ϕ-FEM-FNO: a new approach to train a Neural Operator as a fast PDE solver for variable geometries

Michel Duprez, Vanessa Lleras, Alexei Lozinski, Vincent Vigon, Killian Vuillemot

ϕ-FEM-FNO: a new approach to train a Neural Operator as a fast PDE solver for variable geometries

Michel Duprez*, Vanessa Lleras†, Alexei Lozinski‡,
Vincent Vigon§ and Killian Vuillemot¶

February 22, 2024

Abstract

In this paper, we propose a way to solve partial differential equations (PDEs) by combining machine learning techniques and a new finite element method called ϕ-FEM. For that, we use the Fourier Neural Operator (FNO), a learning mapping operator. The purpose of this paper is to provide numerical evidence to show the effectiveness of this technique. We will focus here on the resolution of the Poisson equation with non-homogeneous Dirichlet boundary conditions. The key idea of our method is to address the challenging scenario of varying domains, where each problem is solved on a different geometry. The considered domains are defined by level-set functions due to the use of the ϕ-FEM approach. We will first recall the idea of ϕ-FEM and of the Fourier Neural Operator. Then, we will explain how to combine these two methods. We will finally illustrate the efficiency of this combination with some numerical results on two test cases, showing in particular that our method is faster than learning-based and finite element solvers for a fixed accuracy.

1 Introduction

Finite Element Method (FEM) is one of the most popular approaches to approximate the solutions of Partial Differential Equations (PDE) arising in engineering, physics, biology, and other applications (see e.g. [10]). It is important to solve them quickly (sometimes in real-time) with good accuracy. There have been numerous attempts to achieve this using machine learning-based (ML-based) methods. They can be split into two groups :

*MIMESIS team, Inria de l'Université de Lorraine, MLMS team, Université de Strasbourg, 2 Rue Marie Hamm, 67000 Strasbourg, France, michel.duprez@inria.fr

†IMAG, Univ Montpellier, CNRS UMR 5149, 499-554 Rue du Truel, 34090 Montpellier, France, vanessa.lleras@umontpellier.fr

‡Université de Franche-Comté, Laboratoire de mathématiques de Besançon, UMR CNRS 6623, 16 route de Gray, 25030 Besançon Cedex, France, alexei.lozinski@univ-fcomte.fr

§Institut de Recherche Mathématique Avancée, UMR 7501, Université de Strasbourg et CNRS, Tonus team, Inria de l'Université de Lorraine, 7 rue René Descartes, 67000 Strasbourg, France, vincent.vigon@math.unistra.fr

¶IMAG, Univ Montpellier, CNRS UMR 5149, 499-554 Rue du Truel, 34090 Montpellier, France. MIMESIS team, Inria de l'Université de Lorraine, MLMS team, Université de Strasbourg, 2 Rue Marie Hamm, 67000 Strasbourg, France, killian.vuillemot@umontpellier.fr

1. **Physics-inspired approaches:** ML-based methods can be used as an approximation ansatz and approximate the solution of PDEs by minimizing the residual or the associated energy of the PDEs and the distance to some observations, without ever using traditional approximation by FEM or similar. The most popular member of this class of methods is PINNs [23], but one can also cite Deep Galerkin [25] and Deep Ritz methods [29]. Despite the initial promise, there is now abundant numerical evidence that these methods do not outperform the classical FEM in terms of solution time and accuracy, see for instance a recent study in [11]. It seems that these methods cannot thus be considered as good candidates for real-time realistic computations.

2. **Classical solver as database:** Classical FEM (or similar) is used to obtain a "database" of solutions for a collection of representative parameter values that are used to train a neural network to learn the mapping linking the parameters to the solution. All this step is computationally expensive and is done in the preparatory stage (offline). The expected outcome is that one can use the trained network to obtain the solution for any given parameters almost instantaneously (online). Examples include U-Net (see e.g. [9]), graph neural operator [15], DeepOnet [18] and Fourier Neural Operator (FNO) [16, 14].

In our article, we focus on FNO as the method that showed a superior cost-accuracy tradeoff over the others (see [16]). The issue with FNO is that it needs Cartesian grids to perform discrete fast Fourier transform, and the initial implementation was thus limited to problems posed on rectangular boxes. There have been attempts to adapt FNO to general geometries, cf. Geo-FNO [14] where the irregular input domain is deformed into a uniform latent mesh on which the FFT can be applied. In our article, we propose an alternative approach: we treat the geometry, given by the level-set function, as one of the inputs of the network alongside the other data of the problem while using a Cartesian grid without deforming it. Incidentally, this viewpoint of treating the geometry (i.e.thanks to the level-set function) together with the data to construct an approximation using a simple (ex. Cartesian) grid was also the starting point to develop ϕ-FEM. It is thus natural to combine ϕ-FEM (at the offline training stage) with FNO. As a bonus, this combination, which we will call ϕ-FEM-FNO, allows us to avoid the interpolation errors from a body-fitted mesh to a Cartesian one, which would be inevitable if we would use a traditional FEM for training.

This paper aims to illustrate the efficiency of our approach ϕ-FEM-FNO, in the case of complex and varying domains for the Poisson-Dirichlet problem:

$$\begin{cases} -\Delta u &= f\,, \quad \text{in } \Omega\,, \\ u &= g\,, \quad \text{on } \Gamma\,, \end{cases} \tag{1}$$

where Ω is a connected domain of $\mathbb{R}^d, d = 1, 2, 3$ and Γ its boundary.

Our contributions:

- We propose a new machine learning approach called ϕ-FEM-FNO which takes as input the parameters of the PDE (f, g) and the geometry of the domain given by a level-set function ϕ and gives as output an approximation of the solution of the PDE. In comparison to [14], we do not need a transformation between the geometry and the unit square.

- In addition, we highlight in Fig. 11 that our approach has a better accuracy/CPU-time ratio than Geo-FNO, which has been compared with other techniques in [14], and than a FNO trained using standard FEM solutions interpolated on cartesian grids.
- Since the geometry is described by a level-set function, the prediction of the PDE solution is exact on the boundary of the domain. We can also find this idea in [26] for an approach with some PINNS.

The paper will be divided into three parts. In section 2, we will first describe the two methods used: ϕ-FEM and FNO. We will then present in Section 3 our idea to combine them. Finally, we will illustrate the efficiency of the method with numerical results with varying ellipses and complex shapes in Section 4 and give some conclusions in Section 5. In the end, we detail in the first appendix the standardization operator. The second appendix details the calibration of the hyperparameters used for our FNO, and in the third appendix, we justify numerically our choice of loss function. In the last appendix, we present the optimizer algorithm.

2 Description of the methods

In the rest of the manuscript, Ω is a domain of dimension 2 included in $[0,1]^2$.

2.1 Overview

Our idea is to build a neural network which will be an approximation of the operator mapping the data f, g, and the geometry to the solution of (1). We want the output to be obtained with good accuracy and a low computational time. The objective is to train this neural network using synthetic data generated by a discrete solver of PDE. The neural network and the discrete solver must be chosen to perform independently of each other, and must also be compatible.

As a discrete solver, we choose ϕ-FEM [8] which is a finite element method with an immersed boundary approach using a level-set function to describe the geometry of the domain. The optimal convergence of ϕ-FEM has been previously proven theoretically and numerically for the Poisson equation with Dirichlet boundary conditions [8], with Neumann boundary conditions [5], for the Stokes problem [6] and for the Heat-Dirichlet equation [7]. Moreover, in [3], the efficiency of the method has been illustrated numerically on multiple examples in the case of linear elasticity (better than the continuous Lagrange FEM approach on conformal meshes).

As neural network, we have decided to use the Fourier Neural Operator (FNO), introduced in [16] and [13]. The FNO relies on an iterative architecture proposed in [15]. An advantage of FNO is that it takes a step size much bigger than is allowed in numerical methods. In the case of the approximation of the PDE solution, the authors of [16] have illustrated that FNO has better performance than the classical Reduced Basis Method (using a POD basis) [4], a Fully Convolution Networks [31], an operator method using PCA as an autoencoder on both the input and output data and interpolating the latent spaces with a neural network [2], the original graph neural operator [15], the multipole graph neural operator [17], a neural operator method based on the low-rank decomposition of a kernel similar to the unstacked DeepONet proposed in [18], a ResNet (18 layers of 2-d convolution with residual connections)

[12], a U-Net [24] and TF-Net [28]. Furthermore, the training can be done on many PDEs with the same underlying architecture.

Moreover, these two methods are compatible since ϕ-FEM is a precise non-conforming finite element method, that can be used on cartesian grids, as required by the FNO that will be used.

In the next subsection, we will describe ϕ-FEM and FNO. We introduce in Table 1 notations that will be used in the rest of the manuscript.

	Notation	Meaning
FNO	θ	Set of trainable parameters
	$\mathcal{G}_\theta$	Operator mapping the input to the solutions
	$\mathcal{G}^\dagger$	Ground truth operator mapping the solutions : ϕ-FEM resolution
	$\mathcal{F}$, $\mathcal{F}^{-1}$	Discrete Fourier and inverse Fourier transform
	$W^{\mathcal{C}^l_\theta}$	Linear transformation applied on lower Fourier modes
	$\mathcal{C}^l_\theta$	Convolution layer
	$\mathcal{B}^l_\theta$	Linear transformations applied on the spatial domain
	P_θ, Q_θ	Transformations between high dimension channel space and original space
	N, N^{-1}	Standardization and unstandardization operators
	σ	Non linear activation function
ϕ-FEM	ϕ	Level-set function defining the domain Ω and its boundary Γ
	$\mathcal{O}$	Box $[0,1]^2$
	$\mathcal{T}_h$	ϕ-FEM computational mesh
	$\mathcal{T}_h^\Gamma$	Set of cells of $\mathcal{T}_h$ cut by the boundary
	$\mathcal{F}_h^\Gamma$	Set of internal facets of $\mathcal{T}_h^\Gamma$
	σ_D	Stabilisation parameter

Table 1: Notations table.

2.2 Description of ϕ-FEM

Let us first briefly describe the ϕ-FEM method introduced in [8] to solve (1). We will skip many theoretical aspects but refer the reader to [8] for more details. We suppose that the domain Ω is included in the box $\mathcal{O} = [0,1]^2 \subset \mathbb{R}^2$. In the ϕ-FEM approach, to solve (1), we assume that the domain Ω and its boundary Γ are given by a level-set function $\phi : [0,1]^2 \to \mathbb{R}$ in the following way

$$\Omega := \{\phi < 0\} \qquad \text{and} \qquad \Gamma := \{\phi = 0\}\,. \tag{2}$$

Let $\mathcal{T}_h^\mathcal{O}$ be a cartesian mesh of $\mathcal{O} = [0,1]^2$ composed by simplexes of size h. More precisely, the background Cartesian mesh $\mathcal{T}_h^\mathcal{O}$ is obtained by dividing $\mathcal{O} = [0,1]^2$ into $(n_x - 1) \times (n_y - 1)$ rectangular cells and then dividing each of them into two triangular cells by a diagonal, where $n_x = 1/h_x + 1$, $n_y = 1/h_y + 1$ and $h = \sqrt{h_x^2 + h_y^2}$. Denoting by ϕ_h the Lagrange interpolation of ϕ on $\mathcal{T}_h^\mathcal{O}$, we consider the submesh $\mathcal{T}_h$ of $\mathcal{T}_h^\mathcal{O}$, called the computational mesh, composed of the cells of $\mathcal{T}_h^\mathcal{O}$ intersecting the domain $\{\phi_h < 0\}$, i.e.

$$\mathcal{T}_h := \left\{T \in \mathcal{T}_h^\mathcal{O} :\ T \cap \{\phi_h < 0\} \neq \emptyset\right\}\,.$$

