ESAIM: PROCEEDINGS AND SURVEYS, June 2025, Vol. 81, p. 33-54
Didier Auroux, Martin Campos Pinto, Bruno Després, Victorita Dolean, Stéphane Lanteri, Victor Michel-Dansac

BENCHMARKING PDE PROBLEMS VIA MACHINE LEARNING FOR ENHANCED COMPUTATIONAL MODELING

Bahae-Eddine Madir[1] and Antoine Lechevallier[2, 3]

Abstract. The rapid advancement of mathematical modeling and scientific computing has become instrumental in addressing complex challenges across diverse fields. This proposed project seeks to leverage the interdisciplinary platform provided by the CEMRACS (Centre d'Été Mathématique de Recherche Avancée en Calcul Scientifique) summer school to contribute to the cutting-edge landscape of mathematical research and its practical applications.
In the context of the 2023 CEMRACS summer school, this project aims to delve into scientific machine learning. By synergizing insights from partial differential equations (PDE) and state of the art machine learning techniques, our project seeks to provide a benchmark for some PDE problems using multiple learning techniques. We begin by considering the conventional Poisson equation and subsequently adapt it to cater to geosciences scenarios like reservoir simulation.
We would like to extend our sincere gratitude to IFP Energies Nouvelles for their sponsorship, and to Thibault Faney for his supervising which has played a pivotal role in enabling the realization of this project.

Résumé. L'avancement rapide de la modélisation mathématique et du calcul scientifique est devenu essentiel pour relever des défis complexes dans de nombreux domaines. Ce projet propose de tirer parti de la plateforme interdisciplinaire offerte par l'école d'été CEMRACS (Centre d'Été Mathématique de Recherche Avancée en Calcul Scientifique) afin de contribuer au paysage de pointe de la recherche mathématique et de ses applications pratiques.
Dans le cadre de l'école d'été CEMRACS 2023, ce projet vise à explorer et approfondir les connaissances en apprentissage automatique appliquées au calcul scientifique (scientifc machine learning). En synergisant les idées issues des équations aux dérivées partielles (EDP) et des techniques de pointe en apprentissage automatique, notre projet cherche à fournir une référence pour certains problèmes d'EDP en utilisant plusieurs techniques d'apprentissage. Nous commençons par examiner la traditionnelle équation de Poisson pour ensuite l'adapter à des problèmes rencontrés en géosciences tels que la simulation de réservoir.
Nous tenons à exprimer notre sincère gratitude à IFP Energies Nouvelles pour leur parrainage, ainsi qu'à Thibault Faney pour sa supervision, qui a joué un rôle crucial dans la réalisation de ce projet.

Introduction

The use of neural networks has recently garnered growing interest in various scientific fields, such as engineering, physics, and finance. Traditional numerical methods used for solving partial differential equations

[1] Laboratoire de Mathématiques Raphaël Salem, France. e-mail: bahae-eddine.madir@univ-rouen.fr
[2] IFP Energies Nouvelles. e-mail: anle@norceresearch.no
[3] LJLL, France.

Article published online by EDP Sciences and available at https://www.esaim-proc.org or https://doi.org/10.1051/proc/202581033

(PDEs) often face challenges when dealing with high-dimensional problems and complex geometries, leading to high computational costs. In contrast, neural networks offer a promising alternative due to their flexibility, ability to learn complex patterns, and grid-free nature, enabling efficient approximation of solutions. Multi-Layer Perceptron neural networks are commonly used for machine learning tasks such as pattern recognition and classification [4], and can also serve as surrogate model for complex PDEs. Techniques such as physics-informed neural networks enhance approximation when data is limited [13, 21]. These methods have found applications in modeling heat transfer in materials, simulating fluid dynamics etc [9, 17, 20]. Advanced architectures, such as DeepONet [18] and Fourier Neural Operator [15], are neural operators designed for learning and approximating operators. They are employed in data-driven surrogate modeling to enable rapid assessments without requiring full simulations, particularly in engineering applications such as structural analysis, optimization of aerodynamic designs, and forecasting in climate models [7, 11, 19].

This project seeks to explore the realm of scientific machine learning within the context of the 2023 CEMRACS summer school. By combining insights derived from partial differential equations (PDE) with cutting-edge machine learning techniques, this work proposes to provide a benchmark for some PDE problems using multiple learning techniques. We study the potential and limits of machine-learning model to replace the traditional numerical solvers.

We begin by considering the conventional elliptic Poisson equation and subsequently adapt it to cater to geosciences scenarios, e.g by adding source terms representing localized injection in an underground reservoir. Regarding this equation, we compare multiple predictive models: the supervised Multi-Layer Perceptron (MLP), the Physics Informed Neural Network (unsupervised MLP), the semi-supervised MLP, the Deep Operator Network (DeepONet) and the Fourier Neural Operator (FNO). Moreover, we also applied the Proper Orthogonal Decomposition (POD) as a reference surrogate model.
Finally, we study the incompressible two-phase equation which is a time-dependent problem and replace the traditional finite volume solver by a machine learning model: the Fourier Neural Operator. We first realize single predictions and then we iterate in time applying subsequently the machine-learning model on its own predictions i.e in an auto-regressive manner. Our results show the potential of machine-learning model to predict accurately solution to differential equation problems and compare the predictive capability of different neural architectures. Moreover we highlight the limits of machine learning models considering the lack of guarantees in the resulting predictions compared to the traditional solvers.

1. Presentation of the methods

1.1. **Proper Orthogonal Decomposition**

Proper Orthogonal Decomposition (POD) [2] [3] is a mathematical technique widely used to analyze and extract information from complex data sets. It involves Truncated Singular Value Decomposition (SVD) to reduce the dimensionality of large datasets and create a POD basis.

SVD decomposes a dataset into orthogonal basis functions, or modes, ordered by importance. These modes succinctly represent the dataset's structure and dynamics. In fluid dynamics, the leading singular vectors correspond to primary POD modes, capturing dominant flow patterns.

After obtaining the POD basis, Galerkin projection is applied for efficient representation of high-dimensional system dynamics using a reduced set of modes. This involves approximating governing Partial Differential Equations (PDEs) with the POD basis functions, offering computational efficiency in studying and simulating system dynamics while reducing the associated computational cost.

1.2. **Machine learning**

In this section, we first present neural network architectures that will be used in the benchmark. We use three different architectures, a Multi Layer Perceptron (MLP) which is the most commonly used model, then Deep Operator Network (DeepONet) and Fourier Neural Operator (FNO) which are neural networks architectures adapted to physical problems. Then we present three ways to train those models: supervised, unsupervised and semi-supervised learning.

1.2.1. *Multi Layer Perceptron (MLP)*

Multi-layer Perceptrons (MLPs) (see [6, Chapter 6]) are a fundamental type of artificial neural network.

Architecture of an MLP
An MLP consists of multiple layers, including an input layer, one or more hidden layers, and an output layer.

Input Layer
Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be the input vector of n features.

Hidden Layers
For the i-th hidden layer with $m^{(i)}$ neurons and with $w^{(i)}$ and $b^{(i)}$ the learnable parameters respectively the weights and biases:

$$z_j^{(i)} = \sum_{k=1}^{n} w_{j,k}^{(i)} x_k + b_j^{(i)}, \quad \text{where} \quad j = 1, 2, \ldots, m^{(i)}$$
$$a_j^{(i)} = \sigma(z_j^{(i)}), \quad \text{where } \sigma \text{ is the activation function.}$$

Output Layer
For the output layer with o neurons:

$$z_k^{(L)} = \sum_{j=1}^{m^{(L-1)}} w_{j,k}^{(L)} a_j^{(L-1)} + b_k^{(L)}, \quad \text{where} \quad k = 1, 2, \ldots, o$$
$$\widehat{u}_k = \sigma(z_k^{(L)}),$$

Let θ represent the set of all learnable parameters across all layers. Specifically:

$$\theta = \left\{ \{w^{(i)}, b^{(i)}\}_{i=1}^{L} \right\},$$

where L is the total number of layers in the network, and $w^{(i)}$, $b^{(i)}$ are the weights and biases for layer i. Finally, we denote the approximation of a function u by a neural network $\widehat{u}(\theta; \cdot)$.

1.2.2. *Deep Operator Network (DeepONet)*

DeepONet [18] is a neural network architecture designed for learning and approximating operators. The goal is to learn an operator $\mathcal{O}$ that maps a function f to another function u, i.e $\mathcal{O}(f) = u$. Unlike traditional neural networks, such as the MLP, that approximate scalar or vector-valued functions, DeepONet focuses on approximating mappings between infinite-dimensional objects.

