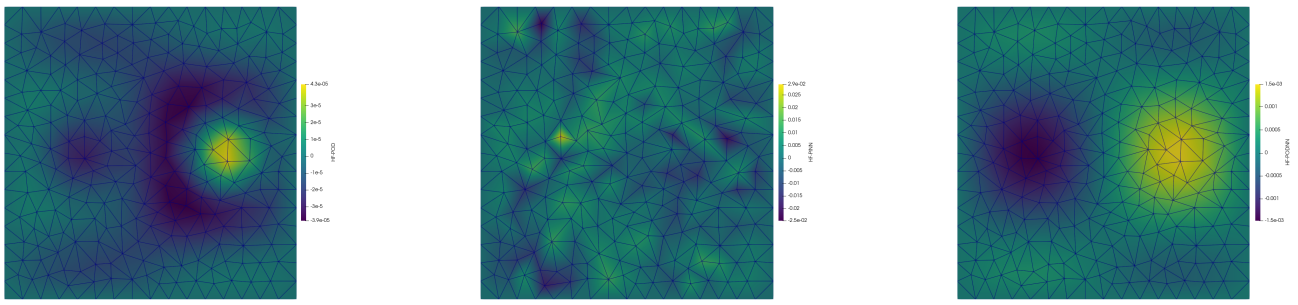
Visual comparison Figure 3 shows a visual comparison between the high fidelity solutions and the reduced order approximations obtained via POD for the two test cases NEP1 and NEP2 with $\mu = (\mu_0, \mu_1) = (0.2, 0.8)$. The HF solution is rendered with a smooth surface using a `viridis` colormap, while the POD approximation is overlaid in wireframe mode. In both cases, the POD solution closely follows the overall shape and magnitude of the HF solution, with minor discrepancies visible particularly in NEP1, where the solution exhibits sharper variations. In Figure 4, the spatial distribution of pointwise absolute errors with respect to the HF solution is shown for NEP1. Subplot (a) shows the HF–POD difference, where the errors are concentrated in localized regions. Subplot (b) reports the HF–PINN difference, showing wider zones of higher residuals, consistent with the higher global error observed in the metrics. Subplot (c) displays the HF–PODNN difference, which remains spatially contained and generally lower than for PINN, confirming its better accuracy on this test case. The underlying mesh is also visible in all three plots, highlighting the spatial resolution and distribution of the degrees of freedom (DoFs) used in the simulations.



(a) High Fidelity Solution and POD solution for NEP1

(b) High Fidelity Solution and POD solution for NEP2

Figure 3: Comparison of High Fidelity and POD solutions for two different forcing terms.



(a) Differences HF and POD solution

(b) Differences HF and PINN solution

(c) Differences HF and POD-NN solution

Figure 4: Differences between High Fidelity Solution and (a) POD, (b) PINN, e (c) POD-NN for NEP1.

3.2 NEP2

In Table 8 is reported a summary of the results obtained with the different methods.

Table 8: Accuracy vs computational time for NEP2 : comparison of the different methods

NEP2 summary table		POD (N=9)	PINN	PODNN
Error w.r.t. HF solution	L2 relative	5.1432×10^{-4}	2.4579×10^{-2}	2.8012×10^{-2}
	H1 relative	9.8641×10^{-4}	1.5484×10^{-1}	2.5705×10^{-2}
Execution time	Average evaluation time (s)	5.5382×10^{-4}	1.1493×10^{-3}	1.8587×10^{-4}
	Average speed-up wrt HF	22.327	10.0273	70.8458
Training loop	Number of training iterations	-	17415	500000
	Training time (s)	-	1068.91	570.4065