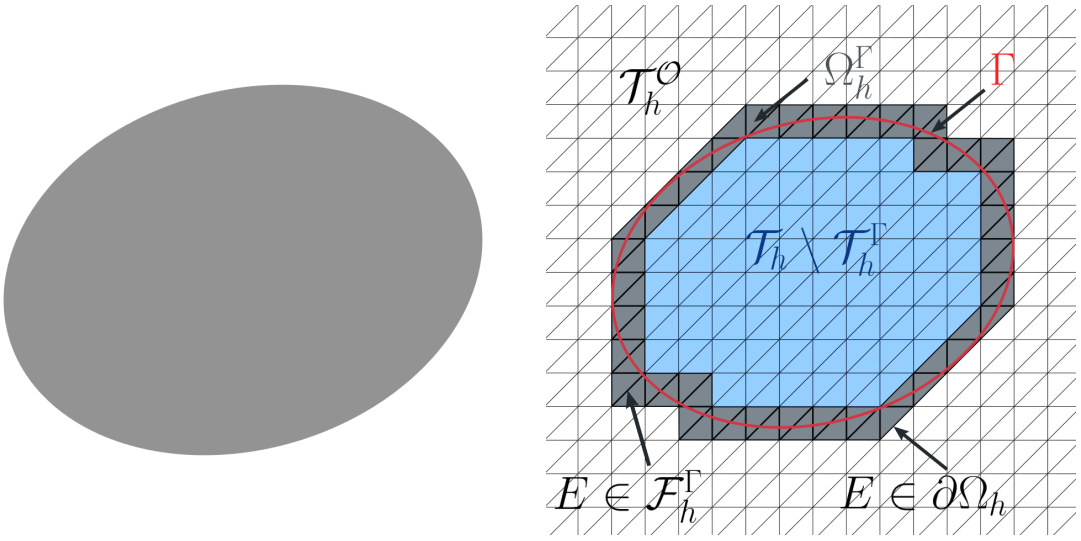

Figure 1: Left: example of exact domain Ω. Right: associated considered ϕ-FEM meshes. In red, the exact boundary Γ of the domain Ω, in white $\mathcal{T}_h^{\mathcal{O}}$, in gray $\mathcal{T}_h^\Gamma$ and in blue, $\mathcal{T}_h \setminus \mathcal{T}_h^\Gamma$.

We also introduce the submesh $\mathcal{T}_h^\Gamma$ containing the cells cut by the approximate boundary ($\{\phi_h = 0\}$), i.e.

$$\mathcal{T}_h^\Gamma := \{T \in \mathcal{T}_h \;\; T \cap \{\phi_h = 0\} \neq \emptyset\} \; .$$

We denote by Ω_h and Ω_h^Γ the domains occupied by $\mathcal{T}_h$ and $\mathcal{T}_h^\Gamma$, respectively, and by $\partial\Omega_h$ the boundary of Ω_h (different from $\Gamma_h = \{\phi_h = 0\}$). All the meshes are illustrated for a specific domain in Fig. 1. Finally, we need to introduce a set of facets containing all the internal faces of the mesh $\mathcal{T}_h^\Gamma$, i.e., the faces of $\mathcal{T}_h^\Gamma \setminus \partial\mathcal{T}_h$. Referring to Fig. 1 (right), these faces are the ones of the gray cells except the ones common to a gray cell and a white cell. We will denote by $\mathcal{F}_h^\Gamma$ this set, defined by

$$\mathcal{F}_h^\Gamma := \{F \text{ (an internal facet of } \mathcal{T}_h) \text{ such that } \exists \; T \in \mathcal{T}_h : T \cap \Gamma_h \neq \emptyset \text{ and } F \in \partial T\} \, .$$

Let $k \geqslant 1$ be an integer. We define the finite element space

$$V_h^{(k)} := \{v_h \in H^1(\Omega_h) \;\; v_h|_T \in \mathbb{P}_k(T) \; \forall \; T \in \mathcal{T}_h\} \, .$$

We now introduce the considered ϕ-FEM formulation of system (1) (see [8]): Find $w_h \in V_h^{(k)}$ such that, for all $s_h \in V_h^{(k)}$, denoting $u_h = \phi_h w_h + g_h$ and $v_h = \phi_h s_h$,

$$\int_{\Omega_h} \nabla u_h \cdot \nabla v_h - \int_{\partial\Omega_h} \frac{\partial u_h}{\partial n} v_h + G_h(u_h, v_h) = \int_{\Omega_h} f_h v_h + G_h^{rhs}(v_h) \, ,$$

where g_h, f_h are some Lagrange interpolations of g and f, respectively,

$$G_h(u, v) = \sigma_D h \sum_{E \in \mathcal{F}_h^\Gamma} \int_E [\partial_n u] \, [\partial_n v] + \sigma_D h^2 \sum_{T \in \mathcal{T}_h^\Gamma} \int_T \Delta u \Delta v \, ,$$

and

$$G_h^{rhs}(v) = -\sigma_D h^2 \sum_{T \in \mathcal{T}_h^\Gamma} \int_T f_h \Delta v \, .$$

The brackets in G_h stand for the jump over the facets of $\mathcal{F}_h^\Gamma$, $\partial_n u$ stands for the normal derivative of u and $\sigma_D > 0$ is a h-independent parameter.

2.3 The "ground truth" operator

In the rest of the manuscript, f_h, g_h, ϕ_h, u_h and w_h will represent the matrices of $\mathbb{R}^{n_x \times n_y}$ associated to these $\mathbb{P}_1$-functions composed for each index $i = 0, \dots, n_x - 1$, $j = 0, \dots, n_y - 1$, of the values of an extrapolation in $V_h^{\mathcal{O}}$ of these functions at the node of the mesh $\mathcal{T}_h^{\mathcal{O}}$ of coordinate (x_i, y_j), with $x_i = h_x i := i/(n_x - 1)$, $y_j = h_y j := j/(n_y - 1)$.

In the tradition of FNO literature, the FNO will approximate an operator called the "ground truth operator" which is denoted by $\mathcal{G}^\dagger$. In our case, $\mathcal{G}^\dagger$ will be the operator mapping the data f_h, g_h, and the geometry given by the level-set ϕ_h to the ϕ-FEM approximated solution w_h . More precisely, $\mathcal{G}^\dagger$ will be defined as follows:

$$\begin{array}{rccc} \mathcal{G}^\dagger : & \mathbb{R}^{n_x \times n_y \times 3} & \to & \mathbb{R}^{n_x \times n_y \times 1} \\ & (f_h, \phi_h, g_h) & \mapsto & w_h \,, \end{array} \tag{3}$$

where w_h is the ϕ-FEM solution associated to f_h, g_h and ϕ_h. The exact form of this extrapolation has no impact on the construction of the FNO since the values of w_h outside Ω_h are not seen in the loss defined below. In practice, this extrapolation will be done by FEniCS ([1]).

2.4 Architecture of the FNO

We will now introduce a few essential points to understand the architecture of the FNO. We refer the reader to [16, 13, 14] for detailed explanations on the FNO or [15] for more details about neural operators.

The goal of the FNO is to construct a parametric mapping

$$\begin{array}{rccc} \mathcal{G}_\theta : & \mathbb{R}^{n_x \times n_y \times 3} & \to & \mathbb{R}^{n_x \times n_y \times 1} \,, \\ & (f_h, \phi_h, g_h) & \mapsto & w_\theta \,, \end{array}$$

that approximates the "ground truth" mapping $\mathcal{G}^\dagger$ (3). We predict the ϕ-FEM representation w_h with w_θ and $u_\theta = \phi_h w_\theta + g_h$ will be an approximation of $u_h = \phi_h w_h + g_h$ as described in Figure 2. Here θ stands for the numerous parameters that we have to find by minimizing the loss function. So by predicting w_h instead of directly predicting the ϕ-FEM solution u_h, our approximation u_θ will be exactly equal to g_h on the discrete boundary Γ_h.

Remark. The idea of predicting w_h instead of u_h directly is driven by the fact that the multiplication by ϕ_h imposes the exact boundary conditions. Indeed, predicting u_h would lead to prediction errors on the boundary of the domain and would lead to the need to add a loss acting on the values of the solution on the boundary. Moreover, this additional term would have not corrected entirely the issues of precision on the boundary since we would never have an exact imposition of the boundary condition with the prediction of the neural operator.

2.4.1 The structure of the FNO

Layer sequence The mapping $\mathcal{G}_\theta$ is composed of several sub-mappings, called layers

$$\mathcal{G}_\theta = N^{-1} \circ Q_\theta \circ \mathcal{H}_\theta^4 \circ \mathcal{H}_\theta^3 \circ \mathcal{H}_\theta^2 \circ \mathcal{H}_\theta^1 \circ P_\theta \circ N \,. \tag{4}$$

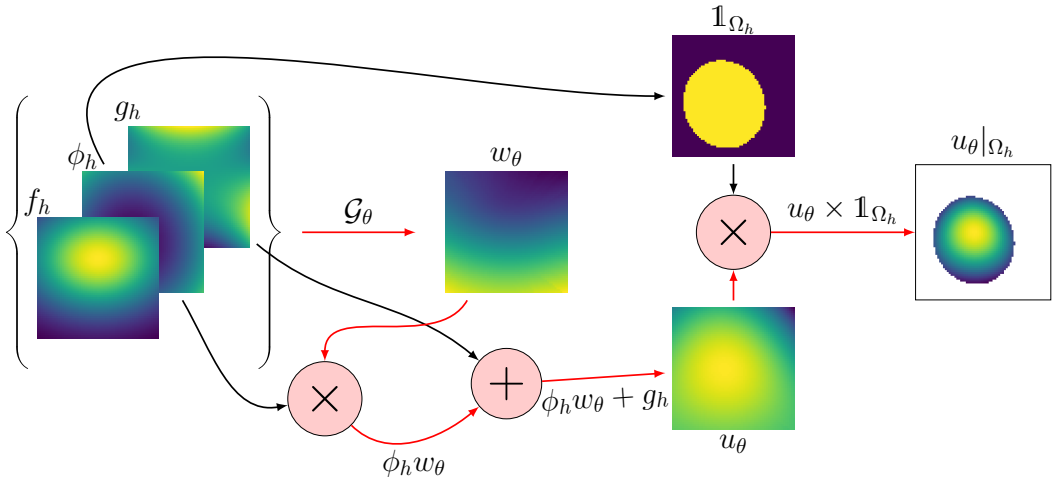

Figure 2: Construction of a prediction of ϕ-FEM-FNO.

Each layer acts on three-dimensional tensors with the third dimension (the number of channels) varying from one layer to another according to the following scheme:

$$\mathcal{G}_\theta : \mathbb{R}^{n_x \times n_y \times 3} \xrightarrow{N} \mathbb{R}^{n_x \times n_y \times 3} \xrightarrow{P_\theta} \mathbb{R}^{n_x \times n_y \times n_d} \xrightarrow{\mathcal{H}^1_\theta} \mathbb{R}^{n_x \times n_y \times n_d} \xrightarrow{\mathcal{H}^2_\theta} \\ \dots \xrightarrow{\mathcal{H}^4_\theta} \mathbb{R}^{n_x \times n_y \times n_d} \xrightarrow{Q_\theta} \mathbb{R}^{n_x \times n_y \times 1} \xrightarrow{N^{-1}} \mathbb{R}^{n_x \times n_y \times 1}, \quad (5)$$

where n_d is a sufficiently large dimension. A graphic representation of $\mathcal{G}_\theta$ is given in Fig. 3. The transformations P_θ and Q_θ are respectively an embedding to the high dimensional channel space and a projection to the target dimension, both computed using a neural network (see [16]). The principal components of FNO are the 4 Fourier layers $\mathcal{H}^\ell_\theta$ having the same structure presented below.

Normalisation N and N^{-1} To improve the performance of a neural network, it is known that normalizing the inputs and outputs is almost mandatory (see [19] for example). So the train data will be standardized channel by channel in the input and unstandardized in the output thanks to the operator N and N^{-1}, respectively. This step is explained in detail in Appendix A.