In the most used way, DeepONet is composed of two neural networks, a branch net and a trunk net as depicted in the figure 1. In practice, the branch network takes a discrete representation of the input function f evaluated at m distinct sensor points $[f(x_1), f(x_2), \cdots, f(x_m)]$ and the trunk network takes a coordinate y at any point of a domain where we want to evaluate u. DeepONet is a linear combination of p basis functions.

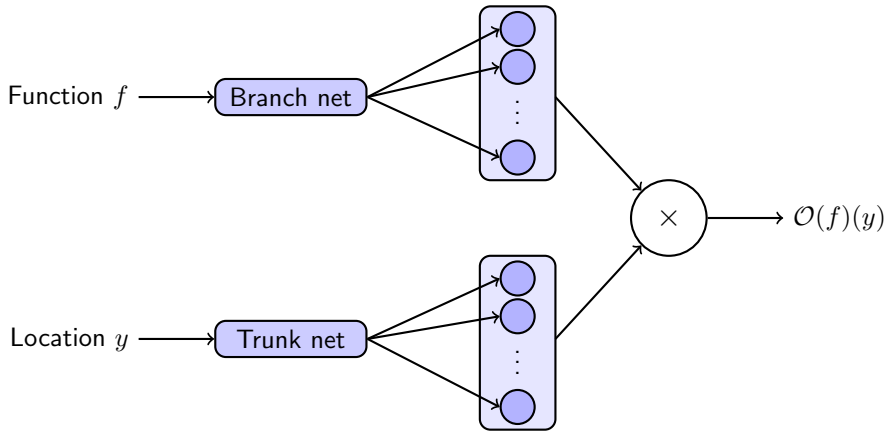


FIGURE 1. DeepONet architecture is composed of two networks: a Branch and a Trunk net. The branch evaluates the function f at different points and returns p basis. The trunk takes any point y on the domain of $\mathcal{O}(f)$ and returns p coefficients. Finally, the two outputs are multiplied and summed to reconstruct the desired evaluation of the operator $\mathcal{O}$.

The basis functions b_k are the output of the branch net while the coefficients t_k of the linear combination are given by the trunk net:

$$u(y) \approx \mathcal{O}(f)(y) \approx \sum_{k=1}^{p} b_k(f) t_k(y).$$

The branch and the trunk net can be any neural architecture. In this work we considered DeepONets composed of two MLPs. The learnable parameters are therefore the respective parameters of the branch and trunk nets. For simplicity, we denote again by $\widehat{u}(\theta;\cdot)$ the approximation of a function u by a DeepONet architecture.

1.2.3. *Fourier Neural Operator (FNO)*

Neural operators are especially useful to learn solution operators associated with PDEs (e.g. Burgers, Advection, Darcy and Navier-Stokes equations [15]). We present succinctly the original neural operator framework from [12], then we detail a specific neural operator: the Fourier Neural Operator (see [15]).

The main parts consist in a lifting operation of the inputs to a higher-dimensional space, apply multiple Kernel Operators analogous to hidden layers, and ultimately project back to the output function. Formally, let A and U be spaces of functions. Given input functions $a \in A = A(D;\mathbb{R}^{d_a})$ with values in $\mathbb{R}^{d_a}$ defined on a bounded domain $D \subset \mathbb{R}^d$, and output functions $u \in U = U(D';\mathbb{R}^{d_u})$ with values in $\mathbb{R}^{d_u}$ and defined on the bounded domain $D' \subset \mathbb{R}^{d'}$. Neural operator $\mathcal{O}_\theta : A \mapsto U$ has the following structure:

(1) **Lifting**: map the input $\{a : D \to \mathbb{R}^{d_a}\} \mapsto \{v_0 : D \to \mathbb{R}^{d_{v_0}}\}$ with $d_{v_0} > d_a$ to the first hidden representation.
(2) **Iterative Kernel Integration**: map each hidden representation to the next $\{v_t : D_t \to \mathbb{R}^{d_{v_t}}\} \mapsto \{v_{t+1} : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}\}$ for $t \in [0, 1, .., T-1]$ (with t the layer index) by combining a local linear operator, a linear integral kernel operator, and a bias function, and then applying a fixed, point-wise non linearity i.e an activation function. It is to note that $D_0 = D$ and $D_T = D'$.
(3) **Projection**: map the output of the last layer $\{v_T : D \to \mathbb{R}^{d_{v_T}}\} \mapsto \{u : D' \to \mathbb{R}^{d_u}\}$ with $d_{v_T} > d_u$ to the output function.

This can be written as:

$$\mathcal{O}_\theta := Q \circ \sigma_T(W_{T-1} + \mathcal{K}_{T-1}) \circ \cdots \circ \sigma_1(W_0 + \mathcal{K}_0) \circ P\,, \tag{1}$$

where $P : \mathbb{R}^{d_a} \mapsto \mathbb{R}^{d_{v_0}}$ is the lifting mapping, $Q : \mathbb{R}^{d_{v_T}} \mapsto \mathbb{R}^{d_u}$ is the projection mapping, $W_t : \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}}$ are local linear operators and $\mathcal{K}_t : \{v_t : D_t \mapsto \mathbb{R}^{d_{v_t}}\} \mapsto \{v_{t+1} \mapsto \mathbb{R}^{d_{v_{t+1}}}\}$ are the linear kernel integral operators. Finally, σ is a non-linear activation function. The kernel integral operator mapping can be defined by:

$$\left(\mathcal{K}_\theta v_t\right)(x) := \int_D \kappa_\theta(x, y) v_t(y) dy, \forall x \in D \tag{2}$$

where κ_θ denotes a neural network with parameters by θ and serving as a kernel function which is to learn from data. Consequently, the overall structure closely resembles that of a finite dimensional neural network.

The direct numerical computation of this kernel proves to be excessively costly. Hence, the Fourier Neural Operator suggests replacing the kernel integral operator in equation (2) with a convolution operator defined within Fourier space. This substitution facilitates the utilization of the Fast Fourier Transform (requiring a uniform discretization) for the efficient computation of (2) with quasi-linear complexity. To achieve this, the constraint $\kappa_\theta(x, y) = \kappa_\theta(x - y)$ is imposed, making (2) a convolution operator.

The Fourier integral operator $\mathcal{K}$ is then defined as:

$$\left(\mathcal{K}_\theta v_t\right)(x) = \mathcal{F}^{-1}\left(R_\theta \cdot (\mathcal{F} v_t)\right)(x) \quad \forall x \in D \tag{3}$$

with $\mathcal{F}$ the Fourier transform, $\mathcal{F}^{-1}$ the inverse Fourier transform and $R_\theta = \mathcal{F}(\kappa_\theta)$. This notation presupposes the periodicity of κ to allow a Fourier series expansion. Additionally, a finite-dimensional parameterization can be chosen by truncating the Fourier series at a maximal number of modes $k_{max} = |\{k \in \mathbb{Z}^d : |k_j| \leq k_{max,j}, j = 1, \cdots, d\}|$. Consequently, R_θ is the complex-valued tensor of parameters to learn.

1.2.4. *Neural network training*

Training a neural network consists in fitting its learnable parameters so as it can accurately map some input data to a desired output or prediction. This fitting is realized by minimizing an objective or loss function denoted as L and can be supervised, unsupervised or semi-supervised. In supervised learning, the loss function measures the misfit between the neural network prediction and the true solution obtained from the solver. In our case, we can define the supervised loss function as the mean squared error MSE:

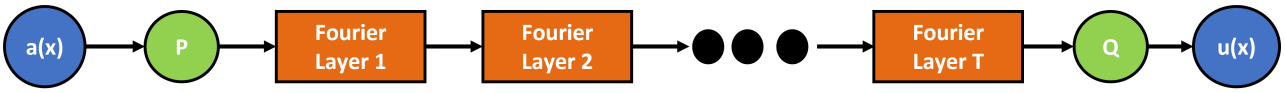
$$L_{super} = \frac{1}{N} \sum_{i=0}^{N} |\widehat{u}(\theta; x_i) - u(x_i)|^2, \tag{4}$$

with N the number of data points, f_θ and f are respectively the neural network prediction and the solution. f evaluated at the points $x_{0 \leq i \leq N}$ can be numerically costly in the context of physical simulation and may lead to have access to only little data points. This poses challenges for training the deep neural network, necessitating a substantial volume of data.

