Check for
updates

Mesh-Informed Neural Networks for Operator Learning in Finite Element Spaces

Nicola Rares Franco[1] · **Andrea Manzoni[1]** · **Paolo Zunino[1]**

© The Author(s) 2023

Abstract

Thanks to their universal approximation properties and new efficient training strategies, Deep Neural Networks are becoming a valuable tool for the approximation of mathematical operators. In the present work, we introduce Mesh-Informed Neural Networks (MINNs), a class of architectures specifically tailored to handle mesh based functional data, and thus of particular interest for reduced order modeling of parametrized Partial Differential Equations (PDEs). The driving idea behind MINNs is to embed hidden layers into discrete functional spaces of increasing complexity, obtained through a sequence of meshes defined over the underlying spatial domain. The approach leads to a natural pruning strategy which enables the design of sparse architectures that are able to learn general nonlinear operators. We assess this strategy through an extensive set of numerical experiments, ranging from nonlocal operators to nonlinear diffusion PDEs, where MINNs are compared against more traditional architectures, such as classical fully connected Deep Neural Networks, but also more recent ones, such as DeepONets and Fourier Neural Operators. Our results show that MINNs can handle functional data defined on general domains of any shape, while ensuring reduced training times, lower computational costs, and better generalization capabilities, thus making MINNs very well-suited for demanding applications such as Reduced Order Modeling and Uncertainty Quantification for PDEs.

✉ Nicola Rares Franco
  nicolarares.franco@polimi.it

  Andrea Manzoni
  andrea1.manzoni@polimi.it

  Paolo Zunino
  paolo.zunino@polimi.it

[1] MOX, Department of Mathematics, Politecnico di Milano, P.zza Leonardo da Vinci, 32, 20133 Milan, Italy

🍨 Springer

1 Introduction

Deep Neural Networks (DNNs) are one of the fundamental building blocks in modern Machine Learning. Originally developed to tackle classification tasks, they have become extremely popular after reporting striking achievements in fields such as computer vision [34] and language processing [50]. Not only, an in-depth investigation of their approximation properties has also been carried out in the last decade [6, 11, 17, 25, 29]. In particular, DNNs have been recently employed for learning (nonlinear) operators in high-dimensional spaces [12, 22, 33, 40], because of their unique properties, such as the ability to blend theoretical and data-driven approaches. Additionally, the interest in using DNNs to learn high-dimensional operators arises from the potential repercussions that these models would have on fields such as Reduced Order Modeling.

Consider for instance a parameter dependent PDE problem, where each parameter instance μ leads to a solution u_μ. In this framework, multi-query applications such as optimal control and statistical inference are prohibitive to implement, as they imply repeated queries to expensive numerical solvers. Then, learning the operator $\mu \to u_\mu$ becomes of key interest, as it allows one to replace numerical solvers with much cheaper surrogates. To this end, DNNs can be a valid and powerful alternative, as they were recently shown capable of either comparable or superior results with respect to other state-of-the-art techniques, e.g. [4, 19, 20]. More generally, other works have recently exploited physics-informed machine learning for efficient reduced order modeling of parametrized PDEs [10, 49]. Also, DNN models have the practical advantage of being highly versatile as, differently from other techniques such as splines and wavelets, they can easily adapt to both high-dimensional inputs, as in image recognition, and outputs, as in the so-called generative models.

However, when the dimensions into play become very high, there are some practical issues that hinder the use of as-is DNN models. In fact, classical dense architectures tend to have too many degrees of freedom, which makes them harder to train, computationally demanding and prone to overfitting [2]. As a remedy, alternative architectures such as Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNNs) have been employed over the years. These architectures can handle very efficiently data defined respectively over hypercubes (CNNs) or graphs (GNNs). Nevertheless, these models do not provide a complete answer, especially when the high-dimensionality arises from the discretization of a functional space such as $L^2(\Omega)$, where $\Omega \subset \mathbb{R}^d$ is some bounded domain, possibly nonconvex. In fact, CNNs cannot handle general geometries and they might become inappropriate as soon as Ω is not an hypercube, although some preliminary attempts to generalize CNN in this direction have recently appeared [21]. Conversely, GNNs have the benefit of considering their inputs and outputs as defined over the vertices of a graph [47]. This appears to be a promising feature, since a classical way to discretize spatial domains is to use meshing strategies, and meshes are ultimately graphs. However, GNNs are heavily based on the graph representation itself, and their construction does not exploit the existence of an underlying spatial domain. In particular, GNNs were not constructed to operate at different levels of resolution: still, in a context in which the discretization is ultimately fictitious, this would be a desirable property.

Inspired by these considerations, we introduce a novel class of sparse architectures, which we refer to as Mesh-Informed Neural Networks (MINNs), to tackle the problem of learning a (nonlinear) operator

$$\mathcal{G} : V_1 \to V_2,$$

where V_1 and V_2 are some functional spaces, e.g. $V_1 = V_2 \subseteq L^2(\Omega)$. The definition of $\mathcal{G}$ may involve both local and nonlocal operations, such as derivatives and integrals, and it may as well imply the solution to a Partial Differential Equation (PDE).