Structure of the embedding P_θ and the projection Q_θ The transformation P_θ is made of one fully connected layer of size n_d neurons acting on each node, i.e. for all $i \in \{1, ..., n_x\}$, $j \in \{1, ..., n_y\}$ and $k \in \{1, ..., n_d\}$,

$$P_\theta(X)_{ijk} = \sum_{k'=1}^{3} W^{P_\theta}_{kk'} X_{ijk'} + B^{P_\theta}_k ,$$

with $W^{P_\theta} \in \mathcal{M}_{n_d,3}(\mathbb{R})$, $B^{P_\theta} \in \mathbb{R}^{n_d}$ some parameters. The transformation Q_θ is made of two fully connected layers of size n_Q then 1 acting also on each node, i.e. $Q_\theta = (Q_{\theta,ijk})_{ijk}$ defined

for all $X = (X_{ijk})_{ijk}$ by, for all $i \in \{1, ..., n_x\}$, $j \in \{1, ..., n_y\}$,

$$Q_\theta(X)_{ij} = \left[\sum_{k=1}^{n_Q} W_{1k}^{Q_\theta,2} \sigma\left(\sum_{k'=1}^{n_d} W_{kk'}^{Q_\theta,1} X_{ijk'} + B_k^{Q_\theta,1}\right)\right] + B^{Q_\theta,2}\,,$$

with $W^{Q_\theta,1} \in \mathcal{M}_{n_d,n_Q}(\mathbb{R})$, $B^{Q_\theta,1} \in \mathbb{R}^{n_Q}$, $W^{Q_\theta,2} \in \mathcal{M}_{n_Q,1}(\mathbb{R})$, $B^{Q_\theta,2} \in \mathbb{R}$ some parameters and σ is an activation function applied term by term. We choose the GELU (Gaussian Error Linear Unit) function given by $f(x) = x\Phi(x)$ with $\Phi(x) = P(X \leqslant x)$ where $X \sim \mathcal{N}(0, 1)$, as in the original implementation of the FNO[1] and of the Geo-FNO[2].

Structure of Fourier layers $\mathcal{H}_\theta^\ell$ A layer $\mathcal{H}_\theta^\ell$ is made of two sub-layers organized as follows (see [16]):

$$\mathcal{H}_\theta^\ell(X) = \sigma\big(\mathcal{C}_\theta^\ell(X) + \mathcal{B}_\theta^\ell(X)\big)\,,$$

where

- $\mathcal{C}_\theta^\ell$ is a convolution layer defined by

$$\mathcal{C}_\theta^\ell(X) = \mathcal{F}^{-1}\Big(W^{\mathcal{C}_\theta^\ell}\mathcal{F}(X)\Big) \in \mathbb{R}^{n_x \times n_y \times n_d \times n_d}\,, \tag{6}$$

  with $W^{\mathcal{C}_\theta^\ell} \in \mathbb{C}^{n_x \times n_y \times n_d \times n_d}$ some parameters and $\mathcal{F}$, $\mathcal{F}^{-1}$ stand for the real Fast Fourier Transform (RFFT and its inverse): for all $i \in \{1, ..., n_x\}$, $j \in \{1, ..., n_y\}$ and $k \in \{1, ..., n_d\}$,

$$\mathcal{F}(X)_{ijk} = \sum_{i'j'} X_{i'j'k} e^{2\sqrt{-1}\pi\left(\frac{ii'}{n_x} + \frac{jj'}{n_y}\right)}\,,$$

  and for $Y \in \mathbb{C}^{n_x \times n_y \times n_d}$

$$\mathcal{F}^{-1}(Y)_{ijk} = \sum_{i'j'} Y_{i'j'k} e^{-2\sqrt{-1}\pi\left(\frac{ii'}{n_x} + \frac{jj'}{n_y}\right)}\,.$$

- $\mathcal{B}_\theta^\ell = (\mathcal{B}_{\theta,ijk}^\ell)_{ijk}$ is the "bias-layer" defined for all $X = (X_{ijk})_{ijk}$ by, for all $i \in \{1, ..., n_x\}$, $j \in \{1, ..., n_y\}$ and $k \in \{1, ..., n_d\}$,

$$\mathcal{B}_\theta^\ell(X)_{ijk} = \sum_{k'=1}^{n_d} W_{kk'}^{\mathcal{B}_\theta^\ell} X_{ijk'} + B_k^{\mathcal{B}_\theta^\ell}\,,$$

  with $W^{\mathcal{B}_\theta^\ell} \in \mathcal{M}_{n_d}(\mathbb{R})$ and $B^{\mathcal{B}_\theta^\ell} \in \mathbb{R}^{n_d}$ are some parameters.

The coefficients $W^\cdot$ and $B^\cdot$ are the trainable parameters, which have some constraints given below.

[1] `https://github.com/neuraloperator/neuraloperator`

[2] `https://github.com/neuraloperator/Geo-FNO`

Constraint on the parameters

- **Symmetry**: To obtain a real matrix $\mathcal{C}_\theta^\ell(X)$, we impose to $W^{\mathcal{C}_\theta^\ell}$ the Hermitian symmetry, i.e. $W^{\mathcal{C}_\theta^\ell}_{n_x-i,n_y-j,k} = \overline{W}^{\mathcal{C}_\theta^\ell}_{i,j,k}$. In practice, since we use a specific implementation of the FFT called RFFT (Real-FFT): the discrete Fourier coefficients are stored in arrays of size $n_x \times (n_y/2+1)$ (integer division) and are automatically symmetrized when performing the inverse transformation. Hence, there is no precaution to take for the matrix $\hat{W}$ in practice. Simply it must be of size $n_x \times (n_y/2+1)$.
- **Low pass filter**: The solutions to our problem (1) are usually rather smooth. Hence, when we perform the RFFT, the "very high" frequencies can be neglected. They simply participate in the fact that the RFFT is a bijection. Typically, a good approximated solution can be recovered, keeping only the $m \times m$ first "low" frequencies.

Remark (Number of parameters). An interesting aspect of the FNO is the small number of parameters to optimize. Indeed, since we truncate the high frequencies, for each $\mathcal{C}_\theta^l$ the number of parameters to optimize is less than $n_x \times n_y \times n_d \times n_d$. In fact, the total number of parameters n_θ does not depend on the resolution and is given by

$$n_\theta = \overbrace{4 \times n_d}^{P_\theta \,:\, 3\times n_d + n_d} + 4 \times \underbrace{\Big(\overbrace{2 \times n_d^2 \times m^2}^{\mathcal{C}_\theta^l \,:\, \text{by truncation of high frequencies}} + \overbrace{n_d^2 + n_d}^{\mathcal{B}_\theta^l} \Big)}_{\mathcal{H}_\theta^l} + \overbrace{(n_d+2) \times n_Q + 1}^{Q_\theta \,:\, n_d \times n_Q + n_Q \times 1 + 1} .$$

In the following test cases, for the chosen hyperparameters this will represent 324577 parameters to optimize.

Remark. Once trained, FNOs can work with input data f_h, g_h, w_h of arbitrary dimensions. This property of multi resolution is due to a special structure of FNO. In fact, FNO is presented in [16, 13] as an approximation of mappings acting on infinite dimensional function spaces. It is thus not surprising that its discretization can be done, in principle, on any mesh.

Border issues An issue of the RFFT applied to non-periodic functions is the Gibbs phenomenon: some oscillations appear on the border. To erase them, we apply a padding technique: we extend the matrices, adding entries all around, before performing the convolution. After the convolution, we restrict the matrix to its original shape to partially erase the oscillations. See the last point of Appendix B for more details.

3 The proposed architecture

As previously said, the main idea of this paper is to present the combination of the two methods we have presented before, namely ϕ-FEM and the FNO. The main objective of this approach is thus to take advantage of the precision of ϕ-FEM and the capacity of the FNO to produce nearly instantaneous results once the network is trained so that the method can be used in real-time simulations. A detailed representation of the entire used pipeline is given in Fig. 3, where the input data and the final output are in red.

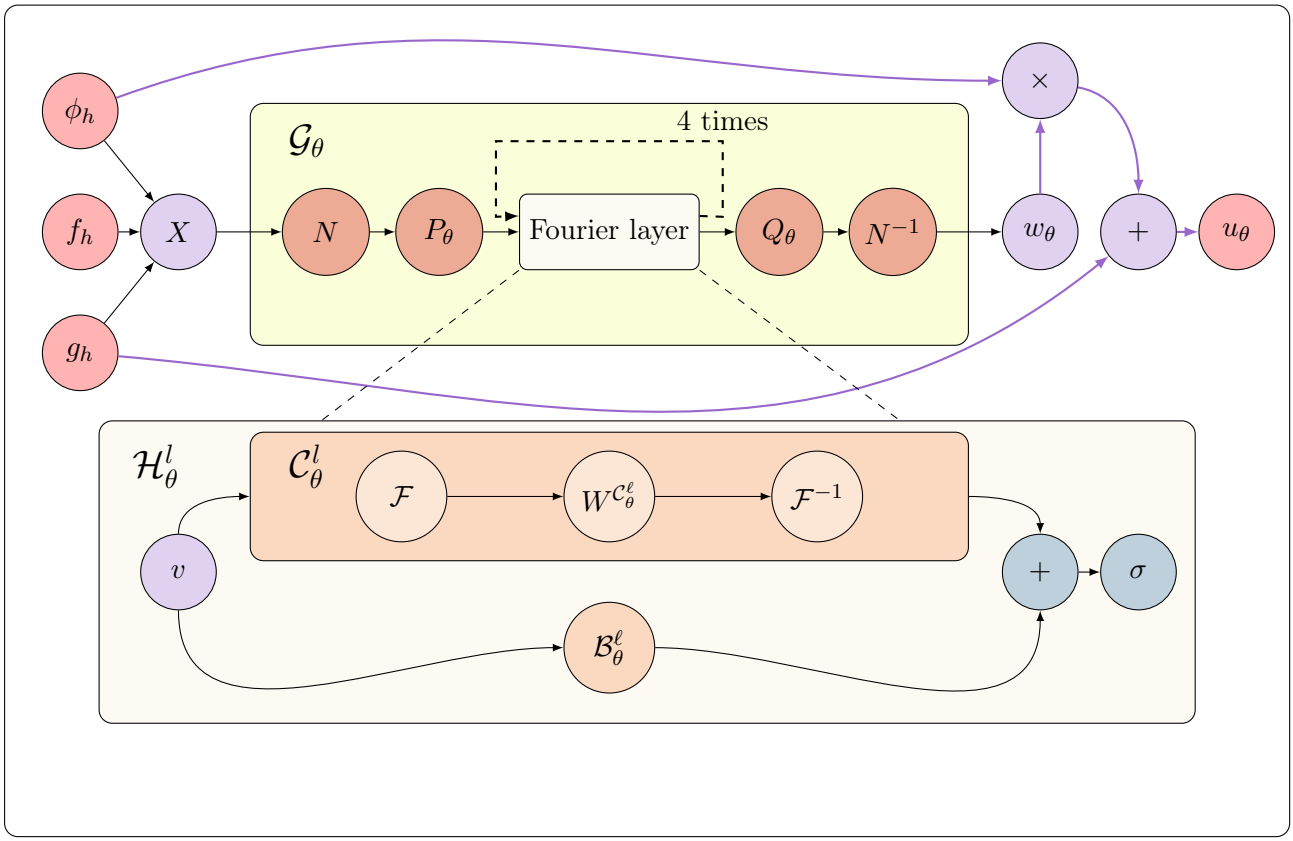

Figure 3: The ϕ-FEM-FNO pipeline. Illustration based on the representation of [16]. The upper part represents the entire pipeline, and the lower part is a zoom on a Fourier layer. The red circles correspond to the inputs provided by the user and the output returned by our ϕ-FEM-FNO. We represent the inputs and outputs seen by the FNO in purple, where $X = (f_h, \phi_h, g_h)$. In orange, P_θ and Q_θ are two transformations parameterized by neural networks. Moreover, $\mathcal{F}$ and $\mathcal{F}^{-1}$ are respectively the Fourier and inverse Fourier transforms. In blue, σ is the activation function. Finally, black arrows correspond to steps inside our FNO, and purple arrows to steps outside the FNO.