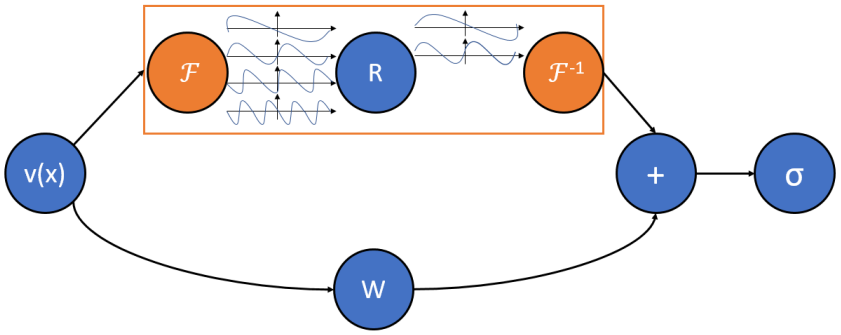
The Physics Informed Neural Network (PINN) framework proposes to train in an unsupervised manner the neural network, necessitating no data by defining a loss function based on a differential equation residual and therefore enforcing physical constraints. Given the residual of a general differential equation G parameterized:

$$G(\vec{x}, \widehat{\mathbf{u}}(\theta; \vec{x})) = 0, \quad \vec{x} \in D \tag{5}$$

and subject to boundary conditions, where $D \subset \mathbb{R}^n$ is the domain of definition and $u(\vec{x})$ is the solution to be computed. The solution u is approximated using the collocation method. Given points $x_i^{\widehat{D}}$ on a subdomain $\widehat{D}$ strictly included in D and points $x_i^{\widehat{S}}$ on the boundary of D. The approximation of u is realized by a neural



(A) Architecture of a Fourier Neural Operator



(B) Fourier Layer

FIGURE 2. **(A) Architecture of a Fourier Neural Operator**: Given an input a, lift to a higher dimension channel space by a neural network P, then apply T layers of integral operator i.e Fourier layers. Finally, project back to the desired dimension using a neural network Q and get the output u. **(B) Fourier layer**: starts with an input vector v, applies the Fourier transform $\mathcal{F}$ to it, then performs a linear transformation R on the lower Fourier modes while filtering out the higher modes. The inverse Fourier transform $\mathcal{F}^{-1}$ is then applied. A local linear transformation W is applied to the original input vector v. The output of the top and bottom operations are then added together and an activation function is applied.

network parameterized by θ and denoted as $\widehat{u}(\theta;\vec{x})$. The loss function is then defined (in the most popular way) as the sum of a mean squared error on the residual term and a mean squared error on the boundary points:

$$L_{unsuper} = \frac{1}{N_G}\sum_{i=1}^{N_G} |G(x_i^{\widehat{D}}, \widehat{u}(\theta; x_i^{\widehat{D}}))|^2 + \lambda \frac{1}{N_u}\sum_{i=1}^{N_u} |\widehat{u}(\theta; x_i^{\widehat{S}}) - u(x_i^{\widehat{S}})|^2, \tag{6}$$

with λ and hyperparameter balancing the contribution between the boundary conditions and the residual.

Finally when data points are available, it is possible to combine the supervised and the unsupervised losses leading to a semi-supervised loss:

$$L_{semi-super} = L_{super} + L_{unsuper} \tag{7}$$

Once the loss function is defined, the optimization is often realized using Gradient-based optimization methods or quasi-Newton methods. In particular, stochastic gradient descent (SGD) and all its variants [22] such as Adam [10] are popular for PINN optimization. Additionally, L-BFGS [16] is a popular quasi-Newton method for PINN optimization.

2. Elliptic problems and benchmark results

To understand the concept of learning an operator relating parameters to the solution of a partial differential equation, we investigate the one-dimensional heat equation with Dirichlet conditions on $(0,1)$:

$$\begin{cases} -\nabla \cdot (D(x)\,\nabla \mathbf{u}) = f(x), & x \in (0,1) \\ \mathbf{u}(0) = \mathbf{u}(1) = 0. \end{cases} \tag{8}$$

The parameters in the problem (8) could be either the function D, f, or both. The solution operator of the parametric PDE can be presented as:

$$\mathcal{O} : \left| \begin{array}{rl} C^1(0,1) \times C(0,1) & \longrightarrow C^2(0,1) \\ (D, f) & \longmapsto \mathbf{u}. \end{array} \right. \tag{9}$$

The goal is to approximate this operator by employing the methods discussed in the preceding section. We will examine the result when the parameters D and/or f are within a specified distribution and when they exceed this distribution. To generate the inside distribution data (training and validation data) and the outside distribution data, we will use finite differences method to solve the problem (8)

$$-(\Delta x)^{-2} \left[D_{i-\frac{1}{2}}\, u_{i-1} - (D_{i-\frac{1}{2}} + D_{i+\frac{1}{2}})\, u_i + D_{i+\frac{1}{2}}\, u_{i+1} \right] = f_i. \tag{10}$$

Here Δx represents the step size, and $D_{i\pm\frac{1}{2}}$ signifies the centered approximation of D within the grid.

N.B. All neural network training in this section, is conducted on a Dell Precision 7560 laptop equipped with an NVIDIA RTX3080 graphics card featuring 16GB of memory.

2.1. Constant coefficients

We start with the simple case when D is a non-zero constant, and the source term $f = 1$. The corresponding equation in this case is given by:

$$-D\nabla^2 \mathbf{u}(x) = 1, \quad x \in (0,1). \tag{11}$$

Obviously the equation (11) is equivalent to

$$-\nabla^2 \mathbf{u}(x) = \frac{1}{D}, \quad x \in (0,1). \tag{12}$$

The study of the operator $\mathcal{O} : D \longmapsto \mathbf{u}_D$ is then equivalent to the study of $\mathcal{O} : f = \frac{1}{D} \longmapsto \mathbf{u}_f$.

To perform the benchmark, we consider a uniform grid of 101 points in $(0,1)$ to solve the problem (11), for 1000 values of D linearly sampled in $(0.1, 10)$ (figure 3 shows the solutions obtained by the solver for different values of D). We use uniformly sampled 80% of the data to train the models and the remaining 20% is used to validate the models.

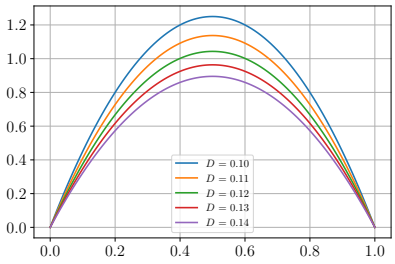


FIGURE 3. the solutions obtained by the solver for different values of D

For this case, the multi layer perceptron (MLP) neural network takes two inputs: the parameter D and the position x. It produces an approximate solution $\widehat{\mathbf{u}}(\theta; D, x)$ for each combination of these inputs. This holds true for the physics-informed neural network (PINN) as well, where $\nabla_x^2 \widehat{\mathbf{u}}(\theta; D, x)$ is computed using automatic differentiation and utilized to incorporate the equation (11) in the loss function. For the DeepONet and Fourier neural operator, as the same grid is used to train and evaluate the models, it is then not necessary to include it as an input, therefore only D is considered as input. To find suitable parameters for training the models, we utilize Optuna [1] as a hyperparameter optimization framework, automating the process of searching for optimal hyperparameters. The identified hyperparameters are presented in the table 5.

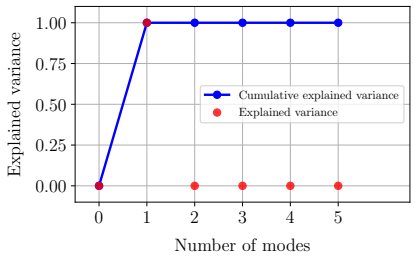


FIGURE 4. POD scree plot

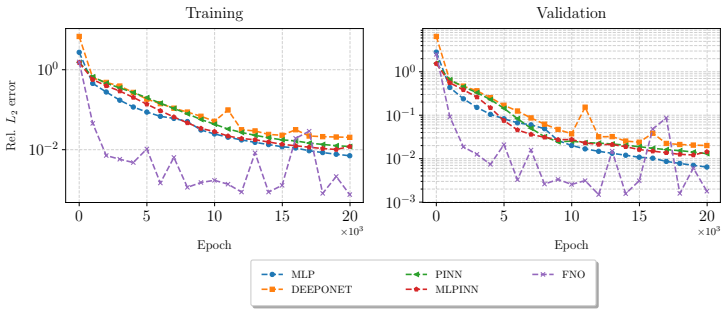


FIGURE 5. Relative L_2 error for the training and validation of machine learning models

Figure 4 presents the scree plot associated to the POD method. It illustrates the number of eigenvalues necessary to explain or interpolate the solution based on the training data. According to this plot, we can see that using one single mode explains over 99% of the variance of the operator $\mathcal{O}$. For machine learning models, the training and validation curves presented in the figure 5 shows an almost identical decrease in the relative L_2 error for all methods, except for the FNO, which exhibits a slightly more pronounced decline. The following table represents the relative L_2 error corresponding to the minimal value of the validation error for machine learning models, alongside the relative L_2 error on the validation data for the POD method.