We cast the above problem in a setting that is more familiar to the Deep Learning literature by introducing a form of high-fidelity discretization. This is an approach that has become widely adopted by now, and it involves the discretization of the functional spaces along the same lines of Finite Element methods, as in [4, 19, 36]. In short, one introduces a mesh having vertices $\{\mathbf{x}_i\}_{i=1}^{N_h} \subset \overline{\Omega}$, and defines $V_h \subset L^2(\Omega)$ as the subspace of piecewise linear Lagrange polynomials, where $h > 0$ is the stepsize of the mesh (the idea can be easily generalized to higher order Finite Elements, as we show later on). Since each $v \in V_h$ is uniquely identified by its nodal values, we have $V_h \cong \mathbb{R}^{N_h}$, and the original operator to be learned can be replaced by

$$\mathcal{G}_h : V_h \cong \mathbb{R}^{N_h} \to V_h \cong \mathbb{R}^{N_h}.$$

The idea is now to approximate $\mathcal{G}_h$ by training some DNN $\Phi : \mathbb{R}^{N_h} \to \mathbb{R}^{N_h}$. As we argued previously, dense architectures are unsuited for such a purpose because of their prohibitive computational cost during training, which is mostly caused by: i) the computational resources required for the optimization, ii) the amount of training data needed to avoid overfitting.

To overcome this bottleneck, we propose Mesh-Informed Neural Networks. These are ultimately based on an *a priori* pruning strategy, whose purpose is to inform the model with the geometrical knowledge coming from Ω. As we will demonstrate later in the paper, despite their simple implementation, MINNs show reduced training times and better generalization capabilities, making them a competitive alternative to other operator learning approaches, such as DeepONets [40] and Fourier Neural Operators [39]. Also, they allow for a novel interpretation of the so-called *hidden layers*, in a way that may simplify the practical problem of designing DNN architectures.

The rest of the paper is devoted to the presentation of MINNs and it is organized as follows. In Sect. 2, we set some notation and formally introduce Mesh-Informed Neural Networks from a theoretical point of view. There, we also discuss their implementation and comment on the parallelism between MINNs and other emerging approaches such as DeepONets [40] and Neural Operators [33]. We then devote Sects. 3, 4 and 5 to the numerical experiments, addressing a different scientific question in each Section. More precisely: in Sect. 3, we provide empirical evidence that the pruning strategy underlying MINNs is powerful enough to resolve the issues of dense architectures; in Sect. 4, we showcase the flexibility of MINNs in handling complex nonconvex domains; finally, in Sect. 5, we compare the performances of MINNs with those of other state-of-the-art Deep Learning algorithms, namely DeepONets and Fourier Neural Operators. Following the numerical experiments, in Sect. 6, we take the chance to present an application where MINNs are employed to answer a practical problem of Uncertainty Quantification related to the delivery of oxygen in biological tissues. Finally, we draw our conclusions and discuss future developments in Sect. 7.

2 Mesh-Informed Neural Networks

In the present Section we present Mesh-Informed Neural Networks, a novel class of architectures specifically built to handle discretized functional data defined over meshes, and thus of particular interest for PDE applications. Preliminary to that, we introduce some notation and recall some of the basic concepts behind classical DNNs.

2.1 Notation and Preliminaries

Deep Neural Networks are a powerful class of approximators that is ultimately based on the composition of affine and nonlinear transformations. Here, we focus on DNNs having a *feedforward* architecture. We report below some basic definitions.

Definition 1 Let $m, n \geq 1$ and $\rho : \mathbb{R} \to \mathbb{R}$. A layer with activation ρ is a map $L : \mathbb{R}^m \to \mathbb{R}^n$ of the form $L(\mathbf{v}) = \rho(\mathbf{W}\mathbf{v} + \mathbf{b})$, for some $\mathbf{W} \in \mathbb{R}^{n \times m}$ and $\mathbf{b} \in \mathbb{R}^n$.

In the literature, $\mathbf{W}$ and $\mathbf{b}$ are usually referred to as the weight and the bias of the layer, respectively. Note that Definition 1 contains an abuse of notation, as ρ is evaluated over an n-dimensional vector: we understand the latter operation componentwise, that is $\rho([x_1, \ldots, x_n]) := [\rho(x_1), \ldots, \rho(x_n)]$.

Definition 2 Let $m, n \geq 1$. A neural network of depth $l \geq 0$ is a map $\Phi : \mathbb{R}^m \to \mathbb{R}^n$ obtained via composition of $l + 1$ layers, $\Phi = L_{l+1} \circ \ldots L_1$.

The layers of a neural network do not need to share the same activation function and usually the output layer, L_{l+1}, does not have one. Architectures with $l = 1$ are known as shallow networks, while the adjective deep is used when $l \geq 2$. We also allow for the degenerate case in which the network reduces to a single layer ($l = 0$). The classical pipeline for building a neural network model starts by fixing the architecture, that is the number of layers and their input–output dimensions. Then, the weights and biases of all layers are tuned according to some procedure, which typically involves the optimization of a loss function computed over a given training set: for further details, the reader can refer to [27].

2.2 Mesh-Informed Layers

We consider the following framework. We are given a bounded domain $\Omega \subset \mathbb{R}^d$, not necessarily convex, and two meshes (see Definition 3 right below) having respectively stepsizes $h, h' > 0$ and vertices

$$\{\mathbf{x}_j\}_{j=1}^{N_h}, \ \{\mathbf{x}_i'\}_{i=1}^{N_{h'}} \subset \overline{\Omega}.$$

The two meshes can be completely different and they can be either structured or unstructured. To each mesh we associate the corresponding space of piecewise linear Lagrange polynomials, namely $V_h, V_{h'} \subset L^2(\Omega)$. Our purpose is to introduce a suitable notion of *mesh-informed layer* $L : V_h \to V_{h'}$ that exploits the *a priori* existence of Ω. In analogy to Definition 1, L should have N_h neurons at input and $N_{h'}$ neurons at output, since $V_h \cong \mathbb{R}^{N_h}$ and $V_{h'} \cong \mathbb{R}^{N_{h'}}$. However, thinking of the state spaces as either comprised of functions or vectors is fundamentally different: while we can describe the objects in V_h as regular, smooth or noisy, these notions have no meaning in $\mathbb{R}^{N_h}$, and similarly for $V_{h'}$ and $\mathbb{R}^{N_{h'}}$. Furthermore, in the case of PDE applications, we are typically not interested in all the elements of V_h and $V_{h'}$, rather we focus on those that present spatial correlations coherent with the underlying physics. Starting from these considerations, we build a novel layer architecture that can meet our specific needs. In order to provide a rigorous definition, and directly extend the idea to higher order Finite Element spaces, we first introduce some preliminary notation. For the sake of simplicity, we will restrict to simplicial Finite Elements [13].