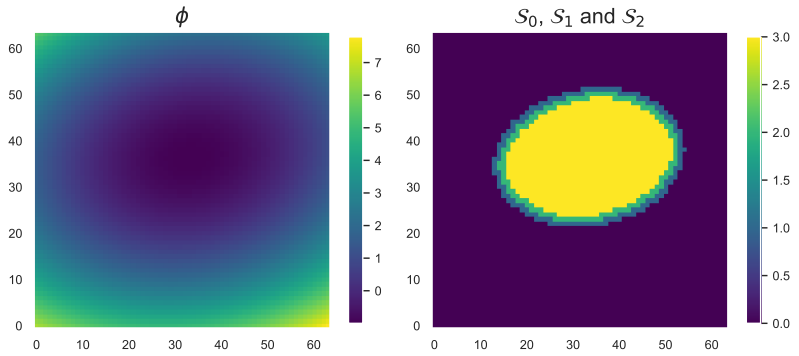

Figure 4: Left: values of the level-set function ϕ. Right: Example of $\mathcal{S}_0$ (yellow, green, and blue pixels), $\mathcal{S}_1$ (yellow and green pixels), and $\mathcal{S}_2$ (yellow pixels).

By construction, a prediction w_θ of our FNO will be given on the same cartesian regular grid of the inputs. Since we are interested in the solution only over Ω_h, we need to define a loss function acting only on the corresponding pixels. An example of data and truncated output of our approach is represented in Fig. 7.

Loss function. Let us denote by $F_h = (f_h^n)_{n=0,\dots,N_{\text{data}}}$, $\Phi_h = (\phi_h^n)_{n=0,\dots,N_{\text{data}}}$, $G_h = (g_h^n)_{n=0,\dots,N_{\text{data}}}$, and $W_\theta = (w_\theta^n)_{n=0,\dots,N_{\text{data}}} = (\mathcal{G}_\theta(f_h^n, \phi_h^n, g_h^n))_{n=0,\dots,N_{\text{data}}}$ with N_{data} the size of the considered sample (validation sample or test sample for example). Moreover, we denote $U_{\text{true}} = (u_{\text{true}}^n)_{n=0,\dots,N_{\text{data}}}$ where $u_{\text{true}}^n = \phi_h^n w_h^n + g_h^n$, the ground truth solution and $U_\theta = (u_\theta^n)_{n=0,\dots,N_{\text{data}}}$ with $u_\theta^n = \phi_h^n \mathcal{G}_\theta(f_h^n, \phi_h^n, g_h^n) + g_h^n = \phi_h^n w_\theta^n + g_h^n$ the output of ϕ-FEM-FNO.

The loss to be optimized is given by

$$\mathcal{L}\left(U_{\text{true}}; U_\theta\right) = \frac{1}{N_{\text{data}}} \sum_{n=0}^{N_{\text{data}}} \sqrt{\frac{\sum_{i=0}^{2} \mathcal{E}_i(u_{\text{true}}^n; u_\theta^n)}{\sum_{i=0}^{2} \mathcal{N}_i(u_{\text{true}}^n)}}\,, \tag{7}$$

where

$$\begin{aligned}
\mathcal{E}_0(u_{\text{true}}^n; u_\theta^n) &= \sum_{(i,j)\in\mathcal{S}_0^n} \|u_{\text{true}}^n(i,j) - u_\theta^n(i,j)\|^2\,,\\
\mathcal{E}_1(u_{\text{true}}^n; u_\theta^n) &= \sum_{(i,j)\in\mathcal{S}_1^n} \left(\|\nabla_x^h u_{\text{true}}^n(i,j) - \nabla_x^h u_\theta^n(i,j)\|^2 + \|\nabla_y^h u_{\text{true}}^n(i,j) - \nabla_y^h u_\theta^n(i,j)\|^2\right),\\
\mathcal{E}_2(u_{\text{true}}^n; u_\theta^n) &= \sum_{(i,j)\in\mathcal{S}_2^n} \Big(\|\nabla_x^h \nabla_x^h u_{\text{true}}^n(i,j) - \nabla_x^h \nabla_x^h u_\theta^n(i,j)\|^2\\
&\quad + \|\nabla_x^h \nabla_y^h u_{\text{true}}^n(i,j) - \nabla_x^h \nabla_y^h u_\theta^n(i,j)\|^2 + \|\nabla_y^h \nabla_y^h u_{\text{true}}^n(i,j) - \nabla_y^h \nabla_y^h u_\theta^n(i,j)\|^2\Big),
\end{aligned}$$

and

$$\mathcal{N}_0(u^n_{\text{true}}) = \sum_{(i,j)\in\mathcal{S}_0^n} \|u^n_{\text{true}}(i,j)\|^2\,,$$
$$\mathcal{N}_1(u^n_{\text{true}}) = \sum_{(i,j)\in\mathcal{S}_1^n} \left(\|\nabla_x^h u^n_{\text{true}}(i,j)\|^2 + \|\nabla_y^h u^n_{\text{true}}(i,j)\|^2\right)\,,$$
$$\mathcal{N}_2(u^n_{\text{true}}) = \sum_{(i,j)\in\mathcal{S}_2^n} \left(\|\nabla_x^h\nabla_x^h u^n_{\text{true}}(i,j)\|^2 + \|\nabla_x^h\nabla_y^h u^n_{\text{true}}(i,j)\|^2 + \|\nabla_y^h\nabla_y^h u^n_{\text{true}}(i,j)\|^2\right),$$

where ∇^h is the centered finite difference approximation of the gradient and $\mathcal{S}_0$ is the set of pixels satisfying $\phi \leqslant 0$, i.e. corresponding to inside vertices of Ω_h, and $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) is the set of pixels of $\mathcal{S}_0$ (resp. $\mathcal{S}_1$) deprived of a layer of pixels. These sets are represented in Fig. 4.

Remark. For the numerical part, the finite differences approximations ∇_x^h and ∇_y^h are defined on the entire tensors. Indeed, we apply a padding technique during their computation. Hence, the values exist everywhere on the tensors. See Fig. 15 for a representation of the results of padding techniques.

Finally, we introduce the relative error of each order, $\mathcal{L}_p$, $p = 0, 1, 2$, given by

$$\mathcal{L}_i\left(U_{\text{true}}; U_\theta\right) = \frac{1}{N_{\text{data}}} \sum_{n=0}^{N_{\text{data}}} \sqrt{\frac{\mathcal{E}_i(u^n_{\text{true}}; u^n_\theta)}{\mathcal{N}_i(u^n_{\text{true}})}}\,. \tag{8}$$

Remark. In (7), we can remark that the loss is computed with respect to u^n_{true} and not w^n_{true}. This way of computing the loss does not mean that our FNO will predict u^n_θ. It only means that we will predict a solution w_θ such that, multiplied by input function ϕ_h and added to g_h, the result will be close to u^n_{true}. Moreover, we use the first and second derivatives in the loss since it leads to better results than using only $\mathcal{L}_0$ as the loss function (see Appendix C).

Relative L^2 norms. We will also compute two types of L^2 relative norms to evaluate the performance of our method. The first one will be used to compute the errors with respect to fine standard finite element solutions u_{ref}. This norm is defined by

$$\frac{\|\Pi_{\Omega_{\text{ref}}} u_\theta - u_{\text{ref}}\|_{0,\Omega_{\text{ref}}}}{\|u_{\text{ref}}\|_{0,\Omega_{\text{ref}}}} = \sqrt{\frac{\int_{\Omega_{\text{ref}}} (\Pi_{\Omega_{\text{ref}}} u_\theta - u_{\text{ref}})^2 \,\mathrm{d}x}{\int_{\Omega_{\text{ref}}} u^2_{\text{ref}} \,\mathrm{d}x}}\,, \tag{9}$$

where $\Pi_{\Omega_{\text{ref}}}$ denotes an approximation of the L^2-orthogonal projection on the reference domain Ω_{ref} (fine conformed mesh of Ω).

The second norm, that will be used to compute the error between a FNO solution and a ground truth solution, is given by

$$\sqrt{\frac{\mathcal{E}_0(u_{\text{true}}; u_\theta)}{\mathcal{N}_0(u_{\text{true}})}}\,, \tag{10}$$

where $u_\theta = \phi_h \mathcal{G}_\theta(\phi_h, f_h, g_h) + g_h$ and $u_{\text{true}} = \phi_h w_h + g_h$.

4 Numerical results

Let us now illustrate the efficiency of our technique by numerical test cases. We will first consider the case of parametric domains, using varying elliptic domains to illustrate the accuracy and the fastness of the method compared to four other methods. Then, we will extend our study to more complex shapes.

We will fix $n_d = 20$ (number of neurons acting on each node), $n_Q = 128$ (number of neurons in the first layer of the projection Q_θ) and $m = 10$ (the number of low frequencies considered in the low pass filter). This choice is motivated by the results of Appendix B. In the data produced by ϕ-FEM, the parameter σ_D is fixed to 1.

Training, validation, and test samples We will detail three types of samples. The one used to train our models (optimization of the parameters) is the training sample. The validation sample will be used to evaluate the accuracy of a model for several validation steps during the training. This sample will help us to determine the best model of a training, avoiding overfitting. Finally, the term testing data will be used for new data, generated in the same way as the training and validation samples but not used at all during the training of the models.

Implementation details All the simulations were executed on a laptop with an `Intel Core i7-12700H CPU`, 32Gb of memory, and an `NVIDIA RTX A2000 GPU` with 8Gb of memory.

The data were generated using the python finite element library *FEniCS*[1] and the FNO is implemented[3] using the *Pytorch*[22] library. Moreover, we will use an ADAM optimizer with an initial learning rate $\alpha = 0.005$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-7}$ to train the operator (see Appendix D Algo. 1). During training, the learning rate is reduced when the loss on the validation sample does not decrease over several epochs. See the numerical implementation of the learning rate scheduler[4] for more details.

The algorithm of the training loop is presented in Appendix D Algo. 2. The chosen hyperparameters are described in Appendix B.

4.1 The case of varying ellipses

Let us first consider the case of elliptic domains. We will solve the equation

$$\begin{cases} -\Delta u &= f\,, \quad \text{in } \Omega\,, \\ u &= g\,, \quad \text{on } \Gamma\,. \end{cases} \tag{11}$$

The level-set functions will be given by

$$\phi_{(x_0,y_0,l_x,l_y,\theta)}(x,y) = \frac{((x-x_0)\cos(\theta)+(y-y_0)\sin(\theta))^2}{l_x^2} + \frac{((x-x_0)\sin(\theta)-(y-y_0)\cos(\theta))^2}{l_y^2} - 1\,, \tag{12}$$

[3] All the codes and datasets used in this paper are available at `https://github.com/KVuillemot/PhiFEM_and_FNO`.

[4] `https://github.com/KVuillemot/PhiFEM_and_FNO/tree/main/Ellipses/main/scheduler.py`

with

$$x_0,\ y_0 \sim \mathcal{U}([0.2, 0.8])\,, \quad l_x,\ l_y \sim \mathcal{U}([0.2, 0.45]) \quad \text{and } \theta \sim \mathcal{U}([0, \pi])\,. \tag{13}$$

The level-set (12) represents the equation of an ellipse centered in (x_0, y_0) of semi-major axis l_x and semi-minor axis l_y, rotated by an angle θ around the center of the ellipse, as illustrated for example in Fig. 1 (left). In addition, since the domain must lie entirely within the unit square, we fix additional constraints on the parameters of $\phi_{(x_0,y_0,l_x,l_y,\theta)}$ to ensure that this condition is satisfied (see the GitHub repository[5] for the implementation of these constraints).