	POD	MLP	PINN	MLPINN	DeepONet	FNO
relative L_2 error	4.6×10^{-7}	3.3×10^{-7}	1.4×10^{-6}	8.9×10^{-7}	3.2×10^{-6}	3.2×10^{-6}
time	–	1min 48	5min 36	6min 6	44.32sec	7min 33

TABLE 1. Training time and relative L_2 error corresponding to the minimal value of the validation error for machine learning models, alongside the relative L_2 error on the validation data for the POD method.

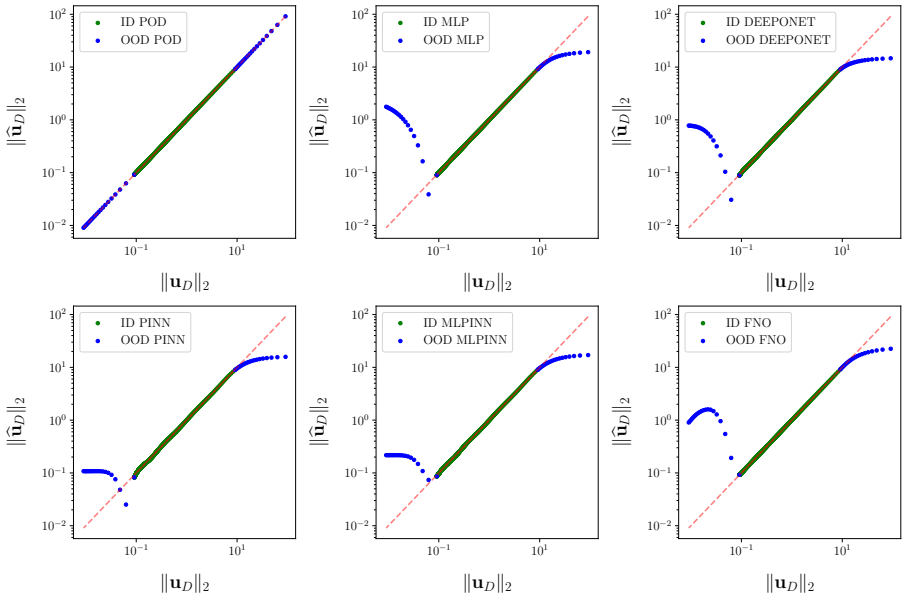


FIGURE 6. Parity plot. The green dots representing the inside distribution data (training and validation data) and blue dots representing outside distribution data which are in this case solutions generated from 21 linearly sampled parameters within each interval (0.01, 0.1) and (10, 101).

Figure 6 presents the parity plot associated to all the used methods. This plot compares the solver and the benchmark results: each result is represented by a positive value (the norm two for instance) then the solver results values are plotted in function of the benchmark results for different parameters. A line with the equation $y = x$ (the dashed red line in the figure) is plotted, indicating model performances over all the data (In distribution and out of distribution). Points that align with this line indicate successful model reproduction of the solver results.

POD is a suitable method for linear elliptic problems. As depicted in figure 6, this method allows for the approximation of the operator $\mathcal{O}$ both within and outside the distribution. For machine learning models, as shown, a good approximation is achieved within the distribution, contrasting with the poor performance observed outside it. This may be due to boundary effects, as the neural networks extrapolate out-of-distribution points that are closer to the boundary than any in-distribution data points.

2.2. **Localized source term**

The second problem includes a source term that is closer to some subsurface application, i.e a function that may represent a localized injection through a well in the underground. The problem is therefore:

$$-D\nabla^2\mathbf{u} = \delta(x, y), \quad x \in (0, 1), \tag{13}$$

with D constant over the whole domain and y is the injection location. We consider here the operator $\mathcal{O}$ mapping from the parameter y to the solution $\mathbf{u}$. It's important to note that selecting a non-differentiable function as a source term may introduce irregularities in the solution and could potentially impact the chosen method, particularly the PINNs method.

To perform the benchmark, we set $D = 10^{-3}$, and we consider a uniform grid of 101 points in $(0, 1)$ to solve the problem (13), for 61 values of y linearly sampled in $(0.2, 0.8)$ (figure 7 shows the solutions obtained by the solver for different values of D). It should be noted that the values taken by y should be included in the grid to take the injections in account. We use uniformly sampled 80% of the data to train the models and the remaining 20% is used for the validation.

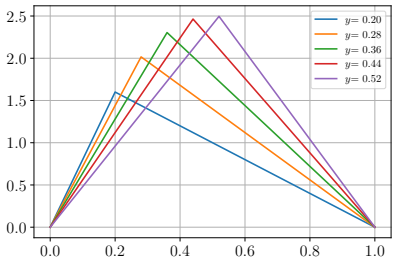


FIGURE 7. the solutions obtained by the solver for different values of y

The multi-layer perceptron (MLP) neural network takes the injection location y and the position x as inputs, generating the approximation $\widehat{\mathbf{u}}(\theta; y, x)$. This also applies to the physics-informed neural network (PINN), where $\nabla_x^2\widehat{\mathbf{u}}(\theta; y, x)$ is employed to incorporate the equation (13). In the case of deep and Fourier neural operators, only D is considered as an input. To determine suitable parameters for training the models, we utilize Optuna as a hyperparameter optimization framework. The identified hyperparameters are outlined in the table 6.

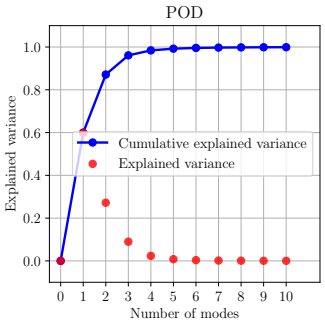


FIGURE 8. POD scree plot

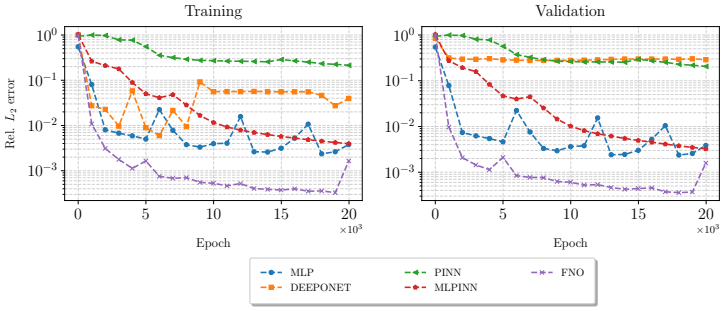


FIGURE 9. Relative L_2 error for the training and validation of machine learning models

In the case of the POD method, as depicted in Figure 8, we observe that with three modes, we surpass a 90% explained variance for the operator $\mathcal{O}$. Turning our attention to the machine learning models in figure 9, we see that PINN and DeepONet models encounter challenges in approximating the solution, these challenges primarily stem from the irregularity of the source term for the PINNs method and the inadequacy of training data for effective training of the DeepONet. Conversely, the remaining models (MLP, MLPINN, FNO) showcase proficiency in approximating the operator $\mathcal{O}$, with FNO notably achieving the lowest error.

	POD	MLP	PINN	MLPINN	DeepONet	FNO
relative L_2 error	3.8×10^{-1}	7.1×10^{-6}	6.9×10^{-2}	1.7×10^{-5}	1.1×10^{-1}	2.06×10^{-7}
time	–	51.2sec	1min 22	2min 7	55.2sec	1min 54

TABLE 2. Training time and relative L_2 error on the approximation of the operator $\mathcal{O}$ for each method.