Definition 3 Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Let $\mathcal{M}$ be a collection of d-simplices in Ω, so that each $K \in \mathcal{M}$ is a closed subset of $\overline{\Omega}$. For each *element* $K \in \mathcal{M}$, define the quantities

$$h_K := \operatorname{diam}(K), \qquad R_K := \sup \{\operatorname{diam}(S) \mid S \text{ is a ball contained in } K\}.$$

We say that $\mathcal{M}$ is an admissible mesh of stepsize $h > 0$ over Ω if the following conditions hold.

1. The elements are exhaustive, that is

$$\text{dist}\left(\overline{\Omega}, \bigcup_{K \in \mathcal{M}} K\right) \leq h$$

where $\text{dist}(A, B) = \sup_{x \in A} \inf_{y \in B} |x - y|$ is the distance between A and B.

2. Any two distinct elements $K, K' \in \mathcal{M}$ have disjoint interiors. Also, their intersection is either empty or results in a common face of dimension $s < d$.

4. The elements are non degenerate and their maximum diameter equals h, that is

$$\min_{K \in \mathcal{M}} R_K > 0 \quad \text{and} \quad \max_{K \in \mathcal{M}} h_K = h.$$

In that case, the quantity

$$\sigma = \min_{K \in \mathcal{M}} \frac{h_K}{R_K} < +\infty,$$

is said to be the aspect-ratio of the mesh.

Definition 4 Let $\Omega \subset \mathbb{R}^d$ be a bounded domain and let $\mathcal{M}$ be a mesh of stepsize $h > 0$ defined over Ω. For any positive integer q, we write $X_h^q(\mathcal{M})$ for the Finite Element space of piecewise polynomials of degree at most q, that is

$$X_h^q(\mathcal{M}) := \{v \in \mathcal{C}(\overline{\Omega}) \text{ s.t. } v_{|K} \text{ is a polynomial of degree at most } q \ \forall K \in \mathcal{M}\}.$$

Let $N_h = \dim(X_h^q(\mathcal{M}))$. We say that a collection of nodes $\{\mathbf{x}_i\}_{i=1}^{N_h} \subset \Omega$ and a sequence of functions $\{\varphi_i\}_{i=1}^{N_h} \subset X_h^q(\mathcal{M})$ define a Lagrangian basis of $X_h^q(\mathcal{M})$ if

$$\varphi_j(\mathbf{x}_i) = \delta_{i,j} \quad i, j = 1, \ldots, N_h.$$

We write $\Pi_{h,q}(\mathcal{M}) : X_h^q(\mathcal{M}) \to \mathbb{R}^{N_h}$ for the function-to-nodes operator,

$$\Pi_{h,q}(\mathcal{M}) : \quad v \to [v(\mathbf{x}_1), \ldots, v(\mathbf{x}_{N_h})],$$

whose inverse is

$$\Pi_{h,q}^{-1}(\mathcal{M}) : \quad \mathbf{c} \to \sum_{i=1}^{N_h} c_i \varphi_i.$$

We now have all we need to introduce our concept of mesh-informed layer.

Definition 5 (Mesh-informed layer) Let $\Omega \subset \mathbb{R}^d$ be a bounded domain and $d : \Omega \times \Omega \to [0, +\infty)$ a given distance function. Let $\mathcal{M}$ and $\mathcal{M}'$ be two meshes of stepsizes h and h', respectively. Let $V_h = X_h^q(\mathcal{M})$ and $V_{h'} = X_{h'}^{q'}(\mathcal{M})$ be the input and output spaces, respectively. Denote by $\{\mathbf{x}_j\}_{j=1}^{N_h}$ and $\{\mathbf{x}_i'\}_{i=1}^{N_{h'}}$ the nodes associated to a Lagrangian basis of V_h and $V_{h'}$ respectively. A mesh-informed layer with activation function $\rho : \mathbb{R} \to \mathbb{R}$ and support $r > 0$ is a map $L : V_h \to V_{h'}$ of the form

$$L = \Pi_{h',q'}^{-1}(\mathcal{M}') \circ \tilde{L} \circ \Pi_{h,q}(\mathcal{M})$$

where $\tilde{L} : \mathbb{R}^{N_h} \to \mathbb{R}^{N_h'}$ is a layer with activation ρ whose weight matrix $\mathbf{W}$ satisfies the additional sparsity constraint below,