The random functions f and g of (11) are given by

$$f_{(\mu_0,\mu_1,\sigma_x,\sigma_y)}(x, y) = 25 \exp\left(-\frac{(x-\mu_0)^2}{2\sigma_x^2} - \frac{(y-\mu_1)^2}{2\sigma_y^2}\right)\,, \tag{14}$$

and

$$g_{(\alpha,\beta)}(x, y) = \alpha\left((x-0.5)^2 - (y-0.5)^2\right)\cos\left(\beta y \pi\right)\,, \tag{15}$$

where $(\mu_0, \mu_1) \sim \mathcal{U}([0.2, 0.8]^2 \cap \{\phi < -0.15\})$, $\sigma_x, \sigma_y \sim \mathcal{U}([0.15, 0.45])$ and $\alpha, \beta \sim \mathcal{U}([-0.8, 0.8])$.

We generate a set of data of size 1800, split into a training set of size 1500 and a validation set of size 300. The training set is then divided into batches of size 32 (number of data considered in the loss function in one computation of the gradient) at each of the 2000 epochs (number of loops over all the batches), as explained in Algorithm 2. This test case will be divided into 3 main parts: we will first study the results during the training on the validation sample, which will lead to the choice of an "optimal" model. Then, we will focus on the results on a first test sample, to compare the results of the models during the training with a ϕ-FEM solutions sample. Finally, in the third part, we will compare our ϕ-FEM-FNO method to ϕ-FEM, a standard FEM solver, a FNO trained with interpolated standard FEM solutions, and Geo-FNO (see [14]).

Results on the validation sample We represent in Fig. 5 (left) the evolution of the loss function $\mathcal{L}$ and of each $\mathcal{L}_i$ on a random subset of the training dataset and in Fig. 5 (right) on the validation dataset, both of size 300 . During the training, we select the model minimizing the loss function $\mathcal{L}$ on the validation set. This model will be considered to be the optimal model returned by the training and will be used in the third part of this test case. To check the optimality of this model, we represent with boxplots the evolution of the L^2 errors given by formula (10) of few models of the training, on the validation sample, in Fig. 6. This representation illustrates that the selected model seems to be the best one. We also represent in Fig. 7 the data and the prediction minimizing the L^2 error among the validation data.

Validation of the model of a first test dataset. We will now focus on a second important point: the error of the models on a test dataset, in the norm (10). This will allow us to verify that the operator is well trained and that on new data, it behaves approximately as on the validation data. This will also be a final check of the optimality of the selected best model. For that, we will consider 10000 new data and compute the error in the norm defined by (10), for some saved models of the training, including the optimal one. The results in Fig. 8 confirm that the selected optimal model seems to be the best of the training. In addition, we also represent in Fig. 9 the evolution of the minimal, maximal, average errors, and the standard deviation of

[5] `https://github.com/KVuillemot/PhiFEM_and_FNO/blob/main/Ellipses/main/prepare_data.py`

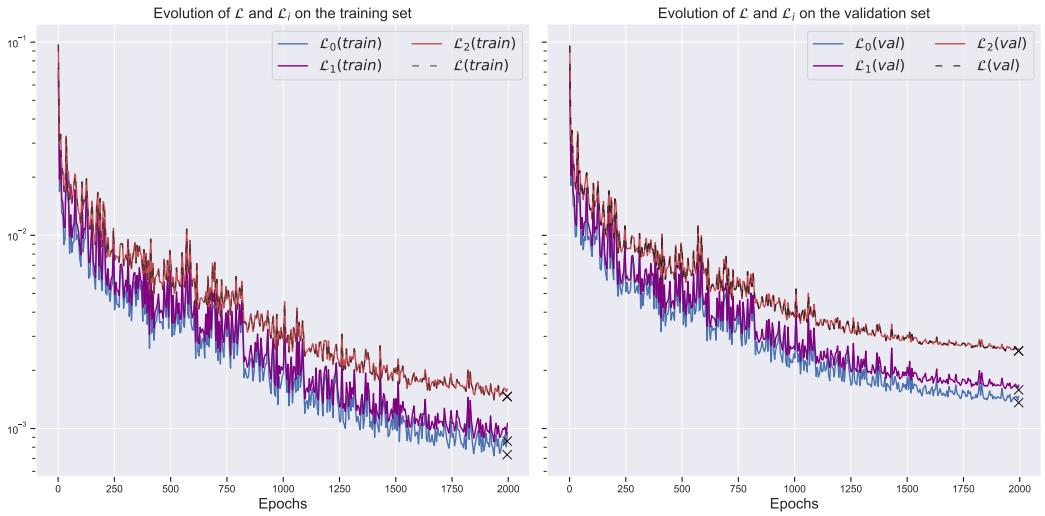

Figure 5: **Test case 1.** Results of the training for the first test case. Evolution of the $\mathcal{L}_i$ for $i = 0, 1, 2$ and of the loss function $\mathcal{L}$. Left: results on the training sample. Right: results on the validation sample.

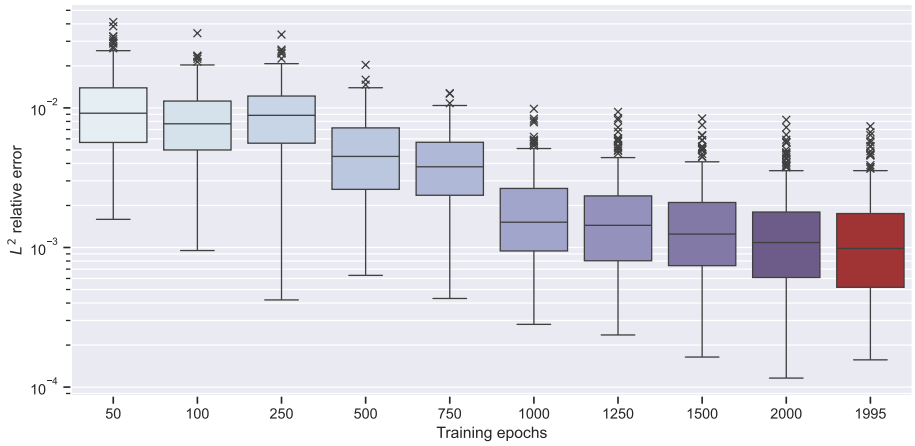

Figure 6: **Test case 1.** Evolution of the (10) errors on the validation set at different steps of the training. The optimal model is represented in red.

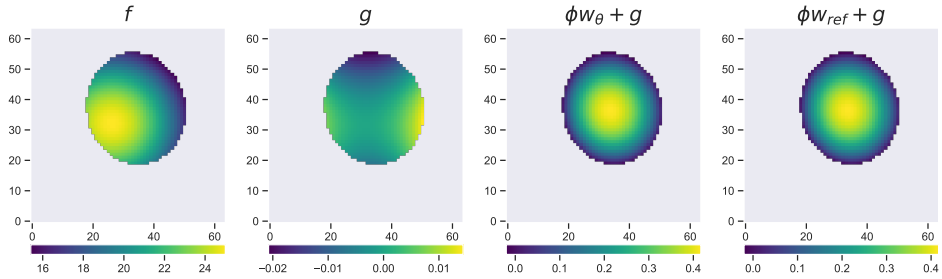

Figure 7: **Test case 1.** Example of result among the validation sample after 1995 epochs, with an error in the norm (10) of 1.5×10^{-4}.

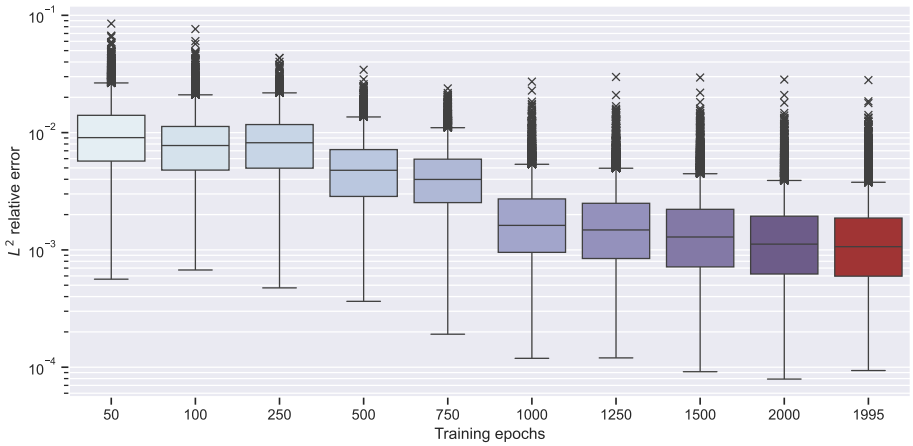

Figure 8: **Test case 1.** Evolution of the (10) errors on the first test sample at different steps of the training.

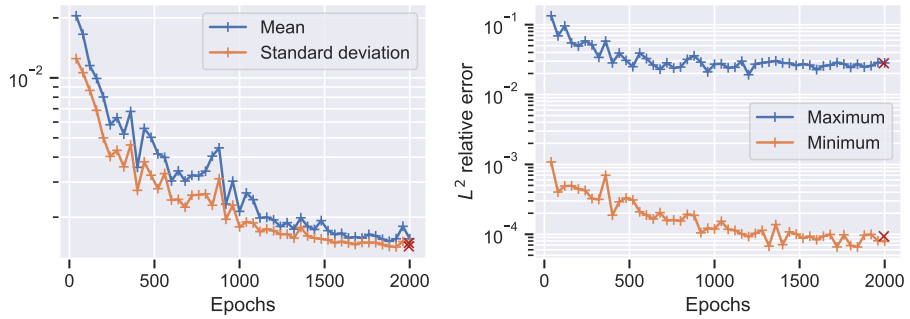

Figure 9: **Test case 1.** Evolution of the (10) errors on the test dataset at different steps of the training. Left: in blue, the average value of the L^2 relative error. In orange, the standard deviation of the error. Right: in blue, the maximal value of the L^2 relative error. In orange, the maximal value. Red crosses correspond to the values for the best model.

the errors, on the same test dataset, with respect to the training epochs. This verification leads to the conclusion that for this test case, 1500 training data seems to be sufficient to describe the space of parameters (and thus of data) and to obtain a precise operator. Moreover, we can see in the different representations that it could have been interesting to continue training the operator since the errors seem to continue decreasing at 2000 epochs. However, we tried to train the operator over 3000 epochs and the results did not significantly improve anymore after 2000 epochs.

Comparison of Standard FEM, ϕ-FEM, ϕ-FEM-FNO, "Standard-FEM-FNO" and Geo-FNO Let us then move to the most important numerical results to illustrate the interest of our ϕ-FEM-FNO technique. In this part, we will compare ϕ-FEM-FNO with four other approaches, to show the effectiveness of our method:

- ϕ-FEM-FNO: we call the previous optimal model with 1500 training data of resolution

64×64 (corresponding to a cell size $h \approx 0.022$), taking, as said before, $\sigma_D = 1$ and $\mathbb{P}^1$ finite elements;

- ϕ-FEM: we apply the operator $\mathcal{G}^\dagger$, with background meshes of resolution 64×64 (corresponding to a cell size $h \approx 0.022$), taking $\sigma_D = 1$ and $\mathbb{P}^1$ finite elements;
- Standard FEM: we use a standard $\mathbb{P}^1$ finite element method to solve the problems on meshes with cell size $h \approx 0.022$, corresponding to the resolution used for the other approaches;
- Standard-FEM-FNO: we use a FNO trained with standard $\mathbb{P}^1$ FEM solutions on meshes of sizes $h \approx 0.022$, interpolated on Cartesian grids of size 64×64 as data. The loss function used to train Standard-FEM-FNO is the relative H^2 norm and the operator is trained during 2000 epochs.
- Geo-FNO: we have trained a Geo-FNO, adapting the approach of [14] to our test case, using as input of the operator a set of 2600 points and the values of f and g at each of these points. We used 2600 points to obtain an average cell size close to 0.02. The operator has been trained during 2000 epochs, using the L^2 relative norm. The difference with the other approaches is that the cell sizes are not constant between two meshes since the constant is the number of points defining the meshes. Hence, in Fig. 11, we consider the average size of cell to compare the methods.