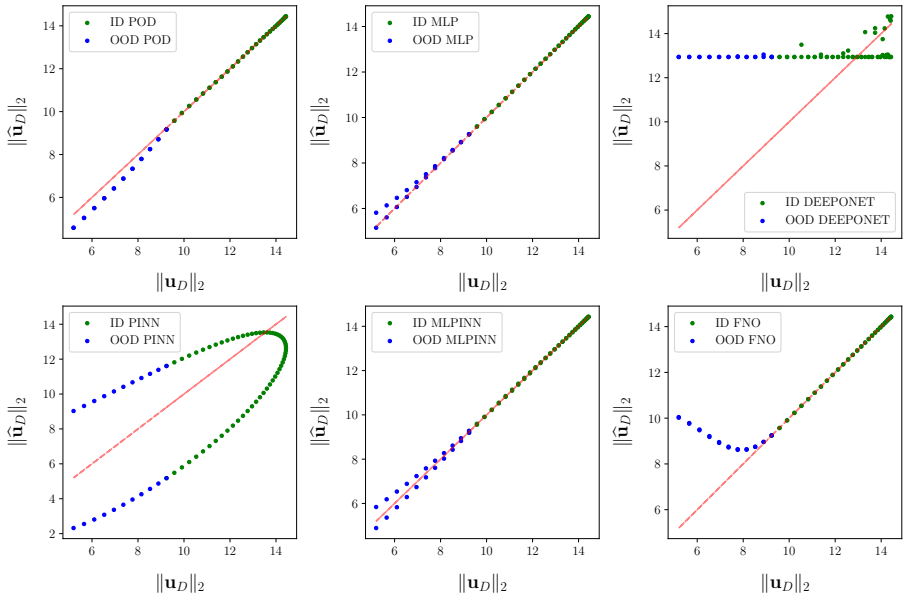


FIGURE 10. Parity plot. The green dots representing the inside distribution (ID) data set (training and validation data) and blue dots representing outside of the distribution (OOD) data set which are in this case solutions generated from 21 linearly sampled parameters within each interval (0, 0.2) and (0.8, 1).

As depicted in Figure 10, it is observable that the POD method enables the approximation of the operator $\mathcal{O}$ outside the distribution, although with less precision compared to the first case. In contrast, the MLP in this case, has successfully approximate the operator both within and outside the distribution, whereas the PINN and DeepONet encounter failure in both scenarios. Regarding the FNO, it proves successful only inside the distribution.

2.3. **Non-constant coefficients**

As we want to get closer to some geosciences applications, the diffusion coefficient must be changed to take into account the heterogeneity of the subsurface. Consequently, the diffusion coefficient, denoted as D, is now a variable dependent on position. The corresponding problem is formulated as follows:

$$-\nabla \cdot (D(x)\,\nabla \mathbf{u}) = 1, \quad x \in (0,1) \tag{14}$$

Here, we consider the mapping $\mathcal{O}$ that relates the parameter $D(x)$ to the solution $\mathbf{u}$. The expression for D is given by:

$$D(x) = 0.1 + \exp\left(-\frac{(x-c)^2}{0.02}\right). \tag{15}$$

We uniformly sample 100 values of c over the domain $(0.2, 0.8)$. For each sampled point, a corresponding D map is generated. It is to note that D is thus implicitly defined by only one parameter. To perform the benchmark, We use uniformly sampled 80% of the data to train the models and the remaining 20% is used for the validation.

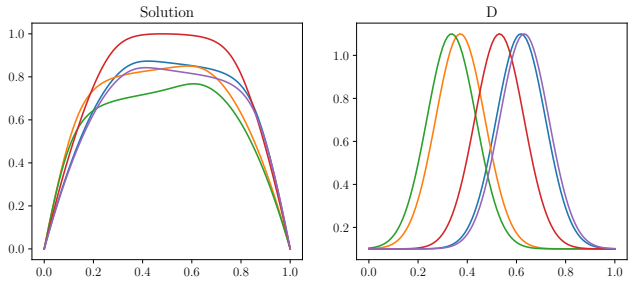


FIGURE 11. the solutions obtained by the solver for different values of y

The multi layer perceptron (MLP) neural network takes the position x and the associated diffusion diffusion coefficient $D(x)$ as inputs, it produce the approximation $\widehat{\mathbf{u}}(\theta; D(x), x)$. In real situations, the value of D is known only for specific positions and there is no available analytical expression for D. Therefore, moving forward, we will refrain from employing the PINN method, assuming that we have knowledge of D values only within the grid. For both DeepONet and FNO $D(x)$ is considered as the input. To find suitable parameters for training the models, we employ Optuna as a hyperparameter optimization framework. The identified hyperparameters are presented in the table 6.

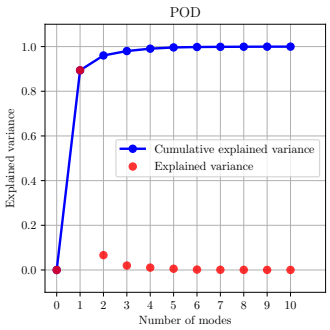


FIGURE 12. POD scree plot

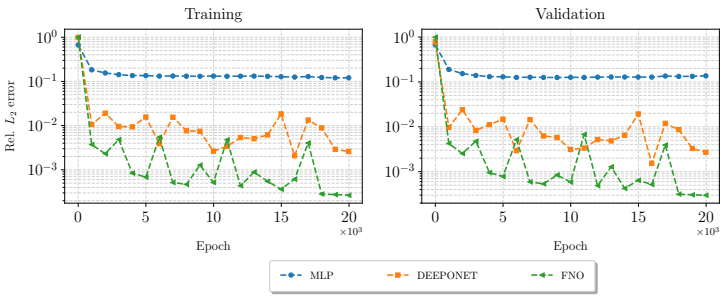


FIGURE 13. Relative L_2 error for the training and validation of machine learning models

From figure 12, we can see that with two modes we are above 90% of explained variance for the operator $\mathcal{O}$. For machine learning models, according to the figure 13, the MLP model fails to approximate the operator while

DeepONet and FNO manage to approach it with the FNO achieve the lowest error. This is can be attributed primarily to the structure and the number of parameters, enabling it to adapt effectively to operator learning. The following table represents the relative L_2 error corresponding to the minimal value of the validation error for machine learning models, alongside the relative L_2 error on the validation data for the POD method.

	POD	MLP	DeepONet	FNO
relative L_2 error	3.7×10^{-3}	7.1×10^{-3}	2.5×10^{-7}	4.1×10^{-8}
time	–	1min 2	57.4sec	1min 56

TABLE 3. Training time and relative L_2 error on the approximation of the operator $\mathcal{O}$ for each method.

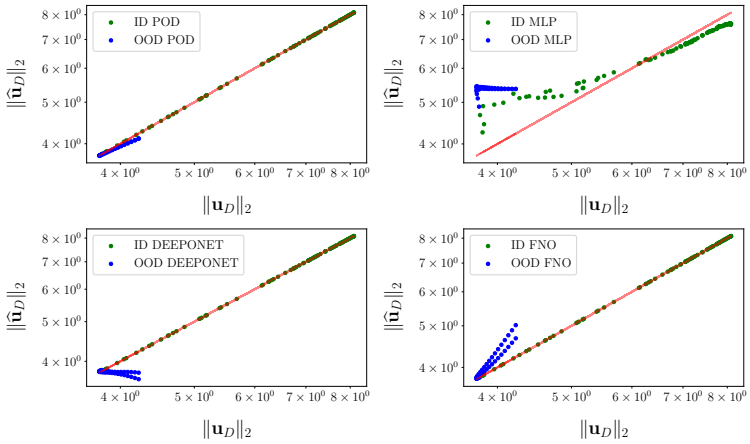


FIGURE 14. Parity plot. The green dots representing the inside distribution data (training and validation data) and blue dots representing outside distribution data which are in this case solutions generated from 21 linearly sampled parameters within each interval (0.1, 0.2) and (0.8, 0.9).

Figure 14 illustrates that, within the distribution, all models demonstrate satisfactory performance, with the exception of MLP. Conversely, outside the distribution, the POD method exhibits relatively favorable performance when compared to alternative methods. As noted the simple structure of MLP did not allow the accurate learning of the operator even within the given distribution.

2.4. **Non constant coefficients and localized source term**

The last problem considered is the combination of the two last problems:

$$-\nabla \cdot (D(x)\,\nabla \mathbf{u}) = \delta(x, y), \quad x \in (0, 1). \tag{16}$$

Here the operator $\mathcal{O}$ maps from the parameters $(D(x), y)$ to the solution $\mathbf{u}$.

To perform the benchmark, we consider a uniform grid of 101 points in $(0, 1)$ to solve the problem (16). We employ D maps of the form (15) with 61 different values of c linearly sampled in $(0.2, 0.8)$. The injection positions correspond to the values of c indicating the locations of highest diffusivity. We use uniformly sampled 80% of the data to train the models and the remaining 20% is used for the validation.