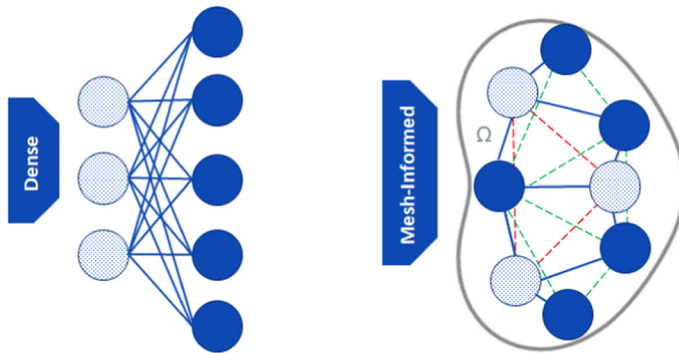$$d(\mathbf{x}_j, \mathbf{x}_i') > r \implies \mathbf{W}_{i,j} = 0.$$

Fig. 1 Comparison of a dense layer (cf. Definition 1) and a mesh-informed layer (cf. Definition 5). The dense model features 3 neurons at input and 5 at output. All the neurons communicate: consequently the weight matrix of the layer has 15 active entries. In the mesh-informed counterpart, neurons become vertices of two meshes (resp. in green and red) defined over the same spatial domain Ω. Only nearby neurons are allowed to communicate. This results in a sparse model with only 9 active weights

The distance function d in Definition 5 can be any metric over Ω. For instance, one may choose to consider the Euclidean distance, $d(\mathbf{x}, \mathbf{x}') := |\mathbf{x} - \mathbf{x}'|$. However, if the geometry of Ω becomes particularly involved, better choices of d might be available, such as the *geodesic distance*. The latter quantifies the distance of two points $\mathbf{x}, \mathbf{x}' \in \Omega$ by measuring the length of the shortest path within Ω between $\mathbf{x}$ and $\mathbf{x}'$, namely

$$d(\mathbf{x}, \mathbf{x}') := \inf \left\{ \int_0^1 |\gamma'(t)| dt, \text{ with } \gamma \in \mathcal{C}^1([0, 1], \mathbb{R}^d), \ \gamma([0, 1]) \subseteq \Omega, \right.$$

$$\left. \gamma(0) = \mathbf{x}, \ \gamma(1) = \mathbf{x}' \right\}.$$

It is worth pointing out that, as a matter of fact, the projections $\Pi_{h,q}(\mathcal{M})$ and $\Pi_{h',q'}^{-1}(\mathcal{M}')$ have the sole purpose of making Definition 5 rigorous. What actually defines the mesh-informed layer L are the sparsity patterns imposed to $\tilde{L}$. In fact, the idea is that these constraints should help the layer in producing outputs that are more coherent with the underlying spatial domain (cf. Fig. 1). In light of this intrinsic duality between L and $\tilde{L}$, we will refer to the weights and biases of L as to those that are formally defined in $\tilde{L}$. Also, for better readability, from now on we will use the notation

$$L : V_h \xrightarrow{r} V_{h'},$$

to emphasize that L is a mesh-informed layer with support r. We note that dense layers can be recovered by letting $r \geq \sup \{d(\mathbf{x}, \mathbf{x}') \mid \mathbf{x}, \mathbf{x}' \in \Omega\}$, while lighter architecture are obtained for smaller values of r. The following result provides an explicit upper bound on the number of nonzero entries in a mesh-informed layer. For the sake of simplicity, we restrict to the case in which d is the Euclidean distance.

Theorem 1 *Let $\Omega \subset \mathbb{R}^d$ be a bounded domain and d the Euclidean distance. Let $\mathcal{M}$ and $\mathcal{M}'$ be two meshes having respectively stepsizes h, h' and aspect-ratios σ, σ'. Let*

$$h_{\min} := \min_{K \in \mathcal{M}} h_K, \quad h'_{\min} := \min_{K' \in \mathcal{M}'} h_{K'}$$

be the smallest diameters within the two meshes respectively. Let $L : V_h \xrightarrow{r} V_{h'}$ be a mesh-informed layer of support $r > 0$, where $V_h := X_h^q(\mathcal{M})$ and $V_{h'} := X_{h'}^{q'}(\mathcal{M}')$. Then,

$$\|\mathbf{W}\|_0 \leq C \left(\sigma\sigma' \frac{r}{h_{\min}h'_{\min}} \right)^d$$

where $\|\mathbf{W}\|_0$ is the number of nonzero entries in the weight matrix of the layer L, while $C = C(\Omega, d, q, q') > 0$ is a constant depending only on Ω, d, q and q'.

Proof Let $N_h := \dim(V_h)$, $N_{h'} := \dim(V_{h'})$ and let $\{\mathbf{x}_j\}_{j=1}^{N_h}$, $\{\mathbf{x}'_i\}_{i=1}^{N_{h'}}$ be the Lagrangian nodes in the two meshes, respectively. Let $\omega := |B(\mathbf{0}, 1)|$ be the volume of the unit ball in $\mathbb{R}^d$. Since $\min_{K'} R_{K'} \geq h'_{\min}/\sigma'$, the volume of an element in the output mesh is at least

$$v_{\min}(h') := (h'_{\min}/\sigma')^d \omega.$$

It follows that, for any $\mathbf{x} \in \Omega$, the ball $B(\mathbf{x}, r)$ can contain at most

$$n_e(r, h') := \frac{\omega r^d}{v_{\min}(h')} = \left(\frac{\sigma' r}{h'_{\min}} \right)^d$$

elements of the output mesh. Therefore, the number of indices i such that $|\mathbf{x}'_i - \mathbf{x}_j| \leq r$ is at most $n_e(r, h')c(d, q')$, where $c(d, q') := (d + q')!/(q'!d!)$ bounds the number of degrees of freedom within each element. Finally,

$$\|\mathbf{W}\|_0 \leq N_h n_e(r, h') c(d, q') \leq$$
$$\leq \frac{c(d, q)|\Omega|}{v_{\min}(h)} n_e(r, h') c(d, q') = \frac{c(d, q)c(d, q')|\Omega|}{\omega} \cdot \left(\frac{\sigma\sigma' r}{h_{\min}h'_{\min}} \right)^d.$$