The three ML-based methods (ϕ-FEM-FNO, Standard-FEM-FNO, and Geo-FNO) have been trained using the same dataset, adapted to each method (i.e., during the data generation steps, we consider the same set of parameters $(x_0, y_0, l_x, l_y, \theta, \mu_0, \mu_1, \sigma_x, \sigma_y, \alpha, \beta)$), and the same hyperparameters during training (see Appendix B).

Remark (Implementation aspect.). To compare our method with a standard finite element method, we need to construct conforming meshes using the values of the level-set function ϕ. The creation of a mesh using a level-set function is not directly possible with *FEniCS*. Thus, we need to create such meshes manually. For this step, we use *pymedit*[6] which uses the Mmg platform[7]. We refer the reader to the GitHub repository[8] for details on the installation and two examples of use.

To compare ϕ-FEM-FNO with the other methods, we generate a new test sample of size 300 and use the best models of each ML-based method to predict the solutions to these problems. The solutions of each method are projected on a reference fine mesh where the size of cells $h_{\text{ref}} \approx 0.002$, as illustrated in Fig. 10. The errors are then computed in the norm (9) using a fine standard finite element solution as a reference solution. Fig. 11 (top left) illustrates that the trained ϕ-FEM-FNO after 1995 epochs approaches the precision of a standard method, and is sometimes even better. Moreover, ϕ-FEM is approximately only 5 times more precise than the ϕ-FEM-FNO operator. In addition, ϕ-FEM-FNO is approximately 2.5 times and 10 times more precise than Standard-FEM-FNO and Geo-FNO respectively.

In Fig. 11 (top right), we illustrate the fastness of ϕ-FEM-FNO, compared to standard FEM and ϕ-FEM. For that, we change the size of the input dataset and apply each of the three methods. As we can make several ϕ-FEM-FNO calls in parallel on the GPU, computation

[6] `https://pypi.org/project/pymedit/`
[7] `https://www.mmgtools.org/`
[8] `https://github.com/KVuillemot/PhiFEM_and_FNO/blob/main/install_and_use_mmg.md`

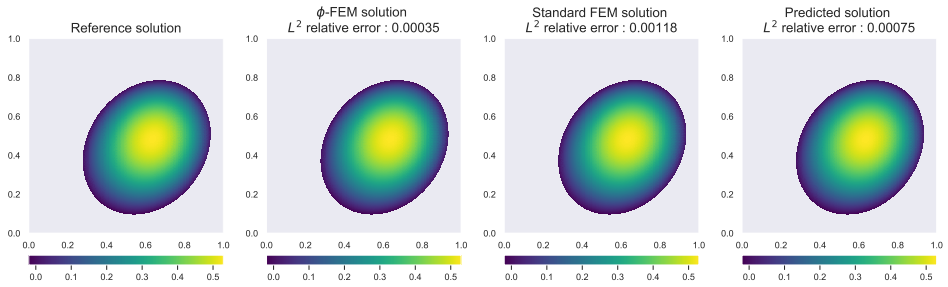

Figure 10: **Test case 1.** Outputs of ϕ-FEM, standard FEM and ϕ-FEM-FNO. The prediction of ϕ-FEM-FNO is made using the model after 1995 epochs.

time does not change much with the number of considered data. Hence, in comparison, the time required to solve multiple problems is much lower with ϕ-FEM-FNO than with the other methods.

Moreover, it is also interesting to locate the precision of the predictions of the FNO-based methods on the convergence curves of the two finite element methods. For that, we generate 20 new data, and we compute reference solutions to all these problems using a fine standard finite element, with $h \approx 0.001$ for each problem. We then compute ϕ-FEM and standard FEM for different cell sizes h. We represent the L^2 errors of the two methods with respect to the mesh size h (Fig. 11, bottom left) and with respect to the computation times (Fig. 11, bottom right). These two representations also justify our choice of using ϕ-FEM as a finite element solver. Indeed, if the standard FEM had given better results than ϕ-FEM on test problems, ϕ-FEM would not necessarily have been the best solver to generate data. Moreover, we represent the errors of ϕ-FEM-FNO and standard-FEM-FNO for the same problems, called on data of resolutions 64×64, and of Geo-FNO for the same problems, using 2600 points. In Fig. 11 (bottom left), blue and orange crosses are the mean error for different sizes h. The filled-in areas correspond to intervals around the mean points, whose lengths are the standard deviations of the errors for each h. The red cross (resp. purple cross) is the average error of ϕ-FEM-FNO (resp. Standard-FEM-FNO) for the same dataset, and the intervals are constructed in the same way as the blue and orange filled-in areas. Finally, in brown, we represent the same for Geo-FNO, using the average h among the 20 resolutions as the x-axis.

Finally, in Fig. 11 (bottom right), we illustrate the biggest advantage of our method, by representing the errors with respect to the computation times for each method. For that, we compute the average time and error for ϕ-FEM and standard FEM for each size h. We represent these averages with orange and blue crosses. Moreover, the blue and orange filled-in areas correspond to the same intervals as in Fig. 11 (bottom left). We represent in blue and orange the intervals centered in the average computation time for each size, whose lengths are the standard deviations of the computation times. Finally, we represent with red, purple, and brown ellipses the same things for ϕ-FEM-FNO, standard-FEM-FNO, and Geo-FNO: the centers of the ellipses are the average computation times and average errors, the heights of the ellipses are the standard deviations of the errors, and their widths are the standard deviations of the computation times. For ϕ-FEM, the computation times are the sum of the times needed to select and to build Ω_h, the assembly of the finite element matrix, and the resolution of the linear system. For standard FEM, we add the mesh construction time, the assembly time, and the resolution time. Finally, for the FNO-based methods, the computation times are only

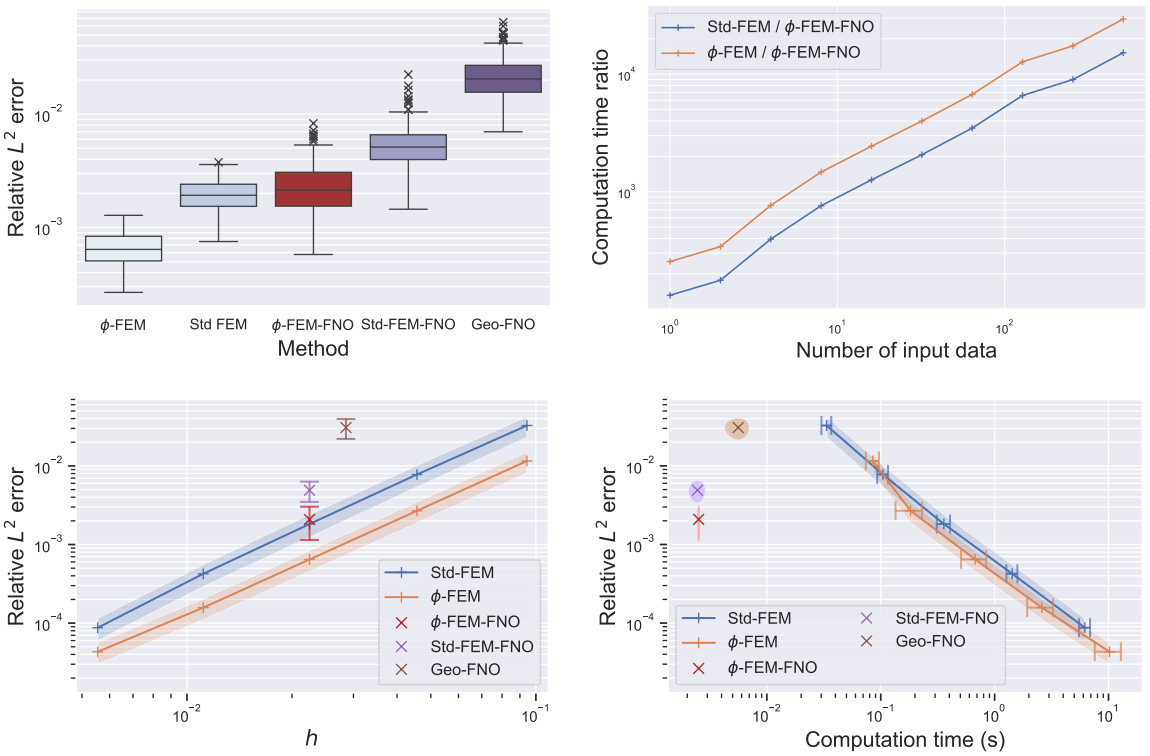

Figure 11: **Test case 1.** Top left: Relative L^2 errors of the methods (from left to right: ϕ-FEM, standard method, ϕ-FEM-FNO, Standard-FEM-FNO, Geo-FNO). Top right: computation times of the methods. Bottom left: Relative L^2 errors, with respect to h. Bottom right: Relative L^2 errors, with respect to the computation times.

the time of one call of the models.

4.2 The case of varying complex shapes

We now move to a more complex case. Indeed, for the first test case, we have considered the case of very smooth and convex shapes. However, since the objective of ϕ-FEM is to work on medical images, it is interesting to work on more complex shapes. Indeed, our idea is to provide a method that can give fast and precise results on such domains. To create complex geometries, we use random level-set functions generated using Fourier series, oscillating in the box $(0,1)^2$, such that the level-set functions ϕ are defined by 9 Fourier modes as follows

$$\phi(x,y) = 0.4 - \sum_{k=1}^{3}\sum_{l=1}^{3} \alpha_{kl} \sin(k\pi x)\sin(l\pi y), \tag{16}$$

where $\alpha_{kl} \sim \mathcal{U}([-1,1])$. Moreover, we add a few constraints on the constructed domain $\Omega = \{\phi < 0\}$: it must not be too close to the boundary, too small, and must be a connected domain. If these conditions are not verified, we create another level-set function ϕ by creating a different set of parameters. We represent 4 examples of such level-set functions and given domains, in Fig. 12. Finally, we solve the equation (11), using f and g defined by (14) and (15) respectively.

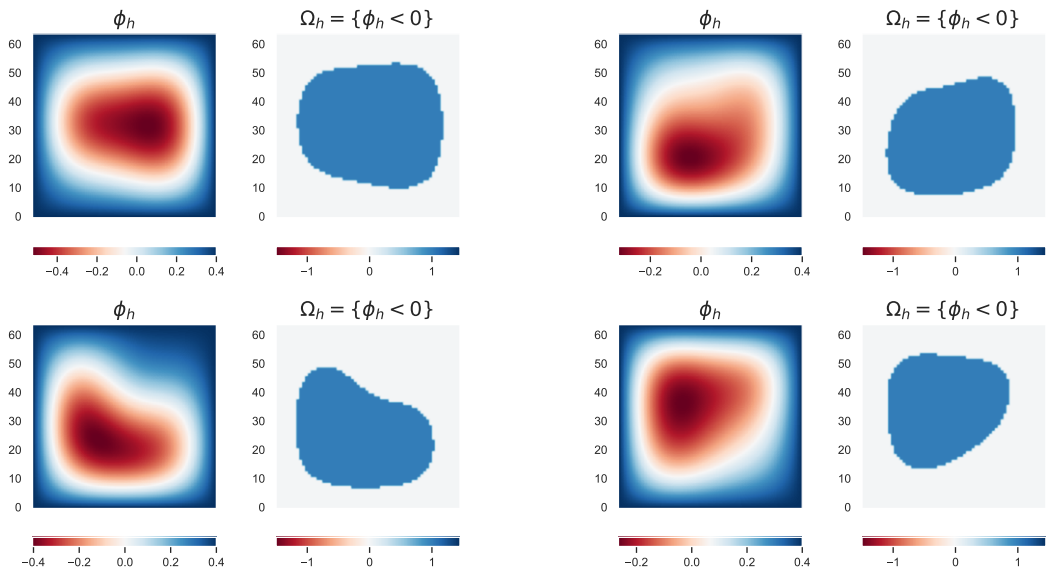

Figure 12: **Test case 2.** Examples of considered level-set functions and domains, using (16).