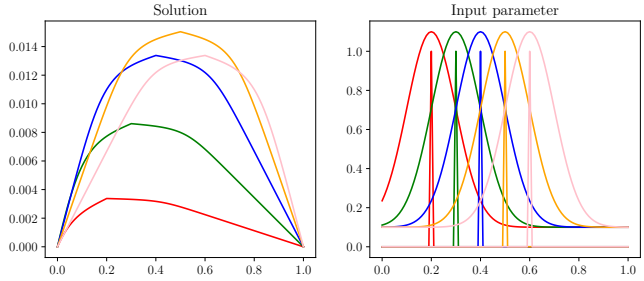


FIGURE 15. the solutions obtained by the solver for different values of y

The multi layer perceptron (MLP) neural network takes the parameter y and the position x with the corresponding diffusion coefficient $D(x)$ as inputs and produce the approximation $\hat{\mathbf{u}}(\theta; y, x, D(x))$. For DeepONet and FNO the inputs are y and $D(x)$. To find suitable parameters for training the models, we employ Optuna as a hyperparameter optimization framework, automating the process of searching for optimal hyperparameters. The identified hyperparameters are presented in the table 6.

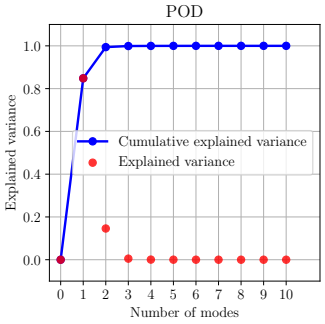


FIGURE 16. POD scree plot

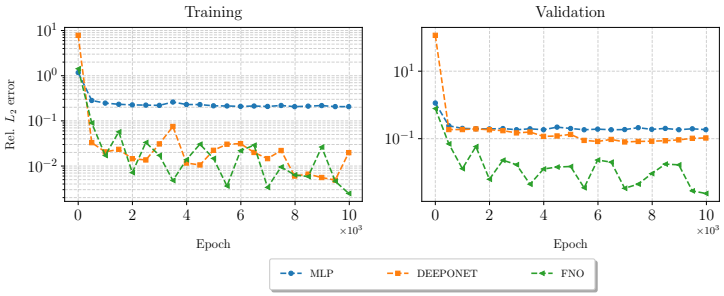


FIGURE 17. Relative L_2 error for the training and validation of machine learning models

Based on figure 16, we can see that with two modes we are above 90% of explained variance for the operator $\mathcal{O}$. Figure 17 indicates that among the machine learning models, the FNO successfully approximates the operator $\mathcal{O}$. In contrast, the MLP and DeepONet struggle in their approximation, despite the training errors of FNO and DeepONet being relatively close. The following table represents the relative L_2 error corresponding to the minimal value of the validation error for machine learning models, alongside the relative L_2 error on the validation data for the POD method.

	POD	MLP	DeepONet	FNO
relative L_2 error	4.2×10^{-4}	2.5×10^{-6}	4.6×10^{-7}	4.1×10^{-10}
time	–	28.2sec	30.76sec	1min 3

TABLE 4. Training time and relative L_2 error on the approximation of the operator $\mathcal{O}$ for each method.

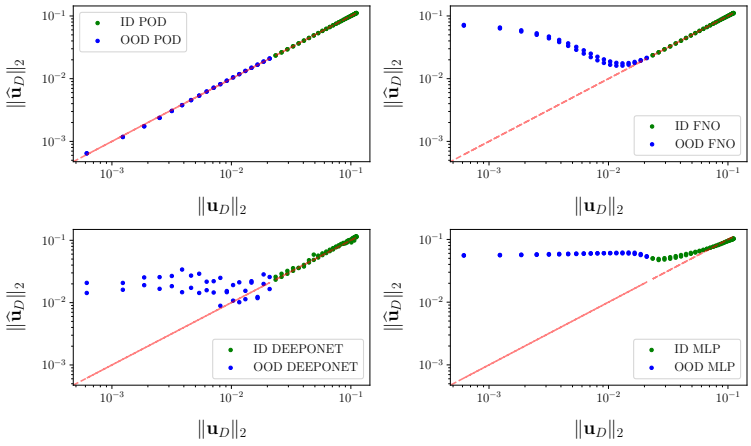


FIGURE 18. Parity plot. The green dots representing the inside distribution data (training and validation data) and blue dots representing outside distribution data which are in this case solutions generated based on 21 values of c linearly sampled within each interval (0, 0.2) and (0.8, 1).

In figure 18, it is shown that FNO performs very well within the distribution, followed by DeepONet. Conversely, the MLP demonstrates poor performance within the distribution. Outside the distribution, the POD method exhibits favorable performance compared to the other methods.

3. Incompressible two-phase flow benchmark and results

We are now examining a "piston" reservoir problem which is time-dependent. We replace the traditional numerical solver by a black-box machine learning model and iterate through time using the machine-learning model in an auto-regressive manner.

Problem formulation

Let us examine a two-phase fluid consisting of an aqueous phase denoted as w and a gaseous phase denoted as g as it flows through a porous medium. Our focus is on understanding the changes in pressure P_α and saturation S_α within each phase.

The conventional method for modeling two-phase flow in a porous medium involves applying Darcy's law [8] to each phase $\alpha \in \{w, g\}$. This law establishes a connection between the phase velocity v_α and the pressure gradient:

$$v_\alpha = -\frac{\overline{\mathrm{K}} k_{r_\alpha}(S_\alpha)}{\mu_\alpha} \nabla P_\alpha, \tag{17}$$

here $\overline{\mathrm{K}}$ represents the permeability tensor, k_{r_α} denotes the relative permeability dependent on saturation S_α, and μ_α stands for the dynamic viscosity.

We treat the medium as isotropic, allowing us to consider the permeability tensor $\overline{\mathrm{K}}$ as a scalar field denoted as K. Reservoir geometries are selected such that gravity can be disregarded. Capillary effects are also neglected, leading to equal pressures for water and gas, i.e., $P_w = P_g$. Additionally, we adopt a quadratic relative permeability law , i.e $kr_\alpha(S_\alpha) = S_\alpha^2$.

For incompressible fluids, the conservation equations governing the changes in pressure P and in saturations S_w and S_g are as follows:

$$\phi\frac{\partial}{\partial t}(S_w) + div(v_w) = 0, \quad \phi\frac{\partial}{\partial t}(S_g) + div(v_g) = q_g, \tag{18}$$

$$v_w = -\frac{Kkr_w(S_w)}{\mu_w}\nabla P, \quad v_g = -\frac{Kkr_g(S_g)}{\mu_g}\nabla P, \tag{19}$$

$$S_g + S_w = 1. \tag{20}$$

Here, ϕ represents the medium porosity field and q_g is the gas flow injected into the wells.

We substitute S_w for $1 - S_g$ using equation (20) allowing us to omit the indices: the variables of interest are the pressure P and the gas saturation denoted as S.

The continuous model is discretized using a standard two-points finite volume spatial scheme on a Cartesian mesh, coupled with an implicit/backward Euler time scheme. The pressure and saturation are solved simultaneously through a fully implicit scheme [5] [14]. Ultimately, the resulting non-linear system of equations is solved using Newton's method.

Data generation

We examine a 1D "piston" reservoir problem discretized in 100 cells. The porosity and the permeability remain constant throughout the whole reservoir. Concerning the boundary conditions, gas is injected at the left boundary with a specific pressure P_{left}. At the right boundary, the pressure is set to 10 MPa, and the initial conditions correspond to a reservoir filled exclusively with water.

Our goal is to learn the operator $\mathcal{O} : (P_{left}, P^t, S^t) \longmapsto (P^{t+1}, S^{t+1})$, with P^t, S^t, P^{t+1} and S^{t+1} respectively the pressure and saturation at the times t and $t + 1 = t + \Delta t$ for $t \in [0, 1, \cdots, T]$.

We generate 10000 data snapshots using a fully implicit solver with a constant time-step Δt of 1 year. Beginning at $t = 0$ with an initial pressure of 10 MPa, we sample a left boundary pressure between 10.5 MPa and 20 MPa. Then we iterate in time for a total of 101 steps, saving the reservoir state at each iteration as pressure and saturation snapshots. The initial snapshot, initialized with a non-usable pressure, is discarded. This process is repeated 100 times, resulting in 10000 snapshots, each comprising a pressure and saturation map. Finally, we split the whole dataset in three sets: train, test and validation with a 60/20/20 ratio by batch originating from the same simulation (i.e 100 per 100). YADS (Yet Another Darcy Solver) python library is used to run the simulations (see Yads).