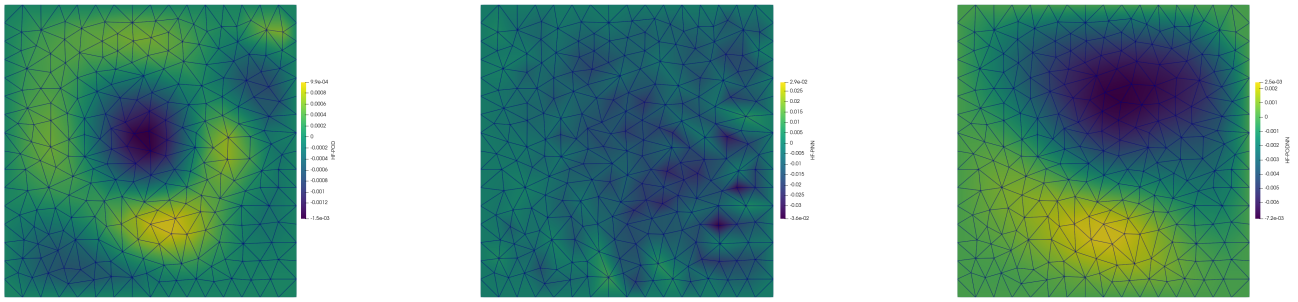
Observations

- **POD** : In the more complex NEP2 scenario, where a parametric forcing term increases the problem's variability, POD remains the most accurate method, yielding relative errors on the order of 10^{-4}, with very low evaluation time ($\approx 5 \times 10^{-4}$ s) and a solid speed-up of over $\times 22$. As it does not require any training, it is particularly efficient when the reduced basis space is expressive enough.

- **PINN** : PINN, being unsupervised, yields higher errors (10^{-2} in L^2, 10^{-1} in H^1), confirming its lower accuracy wrt POD-based techniques. It requires over 17,000 iterations and approximately 1069 seconds of training on CPU. While its inference time is reasonably low ($\approx 10^{-3}$ s), it remains slower than POD and POD-NN, and the overall speed-up ($\times 10$) is comparatively limited. These results highlight the trade-off between flexibility and performance in PINN-based solvers.

- **POD-NN** : POD-NN shows a moderate degradation in accuracy compared to POD, with relative errors in the range of 10^{-2}. Nonetheless, it achieves the fastest inference time ($\approx 2 \times 10^{-4}$ s) and the highest speed-up ($\times 70$). This performance comes at the cost of a longer training phase (around 9.5 minutes for 500,000 iterations), though early stopping strategies could reduce the training burden.

Visual comparison In Figure 5, the spatial distribution of pointwise absolute errors with respect to the high-fidelity solution is shown for NEP2. The underlying mesh is visible in all subplots, helping to localize the regions where the models deviate most from the HF reference. Subplot (a) displays the HF-POD error, which remains very low and concentrated, consistently with the extremely small global errors observed in the metrics. Subplot (b) shows the HF-PINN difference, where more irregular and spatially widespread errors appear, reflecting the higher values in both L^2 and especially H^1 norms. This nearly uniform distribution of the error across the domain is a consequence of the adaptive sampling strategy, which guided the training to focus on regions with higher residuals, allowing the model to balance the approximation quality over the whole domain. Subplot (c) presents the HF-PODNN residuals, which are more structured and confined than those of the PINN, confirming the better quantitative performance of PODNN over PINN for this parametric case.



(a) Differences HF and POD solution (b) Differences HF and PINN solution (c) Differences HF and POD-NN solution

Figure 5: Differences between High Fidelity Solution and (a) POD, (b) PINN, e (c) POD-NN for NEP2.

4 Conclusions

In this project, we conduct a comparative study of three model reduction and learning-based approaches—POD, POD-NN, and PINN—applied to two nonlinear parametric PDE problems, NEP1 and NEP2. The investigation begins by selecting an appropriate mesh size to ensure accurate and efficient discretization. The methods are first validated on a benchmark problem, NEP0, with a known exact solution, allowing for a meaningful assessment of their accuracy and behavior.
After this preliminary step, the methods are applied to the target problems NEP1 and NEP2. For each approach, we evaluate accuracy, computational efficiency, and training requirements, followed by a detailed cross-method comparison.
Our findings show that POD consistently provides the highest accuracy and lowest inference time, thanks to its use of high fidelity snapshots and reduced basis projection. POD-NN strikes a good balance between accuracy and efficiency, achieving near-POD performance with extremely fast evaluations after a moderate offline training phase. PINNs, while generally less accurate, offer the advantage of being fully unsupervised, solving the problem directly from the governing equations without requiring labeled data. This makes them appealing in data-scarce settings, despite longer training times and higher sensitivity to problem complexity.
Overall, this study highlights the strengths and trade-offs of each method, offering guidance on their applicability based on accuracy needs, computational constraints and data availability.
The code to reproduce all results is available at `https://github.com/elisabettaroviera/MOR-Project`.