Remark (Preparation of the dataset.). After the step of data generation, we must observe a step of data preparation. Indeed, some shapes are not well treated with ϕ-FEM such as not smooth enough shapes or some shapes with holes. Hence, we perform a step of selection of the data: we compute the maximum of the residues (i.e. the maximal value of $\Delta^h(\phi_h w_h + g_h) + f_h$, on Ω_h, where $\Delta^h u_h = \nabla_x^h \nabla_x^h u_h + \nabla_y^h \nabla_y^h u_h$) to ensure that the derivatives and second derivatives do not explode. These restriction is done only during the generation of the training and validation data.

For this test case, the previous constraint leads to selecting a sample of size 2637 over a dataset of size 3000. This dataset is then divided into a training set of size 2200 and a validation sample of size 437. Finally, the operator is trained during 2000 epochs, with the same hyperparameters as in the first test case.

The evolution of $\mathcal{L}_i$, $i = 0, 1, 2$ and of $\mathcal{L}$ are presented in Fig. 13. As for the first test case, at the end of the training, we consider as the optimal model the one minimizing the loss function on the validation sample, here, the one after 1984 epochs.

To compare the standard FEM, ϕ-FEM, and ϕ-FEM-FNO, we consider the best ϕ-FEM-FNO model, i.e. after 1984 epochs. We compute the errors with respect to a fine standard finite element solution, as in the first test case, on a test dataset of size 300 (that do not necessarily respect the above constraint on the residuals used during the training data generation). The results are given in Fig. 14 (top). The predictions of ϕ-FEM-FNO are then close to the solutions of the standard FEM and are sometimes even better. However, we can remark some atypical points. These points correspond to very complex shapes, far from the shapes of the training dataset. Increasing the size of the training dataset can correct this issue but our idea was to illustrate that in general, our approach gives precise results without too much training data. Moreover, to illustrate the advantage of our method compared to finite element solvers, we represent, as in the first test case (Fig. 11), the convergence curves of standard FEM and ϕ-FEM, adding the errors of the ϕ-FEM-FNO predictions. In Fig. 14 (bottom left), we represent the error against the cell size h and in Fig. 14 (bottom right), against the computation times.

Figure 13: **Test case 2.** Results of the training for the first test case. Evolution of the $\mathcal{L}_i$ for $i = 0, 1, 2$ and of the loss function $\mathcal{L}$. Left: results on the training sample. Right: results on the validation sample.

The computation time of the standard method is the sum of the times needed to generate a mesh from a level-set function, to assemble and to solve the linear system. For ϕ-FEM, it is the sum of the times needed to select the cells of Ω_h and of Ω_h^Γ, and the times to assemble and solve the linear system. Finally, the computation time of ϕ-FEM-FNO corresponds to the time to call the model for one data. Hence, Fig. 14 (right) illustrates very well the interest of our method: for approximately the same error as a standard finite element solver, the computation is approximately 150 times faster.

5 Conclusion and future works

We have shown on two test cases that after training, our ϕ-FEM-FNO can compute faster than standard finite element methods, ϕ-FEM, an interpolate-FEM approach or Geo-FNO on several problems. Moreover, we have illustrated that these results can be obtained using small amounts of training data, even for complex cases with big variations of geometries.

A number of perspectives remain for future research. It would be interesting to extend the results to other problems since ϕ-FEM schemes have been written and studied theoretically and numerically (Neumann conditions, mixed conditions, linear elasticity, Stokes, time-dependent PDEs, ...). In future works, we can compare the results with other neural networks like for example, CNN ([21]). Some enhancements like factorization of the layer for the operator ([27]) could be made in order to accelerate the training and increase the accuracy.

Moreover, in the future, we can extend our results to the case of hyperelastic materials as in [20], and implement the method in the DeepPhysX project[9]. Furthermore, another interesting point would be to extend our method to more realistic scenarios, considering real medical images and more realistic forces and boundary conditions. Finally, we can also imagine extending our method to the case of $\mathbb{P}^k$ functions, using the degrees of freedom values instead

[9] `https://mimesis.inria.fr/project/deepphysx/`

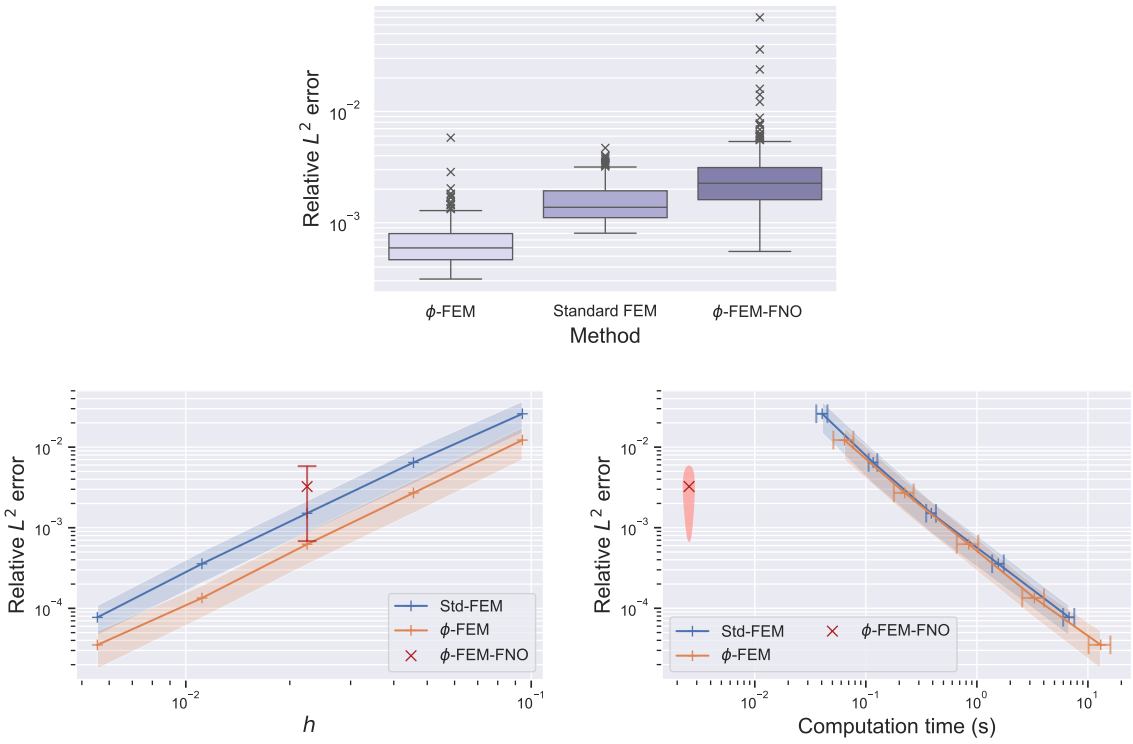

Figure 14: **Test case 2.** Top: relative L^2 errors (9) of the three methods on similar resolutions, with 300 data. Bottom left: relative L^2 errors (9) against resolution. Right: relative L^2 errors (9) of the methods against computation time (in seconds).

of nodal values for the data generation and thus predicting the values of the solution at each $\mathbb{P}^k$ degrees of freedom.

Finally, to represent more complex and general forces, one can train an FNO using Gaussian forces. Then, one can decompose a new random force in a sum of Gaussian distributions and use the trained model on each one of the Gaussian forces. Thanks to GPU parallelization, each prediction can be done simultaneously, and it only remains to sum the predictions to obtain the final result.

6 Acknowledgment

The authors were supported by the ANR project JCJC 22-CE46-0003.

References

[1] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. Rognes, and G. Wells. Archive of numerical software: The fenics project version 1.5. *University Library Heidelberg*, 2015.

[2] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart. Model reduction and neural networks for parametric pdes. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

[3] S. Cotin, M. Duprez, V. Lleras, A. Lozinski, and K. Vuillemot. ϕ-FEM: an efficient simulation tool using simple meshes for problems in structure mechanics and heat transfer. In *Partition of Unity Methods (Wiley Series in Computational Mechanics) 1st Edition*. Wiley, Nov. 2022.

[4] R. A. DeVore. The theoretical foundation of reduced basis methods. *Model reduction and approximation: theory and algorithms*, 15:137, 2017.

[5] M. Duprez, V. Lleras, and A. Lozinski. A new ϕ-FEM approach for problems with natural boundary conditions. *Numer. Methods Partial Differential Equations*, 39(1):281–303, 2023.

[6] M. Duprez, V. Lleras, and A. Lozinski. ϕ-FEM: an optimally convergent and easily implementable immersed boundary method for particulate flows and Stokes equations. *ESAIM Math. Model. Numer. Anal.*, 57(3):1111–1142, 2023.

[7] M. Duprez, V. Lleras, A. Lozinski, and K. Vuillemot. ϕ-FEM for the heat equation: optimal convergence on unfitted meshes in space. *Comptes Rendus. Mathématique*, 361:1699–1710, 2023.

[8] M. Duprez and A. Lozinski. ϕ-FEM: a finite element method on domains defined by level-sets. *SIAM J. Numer. Anal.*, 58(2):1008–1028, 2020.

[9] R. Enjalbert, A. Odot, and S. Cotin. mimesis-inria/deepphysx: 22.06, Dec. 2022.

[10] A. Ern and J.-L. Guermond. *Theory and practice of finite elements*, volume 159. Springer, 2004.

[11] T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb. Can physics-informed neural networks beat the finite element method? *arXiv preprint arXiv:2302.04107*, 2023.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces. *CoRR*, abs/2108.08481, 2021.

[14] Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries, 2022.

[15] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[16] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations, ICLR 2021.

[17] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, A. Stuart, K. Bhattacharya, and A. Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020.

[18] L. Lu, P. Jin, and G. E. Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[19] M. Nastorg, M.-A. Bucci, T. Faney, J.-M. Gratien, G. Charpiat, and M. Schoenauer. An Implicit GNN Solver for Poisson-like problems. working paper or preprint, Feb. 2023.

[20] A. Odot, R. Haferssas, and S. Cotin. DeepPhysics: a physics aware deep learning framework for real-time simulation. working paper or preprint, Mar. 2023.

[21] K. O'Shea and R. Nash. An introduction to convolutional neural networks, 2015.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[23] M. Raissi, P. Perdikaris, and G. 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, Feb. 2019.

[24] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[25] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[26] N. Sukumar and A. Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.

[27] A. Tran, A. Mathews, L. Xie, and C. S. Ong. Factorized fourier neural operators. In *The Eleventh International Conference on Learning Representations*, 2023.

[28] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1457–1466, 2020.

[29] B. Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[30] J. Zhao, R. J. George, Z. Li, and A. Anandkumar. Incremental spectral learning in fourier neural operator, 2023.

[31] Y. Zhu and N. Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

A Standardization of the data

To improve the performance of our FNO, since the data can have very different values, we have decided to standardize the input and output data, as in [16]. The standardization is applied independently channel by channel of X, i.e. for F, Φ and G. For each channel C of X, denoting by C^{train} the training part of the data-set corresponding to the channel C, the associated standardized channel is given by

$$N_C(C) = \left(\frac{C - \text{mean}(C^{\text{train}})}{\text{std}(C^{\text{train}})} \right) . \tag{17}$$

Thus $N(X) = (N_F(F), N_\phi(\Phi), N_G(G))$.