3.1. **Results**

We present the results of the problem 3. We use the Fourier Neural Operator architecture with 12 parameterized modes, such as in the original paper from [15], and 128 channels, an arbitrary power of two (16 or 32 in the original paper). The input features are the pressure and saturation snapshots, together with a constant map of boundary pressure, therefore a single input data shape is $(1, 100, 3)$. The predicted features are the pressure and saturation after a time-step, therefore the output shape for a single sample is $(1, 100, 2)$. We train on a A100 GPU using Adam optimizer, a learning rate of 1×10^{-4}, a momentum of 0.9 during 2500 epochs and keep the model parameters corresponding the the lowest test value. We show in figure 20 the loss curve. The minimum loss value is 1.3×10^{-3} and is reached at epoch 2427. Its corresponding train loss value is 2.2×10^{-3}.

We consider two cases for evaluation, first given a snapshot generated by the traditional solver, we predict the solution at the next step of time. Secondly, given a snapshot at $t = 0$, we apply subsequently the machine-learning model on its outputs to iterate in time.

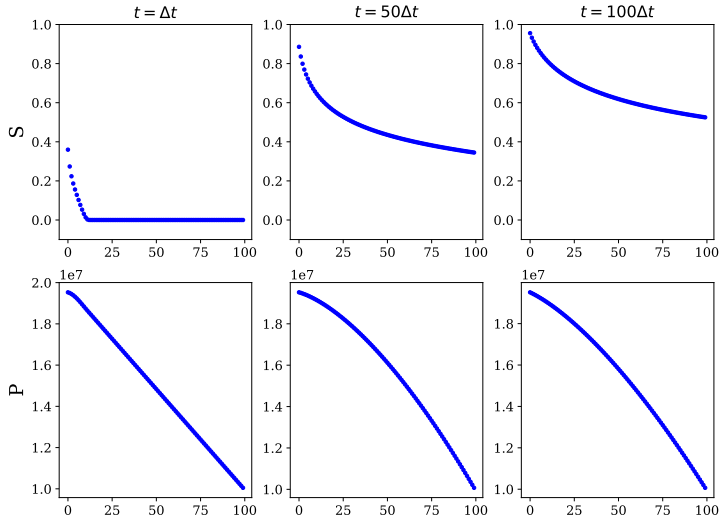


FIGURE 19. Example of saturation and pressure solutions at different times using a left boundary pressure of 19.5×10^6 Pa.

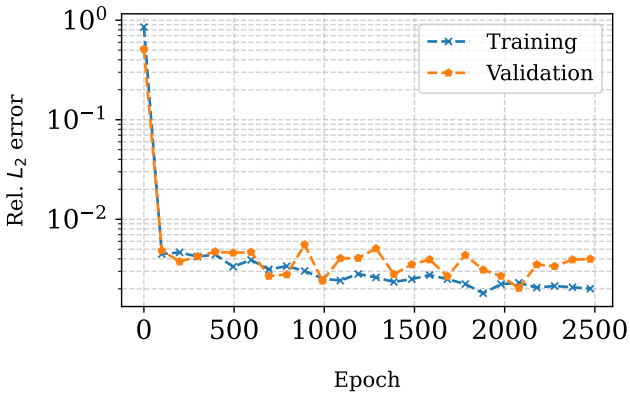


FIGURE 20. Evolution of relative L_2 error for training and validation through epochs.

Regarding the first case, i.e the direct evaluation of the model, we show on figure 21 example of predictions at different times and figure 22 depicts the corresponding parity plot over the whole datasets. We observe that given a snapshot originating from the traditional simulator, it seems possible to predict accurately the solution in saturation and pressure after a step of time and for different boundary conditions (left pressure in this case).

Time iteration

Now, given an initial snapshot at $t = 0$, we iterate through time using only the machine-learning model i.e, we consecutively apply the model on its own predictions. The figure 23 exposes an example where the pressure remains close to the solution for all snapshots while the saturation remains accurate everywhere except at the front of saturation.

Then, figure 24 depicts an example where the predictions in saturation and pressure remain accurate even after multiples iterations in time. It is to note that we apply for this case clipping on the saturation to ensure

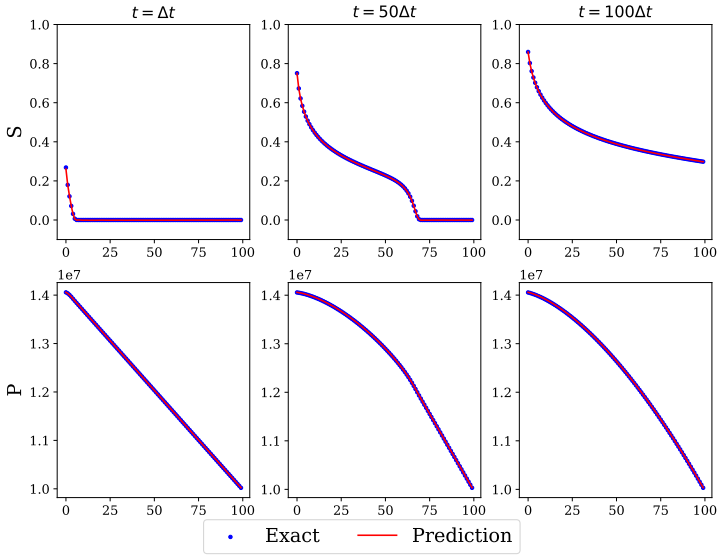


FIGURE 21. Example of exact and predicted saturation and pressure at different times with a left boundary pressure of 14.1×10^6 Pa.

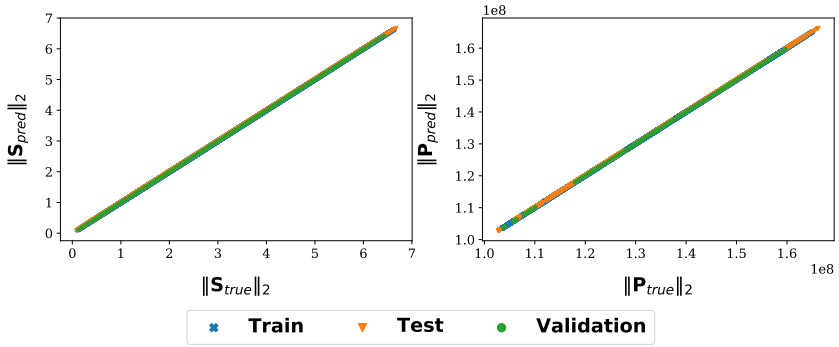


FIGURE 22. Parity plot of the predicted versus exact saturation (left figure) and pressure (right figure) using the L_2 norm for the three datasets.

that the saturation remains between 0 and 1. Appendix 25 exposes the same test case without clipping, resulting in predictions significantly drifting away from the solution.

4. Discussion and conclusion

In this work we replaced the traditional numerical solvers of an elliptic differential equation and a hyperbolic time-dependent differential equation by surrogate models using POD and machine-learning models. Regarding the elliptic equation, we derived four cases starting from a standard Poisson equation and adding progressively constraints related to geosciences. Our results show that it is possible to learn accurately the solution of differential equations for a wide range of input parameters. Overall, the neural operators such as the FNO or the DeepONet outperformed the other machine-learning models. Moreover, the POD surrogate model is well-suited for elliptic problems and show significantly better accuracy compared to machine-learning models

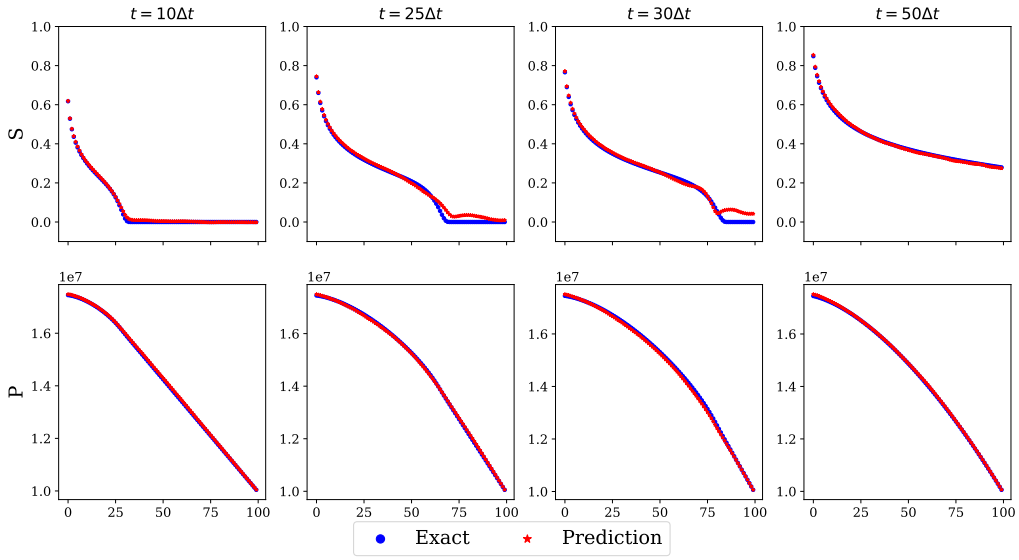


FIGURE 23. Example of saturation and pressure snapshots obtained using the machine-learning model to iterate through time starting from a time $t = 0$ and with a left boundary pressure of 17.5×10^6 Pa.