$\square$

Starting from here, we define MINNs by composition, with a possible interchange of dense and mesh-informed layers. Consider for instance the case in which we want to define a Mesh-Informed Neural Network $\Phi : \mathbb{R}^p \to V_h \cong \mathbb{R}^{N_h}$ that maps a low-dimensional input, say $p \ll N_h$, to some functional output. Then, using our notation, one possible architecture could be

$$\Phi : \mathbb{R}^p \longrightarrow \underset{\substack{\cong \\ \mathbb{R}^{N_{4h}}}}{V_{4h}} \xrightarrow{r = 0.5} \underset{\substack{\cong \\ \mathbb{R}^{N_{2h}}}}{V_{2h}} \xrightarrow{r = 0.25} \underset{\substack{\cong \\ \mathbb{R}^{N_h}}}{V_h} , \qquad (1)$$

The above scheme defines a MINN of depth $l = 2$, as it is composed of 3 layers. The first layer is dense (Definition 1) and has the purpose of preprocessing the input while mapping the data onto a coarse mesh (stepsize $4h$). Then, the remaining two layers perform local transformations in order to return the desired output. Note that the three meshes need not to satisfy any hierarchy, see e.g. Fig. 2. Also, the corresponding Finite Element spaces need not to share the same polynomial degree. Clearly, (1) can be generalized by employing any number of layers, as well as any sequence of stepsizes $h_1, \ldots, h_n$ and supports $r_1, \ldots, r_{n-1}$. Similarly, as we shall demonstrate in our experiments, one may also modify (1) to handle functional inputs, e.g. by introducing a mesh-informed layer at the beginning of the architecture. The choice of the hyperparameters remains problem specific, but a good rule of thumb is to decrease the supports as the mesh becomes finer, so that the network complexity is kept under control (cf. Theorem 1).
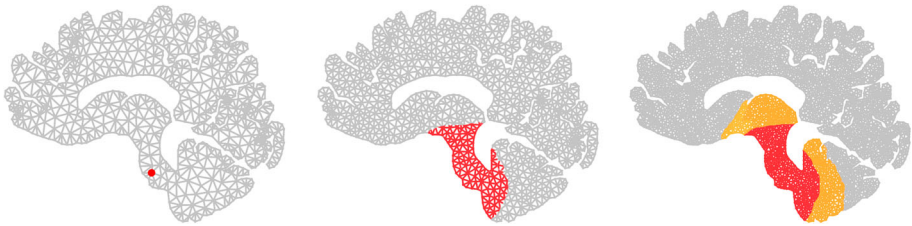
Fig. 2 MINNs operate at different resolution levels to enforce local properties. Here, three meshes of a vertical brain slice, represent three different hidden states in the pipeline of a suitable MINN architecture, where neurons are identified with mesh vertices. Due to the sparsity constraint (Definition 5), the red neuron only fires information to those in the highlighted region (second mesh); in turn, these can only communicate with the neurons in the third mesh that are sufficiently close (orange region). In this way, despite relying on local operations, MINNs can eventually spread information all over the domain by exploiting the composition of multiple layers

2.3 Implementation Details

For simplicity, let us first focus on the case in which distances are evaluated according to the Euclidean metric, $d(\mathbf{x}, \mathbf{x}') = |\mathbf{x} - \mathbf{x}'|$. In this case, the practical implementation of mesh-informed layers is straightforward and can be done as follows, cf. Fig. 3. Given $\Omega \subset \mathbb{R}^d$, $h, h' > 0$, let $\mathbf{X} \in \mathbb{R}^{N_h \times d}$ and $\mathbf{X}' \in \mathbb{R}^{N_{h'} \times d}$ be the matrices containing the degrees of freedom associated to the chosen Finite Element spaces, that is $\mathbf{X}_{j,.} := [X_{j,1}, \ldots, X_{j,d}]$ are the coordinates of the jth node in the input mesh, and similarly for $\mathbf{X}'$. In order to build a mesh-informed layer of support $r > 0$, we first compute all the pairwise distances $D_{i,j} := |\mathbf{X}_{j,.} - \mathbf{X}'_{i,.}|^2$ among the nodes in the two meshes. This can be done efficiently using tensor algebra, e.g.

$$D = \sum_{l=1}^{d} (e_{N_{h'}} \otimes \mathbf{X}_{.,l} - \mathbf{X}'_{.,l} \otimes e_{N_h})^{\circ 2}$$

where $\mathbf{X}_{.,l}$ is the lth column of $\mathbf{X}$, $e_k := [1, \ldots, 1] \in \mathbb{R}^k$, $\otimes$ is the tensor product and $\circ 2$ is the Hadamard power. We then extract the indices $\{(i_k, j_k)\}_{k=1}^{\mathrm{dof}}$ for which $D_{i_k, j_k} \leq r^2$ and initialize a weight vector $\mathbf{w} \in \mathbb{R}^{\mathrm{dof}}$. This allows us to declare the weight matrix $\mathbf{W}$ in sparse format by letting the nonzero entries be equal to w_k at position (i_k, j_k), so that $\|\mathbf{W}\|_0 = \mathrm{dof}$. Preliminary to the training phase, we fill the entries of $\mathbf{w}$ with random values sampled from a suitable Gaussian distribution. The optimal choice for such distribution may be problem dependent. Empirically, we see that good results can be achieved by sampling the weights $w_1, \ldots, w_{\mathrm{dof}}$ independently from a centered normal distribution with variance $1/\mathrm{dof}$. Otherwise, if the architecture is particularly deep, another possibility is to consider an adaptation of the He initialization [31]. In the case of α-leakyReLU nonlinearities, the latter suggests sampling w_i from a centered Gaussian distribution with variance

$$\sigma_i^2 = \frac{2}{(1 + \alpha^2)s_i}, \tag{2}$$

where s_i is the cardinality of the set $\{j \mid D_{i,j} \leq r^2\}$, that is, the number of input neurons that communicate with the ith output neuron. Finally, in analogy to [31], if the layer has no activation then one may just set $\alpha = 1$ in Eq. (2).