The unstandardization function N^{-1} is given by

$$N^{-1}(Y) = Y \times \text{std}(Y^{\text{train}}) + \text{mean}(Y^{\text{train}}) , \tag{18}$$

where Y denotes the output of the FNO and Y^{train} is the vector composed of the training ground truth solutions.

B Calibration of the hyperparameters.

In this section, we justify our choice of hyperparameters. For that, we have performed in Table 2 a random grid search over the following hyperparameters :

- The hidden dimension, n_d. We have tested with $n_d = 10$, 15, 20 and 25.

- The number of Fourier modes m. We have tested to train an operator with 5, 10, 15 or 20 Fourier modes. It is important to remark that there is an inescapable constraint on this hyperparameter: the number of Fourier modes must be strictly less than the number of nodes divided by two. So, if we increase the number of Fourier modes to more than 31, we will not be able to train or use our FNO on meshes of size 64×64. Moreover, as explained in [30], choosing a too high number of modes can lead to overfitting during the training.
- The initial value of the learning rate α, chosen in $\{0.01, 0.005, 0.002, 0.001\}$.
- The batch size, taken equal to 32, 64, or 128.
- The L^2 regularization parameter λ, chosen in $\{0, 0.01, 0.001, 0.0001\}$.
- The padding technique and the proportion of padding: we have tested three techniques: apply no padding, constant padding, and reflective padding. The differences between the three techniques for a simple example are illustrated in Fig. 15 (see `https://www.tensorflow.org/api_docs/python/tf/pad` for the original version). We have tested with 0.025 and 0.05 as padding proportion, i.e. adding respectively 1 and 3 pixels at each side of the images.

1	2	3
4	5	6
7	8	9

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	2	3	0	0
0	0	4	5	6	0	0
0	0	7	8	9	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

9	8	7	8	9	8	7
6	5	4	5	6	5	4
3	2	1	2	3	2	1
6	5	4	5	6	5	4
9	8	7	8	9	8	7
6	5	4	5	6	5	4
3	2	1	2	3	2	1

Figure 15: Different versions of padding techniques. Left: the original image (equivalent to the one with no padding). Center: the result of a constant padding of the original image. Right: the result of a reflective padding.

To determine an optimal choice of hyperparameters we have randomly chosen several combinations of them, among the sets of values given previously. For each of these combinations, we have performed training over 2000 epochs, using the dataset of the first test case of Section 4. For each training, we have computed two metrics on the validation sample: the relative L^2 error and the relative H^2 error (i.e. the loss function). We present in Table 2 the results of the best model for each set of parameters. This best model has been obtained by taking the minimal relative L^2 error over the training epochs. The set of parameters giving the best results among the tested parameters is on the top line. We present the 20 best results. Moreover, in Table 2, the column "try" corresponds to the try with the set of parameters. Indeed, we have performed at most 3 tries for each set, to see how much the errors depend on the random initialization of the weights and if there was one set of parameters that would have been better for different tries.

This random search leads to four initial guesses of optimal hyperparameters (the red lines of Table 2). We have then fine-tuned the hyperparameters around the previous initial

L^2 error	α	m	n_d	Batch size	λ	padding proportion	padding mode	try	H^2 error	epoch duration (s)
0.0013	0.005	10	20	32	0.001	0.05	Reflect	1	0.0025	2.4
0.0014	0.002	20	25	32	0.001	0	/	1	0.0026	2.7
0.0016	0.005	15	20	32	0.001	0.025	Constant	3	0.0034	2.7
0.0017	0.005	20	25	32	0.0001	0	/	3	0.0031	3.0
0.0017	0.002	10	25	32	0.0001	0.05	Reflect	3	0.0030	3.1
0.0018	0.002	10	15	64	0.0001	0.025	Constant	2	0.0040	1.8
0.0018	0.005	5	25	64	0.01	0.025	Constant	1	0.0032	2.4
0.0018	0.001	20	20	64	0.0001	0	/	1	0.0042	2.0
0.0018	0.002	15	15	64	0.0001	0	/	3	0.0038	1.7
0.0019	0.002	20	15	64	0.0001	0.05	Constant	2	0.0040	1.9
0.0019	0.002	10	15	32	0.0001	0.05	Constant	2	0.0035	2.5
0.0020	0.01	5	25	64	0.01	0	/	1	0.0034	2.0
0.0021	0.01	10	25	64	0.0001	0.05	Constant	3	0.0038	2.4
0.0021	0.001	5	10	32	0.001	0	/	1	0.0041	1.8
0.0021	0.001	15	20	128	0.0001	0.025	Reflect	1	0.0051	1.8
0.0021	0.002	5	25	64	0	0	/	3	0.0045	2.2
0.0022	0.002	20	20	32	0.01	0.05	Reflect	2	0.0044	2.8
0.0022	0.001	15	20	128	0.0001	0.05	Reflect	1	0.0051	1.9
0.0024	0.002	10	20	32	0.01	0	/	1	0.0045	2.6
0.0024	0.005	15	15	128	0.0001	0.025	Constant	3	0.0049	1.4
0.0024	0.001	10	10	32	0.01	0.025	Constant	1	0.0049	2.3

Table 2: Results of the hyperparameters random search.

guesses, finally choosing the ones giving the best compromise between H^2 error, L^2 error, and computation times. These parameters are $\alpha = 0.005$, $m = 10$, $n_d = 20$, $\lambda = 0.001$, a batch size of 32, and a reflective padding with a proportion of 0.05, leading to the optimal results given in the first line of Table 2.

C Choice of the loss

To justify our choice of including the first and second derivatives in the loss function during training, we compare the results of three training using the 3 following losses :

$$\mathcal{L}_0\left(U_{\text{true}}; U_\theta\right) := \frac{1}{N_{\text{data}}} \sum_{n=0}^{N_{\text{data}}} \sqrt{\frac{\mathcal{E}_0(u^n_{\text{true}}; u^n_\theta)}{\mathcal{N}_0(u^n_{\text{true}})}},$$

$$\mathcal{L}_{H^1}\left(U_{\text{true}}; U_\theta\right) := \frac{1}{N_{\text{data}}} \sum_{n=0}^{N_{\text{data}}} \sqrt{\frac{\mathcal{E}_0(u^n_{\text{true}}; u^n_\theta) + \mathcal{E}_1(u^n_{\text{true}}; u^n_\theta)}{\mathcal{N}_0(u^n_{\text{true}}) + \mathcal{N}_1(u^n_{\text{true}})}},$$

and the loss $\mathcal{L}$ defined by (7).

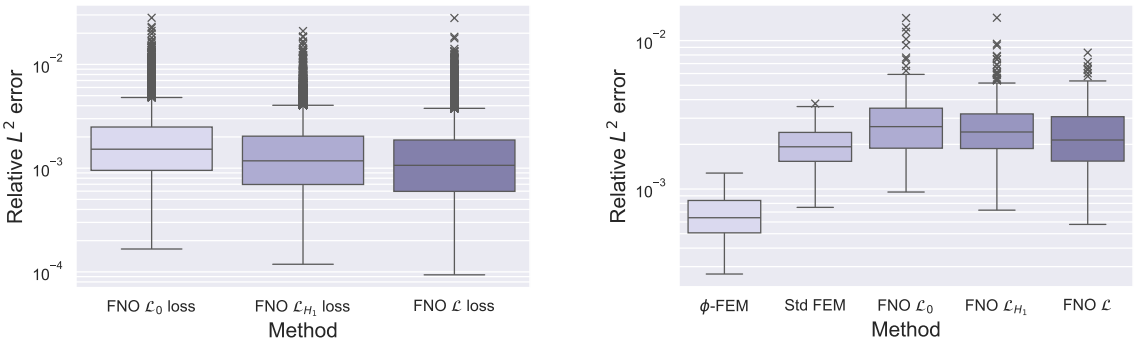

Figure 16: Left: Errors of prediction on 10000 data in the norm (10) (with respect to ϕ-FEM solutions). Right: Errors on 300 data in the norm (9) (with respect to fine standard FEM solutions).

To compare the three losses, we will predict the results of a dataset of size 10000 using the best model of each training and compute the L^2 relative errors with respect to a ground truth ϕ-FEM solution (i.e. in the norm (10). The results are presented in Fig. 16 (left), and confirm that using the second derivatives in the loss function leads to better results. However, since the differences between the three losses are small, we cannot say that it is mandatory to use $\mathcal{L}$ instead of $\mathcal{L}_0$. We also compare the three versions on a test dataset of 300 data, on which we compute the errors with respect to fine standard reference solutions. The results in Fig. 16 (right) lead to the same conclusion.

D ADAM and training loop algorithm

We present the details of the considered ADAM optimizer in Algorithm 1. In Algorithm 2 we denote (F^i, Φ^i, G^i) a batch of data. The batches are randomly chosen such that $F^i = (f_h^k)_{k\in K_i}$, $\Phi^i = (\phi_h^k)_{k\in K_i}$, $G^i = (g_h^k)_{k\in K_i}$ with K_i a collection of random indices of data and $i \in \{1, \dots, \text{number of batches}\}$. The sets K_i are constructed such that $K_i \cap K_j = \emptyset$ for $i \neq j$.

Algorithm 1 ADAM optimizer step.

Initialisation : t, θ_{t-1}, β_1, β_2, ε, m_{t-1}, v_{t-1}.

Compute the gradient : $g_t \leftarrow \nabla f(\theta_{t-1})$

Momentum update :

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1)\cdot g_t\,, \qquad v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2)\cdot g_t \cdot \bar{g}_t$$

Bias correction :

$$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}\,, \qquad \hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$$

Parameters update :

$$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t}+\varepsilon}\cdot \hat{m}_t - w_1\theta_{t-1}$$

Remark (Calibration of the learning rate.). The learning rate is a very important parameter to tune to obtain precise results. Indeed, we have not included results to illustrate our choice of learning rate, but it is important to specify that many tests have been done to determine the right parameter: choosing a high learning rate or decreasing its value too slowly leads to big oscillations and bad convergence. Choosing a too-low value or decrease too fast leads to a very slow and bad convergence since the loss decreases very slowly and does not succeed in decreasing enough to have good results. Hence, our learning rates have been fine-tuned using several training on the two test cases using different learning rate schedulers. The scheduler giving the best results was the chosen one, using the validation loss to make the learning rate decrease.

Algorithm 2 Training loop.

Initialisation: θ_0 the initial random parameters, $X = (F, \Phi, G)$ and Y_{true} the training part of the dataset, the batch size and the regularization parameter λ .

for $t = 1$ **to** number of epochs **do**

  for $i = 1$ **to** number of batches **do**

    Select a batch $(F^i, \Phi^i, G^i) \subset X$ and $Y^i_{\text{true}} \subset Y_{\text{true}}$ of size batch size.

    Call the model : $Y_\theta = \mathcal{G}_{\theta_{ti-1}}(F^i, \Phi^i, G^i)$.

    Compute the loss :

$$\mathcal{L}(Y^i_{\text{true}}, Y_\theta) + \underbrace{\frac{\lambda}{2 \times \text{batch size}} \sum_j |w_j|^2}_{L^2 \text{ regularization}} .$$

    Compute the gradient of the loss, w.r.t the parameters θ_{ti-1}: $\nabla_{\theta_{ti-1}} \mathcal{L}$.

    Optimizer step : step of Algorithm 1.

  end for } Training step

  Let $(F_{\text{val}}, \Phi_{\text{val}}, G_{\text{val}})$ and Y_{val} be the validation part of the dataset.

  Call the model on the validation sample : $Y_\theta = \mathcal{G}_{\theta_{ti}}(F_{\text{val}}, \Phi_{\text{val}}, G_{\text{val}})$.

  Compute the loss : $\mathcal{L}(Y_{\text{val}}, Y_\theta)$. } Validation step

  Learning rate scheduler step.

end for