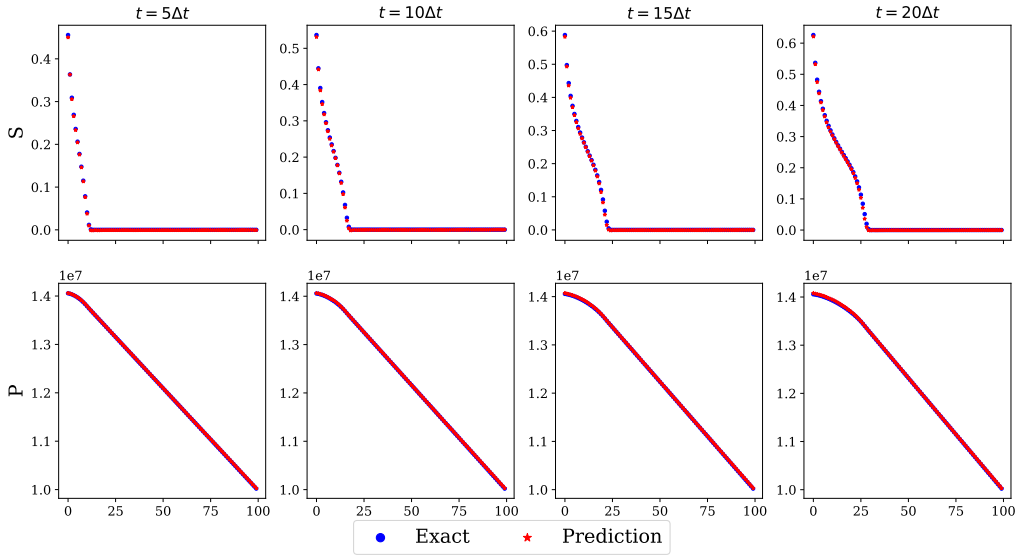


FIGURE 24. Example of saturation and pressure snapshots obtained using the machine-learning model and saturation clipping to iterate through time starting from a time $t = 0$ and with a left boundary pressure of 14.1×10^6 Pa. The predictions are represented in red dots while the solutions are in blue dots.

in the resulting predictions regardless of the distribution of the input parameters. Then, considering the time-dependent incompressible two-phase flow equation, we replaced the traditional solver by a FNO model. We observe that the predictions on single samples are quite accurate. Finally, when iterating in time by using the machine-learning model on its own predictions, we observe that the resulting predictions are overall accurate except at the front of saturation where the majority of the errors are located. Moreover, without clipping the saturation prediction in a physical range, errors may accumulate resulting in predictions significantly diverging from the solution. It is to note that this work lacks a figure exposing the behavior at the entire dataset scale, such as a parity plot, for this case. This missing figure is a straightforward perspective.

Appendix

hyperparameters / models	architecture	activations	optimizer	learning rate
MLP	$2, 30, 23, 20, 19, 32, 1$	tanh	Adam	$2e-4$
PINN	$2, 17, 15, 26, 31, 1$	tanh	Adam	$2e-4$
MLPINN	$2, 17, 15, 26, 31, 1$	tanh	Adam	$2e-4$
DeepONet	Branch: $1, 22, 9, 21, 32, 19.$ Trunk: $1, 13, 19$	tanh	Adam	$4e-04$
FNO	$1, 33, 45, 38, 49$ with Fourier modes $15, 12, 14$	tanh	Adam	$8e-4$

Table 5. hyperparameters table

hyperparameters / models	architecture	activations	optimzer	learning rate
MLP	$2, 24, 28, 11, 18, 21, 1$	$3 \times relu$, $2 \times tanh$, $relu$	Adam	$4e-3$
PINN	$2, 21, 30, 1$	tanh	Adam	$4e-3$
MLPINN	$2, 17, 15, 26, 31, 1$	tanh	Adam	$1e-4$
DeepONet	Branch: $2, 17, 15, 26, 31, 1.$ Trunk: $1, 868, 434, 552, 929, 756, 215$	tanh	Adam	$1e-03$
FNO	$1, 55, 32, 50, 37, 42$ with Fourier modes $14, 12, 15, 15$	tanh	Adam	$4e-4$

Table 6. hyperparameters table

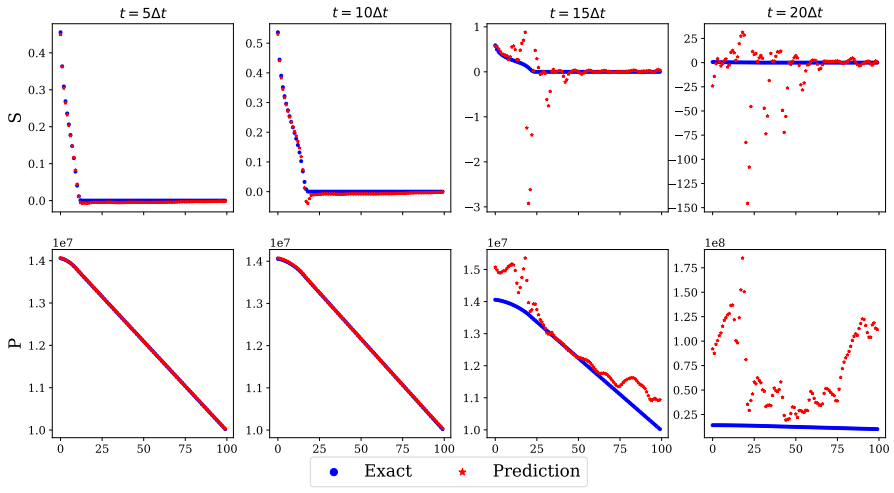


Figure 25. Example of saturation and pressure snapshots obtained using the machine-learning model to iterate through time starting from a time $t = 0$ and with a left boundary pressure of 14.1×10^6 Pa.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *CoRR*, abs/1907.10902, 2019.

[2] Peter Benner, Stefano Grivet-Talocia, Alfio Quarteroni, Gianluigi Rozza, Wil Schilders, and Luís Miguel Silveira. *System-and data-driven methods and algorithms*. De Gruyter, 2021.

[3] Peter Benner, Mario Ohlberger, Albert Cohen, and Karen Willcox. *Model reduction and approximation: theory and algorithms*. SIAM, 2017.

[4] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[5] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016.

[7] Somdatta Goswami, Minglang Yin, Yue Yu, and George Em Karniadakis. A physics-informed variational deeponet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 391:114587, 2022.

[8] M. King Hubbert. Darcy's Law and the Field Equations of the Flow of Underground Fluids. *Transactions of the AIME*, 207(01):222–239, 12 1956.

[9] K Jambunathan, SL Hartle, S Ashforth-Frost, and VN Fontama. Evaluating convective heat transfer coefficients using neural networks. *International Journal of Heat and Mass Transfer*, 39(11):2329–2332, 1996.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[11] Kazuma Kobayashi, James Daniell, and Syed Bahauddin Alam. Improved generalization with deep neural operators for engineering systems: Path towards digital twin. *Engineering Applications of Artificial Intelligence*, 131:107844, 2024.

[12] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces, 2023.

[13] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[14] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

[15] Zongyi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *CoRR*, abs/2010.08895, 2020.

[16] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(1–3):503–528, aug 1989.

[17] Yang Liu, Nam Dinh, Yohei Sato, and Bojan Niceno. Data-driven modeling for boiling heat transfer: using deep neural networks and high-fidelity simulation results. *Applied Thermal Engineering*, 144:305–320, 2018.

[18] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *CoRR*, abs/1910.03193, 2019.

[19] Lu Lu, Raphaël Pestourie, Steven G Johnson, and Giuseppe Romano. Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. *Physical Review Research*, 4(2):023210, 2022.

[20] Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[21] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[22] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.