The above reasoning can be easily adapted to the general case, provided that one is able to compute efficiently all the pairwise distances $d(\mathbf{X}_{j,.}, \mathbf{X}'_{i,.})$. Of course, the actual imple-
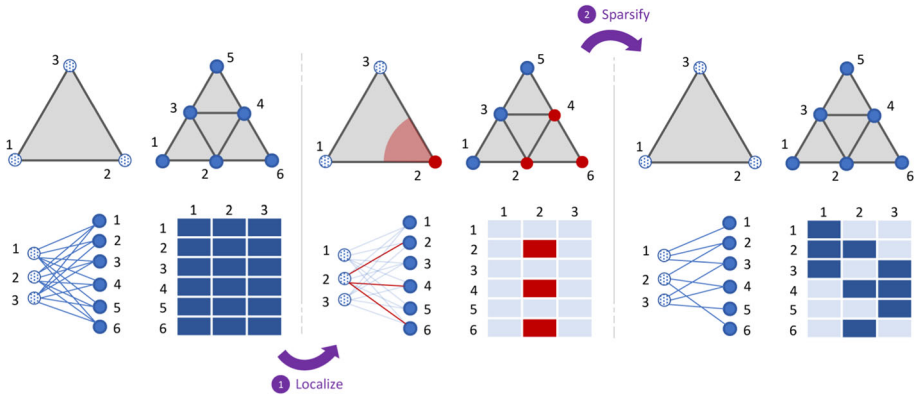
Fig. 3 Implementation of a mesh-informed layer. Each panel reports a representation of the input/output meshes (top row), the network architecture (bottom left) and the corresponding weight matrix (bottom right). The degrees of freedom at the input/output meshes are associated to the input/output neurons of a reference dense layer (left panel). For each input node, the neighbouring nodes at output are highlighted (center panel). Then, all the remaining connections are pruned, and the final model is left with a sparse weight matrix (right panel)

mentation will then depend on the specific choice of d. Since the case of geodesic distances can be of particular interest in certain applications, we shall briefly discuss it below. In this setting, the main difficulty arises from the fact that, in general, we are required to compute distances between points of different meshes. Additionally, if we consider Finite Element spaces of degree $q > 1$, not all the Lagrangian nodes will be placed over the mesh vertices, meaning that we cannot exploit the graph structure of the mesh to calculate shortest paths.

To overcome these drawbacks, we propose the introduction of an auxiliary coarse mesh $\mathcal{M}_0 := \{K_i\}_{i=1}^m$, whose sole purpose is to capture the geometry of the domain. We use this mesh to build another graph, $\mathscr{G}$, which describes the location of the elements K_i. More precisely, let $\mathbf{c}_i$ be the centroid of the element K_i. We let $\mathscr{G}$ be the weighted graph having vertices $\{\mathbf{c}_i\}_{i=1}^m$, where we link $\mathbf{c}_i$ with $\mathbf{c}_j$ if and only if the elements K_i and K_j are adjacent. Then, to weight the edges, we use the Euclidean distance between the centroids. Once $\mathscr{G}$ is constructed, we use Dijkstra's algorithm to compute all the shortest paths along the graph. This leaves us with an estimated geodesic distance $g_{i,j}$ for each pair of centroids $(\mathbf{c}_i, \mathbf{c}_j)$, which we can precompute and store in a suitable matrix. Then, we approximate the geodesic distance of *any* two points $\mathbf{x}, \mathbf{x}' \in \Omega$ as

$$d(\mathbf{x}, \mathbf{x}') \approx g_{i(\mathbf{x}), i(\mathbf{x}')},$$

where $i : \Omega \to \{1, \ldots, m\}$ maps each point to an element containing it, see Fig. 4. In other words,

$$i(\mathbf{x}) := \min\{i \mid \mathbf{x} \in K_i\}.$$

If the auxiliary mesh does not fill Ω completely, the relaxed version below may be used as well

$$i(\mathbf{x}) := \min \left\{ i \mid \mathrm{dist}(\mathbf{x}, K_i) = \min_{j=1,\ldots,m} \mathrm{dist}(\mathbf{x}, K_j) \right\}.$$

In general, evaluating the index function i can be done in $O(m)$ time. In particular, going back to our original problem, we can approximate all the pairwise distances between the
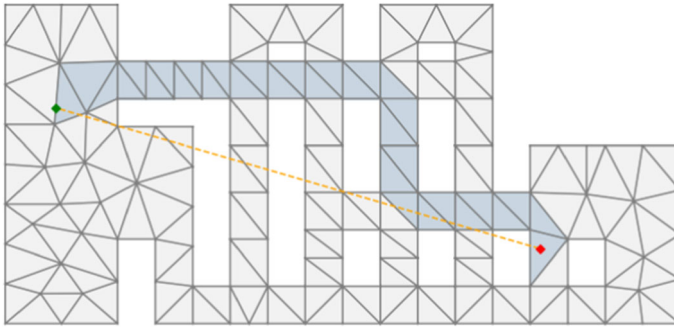
Fig. 4 Approximation of the geodesic distance $d(\mathbf{x}, \mathbf{y})$ as the shortest path across elements (blue triangles). In general, the latter will differ from the Euclidean distance between the two points (orange dashed line) (Color figure online)

nodes of the input and the output spaces in $O(mN_h + mN'_n + m^2)$ time. In fact, we can run Dijkstra's algorithm once and for all with a computational cost of $O(m^2)$. Then, we just need to evaluate the index function i for all the Lagrangian nodes $\{\mathbf{x}\}_{i=1}^{N_h}$ and $\{\mathbf{x}'\}_{i=1}^{N_{h'}}$, which takes respectively $O(mN_h)$ and $O(mN_{h'})$.

2.4 A Supervised Learning Approach Based on MINNs

Once a mesh-informed architecture has been constructed, we learn the operator of interest through a standard supervised approach. Let $\mathcal{G}_h : \Theta \to V_h$ be the high-fidelity discretization of a suitable operator, whose pointwise evaluations may as well involve the numerical solution to a PDE. Here, we allow the input space, Θ, to either consist of vectors, $\Theta \subset \mathbb{R}^p$, or (discretized) functions, e.g. $\Theta = V_h$. Let $\Phi : \Theta \to V_h$ be a given MINN architecture: in general, depending on the input type, the latter will consist of both dense and mesh-informed layers. We assume that Φ has already been initialized according to some procedure, such as those reported in Sect. 2.3, and that it is ready for training.

We make direct use of the forward operator to compute a suitable collection of training pairs (thereby also referred to as *training samples* or *snapshots*, following the classical terminology adopted in the Reduced Order Modeling literature),

$$\{f_i, \ u_h^i\}_{i=1}^{N_{\text{train}}} \subset \Theta \times V_h,$$

with $u_h^i := \mathcal{G}_h(f_i)$. Here, with little abuse of notation, we write $f \in \Theta$, that is, we assume to be in the case of functional inputs. If not so, the reader may simply replace the f_i's with suitable $\boldsymbol{\mu}_i$'s. The training pairs are typically chosen at random, i.e. by equipping the input space with a probability distribution $\mathbb{P}$ and by computing N_{train} independent samples.

We then train Φ by minimizing the mean squared L^2-error, that is, by optimizing the following loss function

$$\hat{\mathscr{L}}(\Phi) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \|u_h^i - \Phi(f_i)\|_{L^2(\Omega)}^2, \tag{3}$$

which acts as the empirical counterpart of

$$\mathscr{L}(\Phi) = \mathbb{E}_{f \sim \mathbb{P}} \|\mathcal{G}_h(f) - \Phi(f)\|_{L^2(\Omega)}^2. \tag{4}$$

This allows one to actually tune the model parameters (i.e. all layers weights and biases), and obtain a suitable approximation $\Phi \approx \mathcal{G}_h$. Then, multiple metrics can be used to evaluate the quality of such an approximation. In this work, we shall often consider the average relative L^2-error, which we estimate thanks to a precomputed *test set*

$$\mathscr{E}(\Phi) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{\|u_h^{i,\text{test}} - \Phi(f_i^{\text{test}})\|_{L^2(\Omega)}}{\|u_h^{i,\text{test}}\|_{L^2(\Omega)}} \approx \mathbb{E}_{f \sim \mathbb{P}} \left[\frac{\|\mathcal{G}_h f - \Phi(f)\|_{L^2(\Omega)}}{\|\mathcal{G}_h f\|_{L^2(\Omega)}} \right]. \quad (5)$$

The test set $\{f_i^{\text{test}}, u_{h,\text{test}}^i\}_{i=1}^{N_{\text{test}}}$ is constructed independently of the training set, but still relying on the high-fidelity operator $\mathcal{G}_h$ and the probability distribution $\mathbb{P}$.

If the approximation is considered satisfactory, then the expensive operator $\mathcal{G}_h$ can be replaced with the cheaper MINN surrogate Φ, whose outputs can be evaluated with little to none computational cost.

Before concluding this paragraph, it may be worth to emphasize a couple of things. First of all, we highlight the fact that both the loss and the error function, respectively $\widetilde{\mathscr{L}}$ and $\mathscr{E}$, require the computation of integral norms. However, this is not an issue: since both u_h^i and $\Phi(f_i)$ lie in V_h, we can compute these norms by relying on the mass matrix $\mathbf{M} \in \mathbb{R}^{N_h \times N_h}$, which can be precomputed and stored once and for all. We recall in fact that the latter is a highly sparse matrix defined in such a way that

$$\forall v \in V_h, \quad \mathbf{v} := [v(\mathbf{x}_1), \dots, v(\mathbf{x}_{N_h})]^T \implies \|v\|_{L^2(\Omega)} = \mathbf{v}^T \mathbf{M} \mathbf{v},$$

where $\{\mathbf{x}_i\}_{i=1}^{N_h}$ are the nodes corresponding to the degrees of freedom in V_h, cf. *function-to-nodes operator* in Definition 4.

Finally, we remark that the MINN architecture is trained in a purely supervised fashion. Even if the definition of the operator $\mathcal{G}_h$ might involve a PDE or any other physical law, none of this knowledge is imposed over Φ. Similarly, we do not impose boundary conditions, mass conservation, or other constraints, on the outputs of the MINN model: in principle, these should be learned implicitly. Nonetheless, we recognize that MINNs should benefit from the integration of such additional knowledge, as this is what other researchers have already observed for other approaches in the literature [10, 21, 49]. As of now, we leave these considerations for future works.

2.5 Relationship to Other Deep Learning Techniques

It is worth to comment on the differences and similarities that MINNs share with other Deep Learning approaches. We discuss them below.

2.5.1 Relationship to CNNs and GNNs

Mesh-Informed architectures can operate at different levels of resolution, in a way that is very similar to CNNs. However, their construction comes with multiple advantages. First of all, Definition 5 adapts to any geometry, while convolutional layers typically operate on square or cubic input–output. Furthermore, convolutional layers use weight sharing, meaning that all parts of the domain are treated in the same way. This may not be an optimal choice in some applications, such as those involving PDEs, as we may want to differentiate our behavior over Ω (for instance near of far away from the boundaries).

Conversely, MINNs share with GNNs the ability to handle general geometries. As a matter of fact, we mention that these architectures have been recently applied to mesh-

based data, see e.g. [15, 28, 45, 51]. With respect to GNNs, the main advantage of MINNs lies in their capacity to work at different resolutions. This fact, which essentially comes from the presence of an underlying spatial domain, has at least two advantages: it provides a direct way for either reducing or increasing the dimensionality of a given input, and it increases the interpretability of hidden layers (now the number of neurons is not arbitrary but comes from the chosen discretization). In a way, mesh-informed architectures are similar to hierarchical GNNs, however, their construction is fundamentally different. While GNNs use *aggregation* strategies to collapse neighbouring information, MINNs differentiate their behavior depending on the nodes that are being involved. This difference, makes the two approaches better suited for different applications. For instance, GNNs can be of great interest when dealing with geometric variability, as their construction allows for a single architecture to operate over completely different domains, see e.g. [23]. Conversely, if the shape of the domain is fixed, then MINNs may grant a higher expressivity, as they are ultimately obtained by pruning dense feedforward models. In this sense, MINNs only exploit meshing strategies as auxiliary tools, and they appear to be a natural choice for learning discretized functional outputs.

2.5.2 Relationship to DeepONets and Neural Operators

Recently, some new DNN models have been proposed for operator learning. One of these are DeepONets [40], a mesh-free approach that is based on an explicit decoupling between the input and the space variable. More precisely, DeepONets consider a representation of the following form

$$(\mathcal{G}_h f)(\mathbf{x}) \approx \Psi(f) \cdot \phi(\mathbf{x}),$$

where $\cdot$ is the dot product, $\Psi : V_h \to \mathbb{R}^m$ is the branch-net, and $\phi : \Omega \to \mathbb{R}^m$ is the trunk-net. DeepONets have been shown capable of learning nonlinear operators and are now being extended to include *a priori* physical knowledge, see e.g. [49]. We consider MINNs and DeepONets as two fundamentally different approaches that answer different questions. DeepONets were originally designed to process input data coming from sensors and, being mesh-free, they are particularly suited for those applications where the output is only partially known or observed. In contrast, MINNs are rooted on the presence of a high-fidelity model $\mathcal{G}_h$ and are thus better suited for tasks such as reduced order modeling. Another difference lies in the fact that DeepONets include explicitly the dependence on the space variable $\mathbf{x}$: because of this, suitable strategies are required in order for them to deal with complex geometries or incorporate additional information, such as boundary data, see e.g. [24]. Conversely, MINNs can easily handle this kind of issues thanks to their global perspective.

In this sense, MINNs are much closer to the so-called Neural Operators, a novel class of DNN models first proposed by Kovachki et al. in [33]. Neural Operators are an extension of classical DNNs that was developed to operate between infinite dimensional spaces. These models are ultimately based on Hilbert-Schmidt operators, meaning that their linear part, that is ignoring activations and biases, is of the form

$$W : f \to \int_\Omega k(\cdot, \mathbf{x}) f(\mathbf{x}) d\mathbf{x} \tag{6}$$

where $k : \Omega \times \Omega \to \mathbb{R}$ is some *kernel* function that has to be identified during the training phase. Clearly, the actual implementation of Neural Operators is carried out in a discrete setting and integrals are replaced with suitable quadrature formulas. We mention that, among

these architectures, Fourier Neural Operators (FNO) are possibly the most popular ones: we shall discuss more about the comparison between MINNs and FNOs in Sect. 5.

The construction of Neural Operators is very general, to the point that other approaches, such as ours, can be seen as a special case. Indeed, a rough Monte Carlo-type estimate of (6) would yield

$$(Lf)(\mathbf{x}'_i) = \int_\Omega k(\mathbf{x}'_i, \mathbf{x}) f(\mathbf{x}) d\mathbf{x} \approx \frac{|\Omega|}{N_h} \sum_j k(\mathbf{x}'_i, \mathbf{x}_j) f(\mathbf{x}_j).$$

If we let $\{\mathbf{x}_j\}_j$ and $\{\mathbf{x}'_i\}_i$ be the nodes in the two meshes, then the constraint in Definition 5 becomes equivalent to the requirement that k is supported somewhere near the diagonal, that is $\mathrm{supp}(k) \subseteq \{(\mathbf{x}, \mathbf{x} + \varepsilon) \text{ with } |\varepsilon| \leq r\}$.

We believe that these parallels are extremely valuable, as they indicate the existence of a growing scientific community with common goals and interests. Furthermore, they all contribute to the enrichment of the operator learning literature, providing researchers with multiple alternatives from which to choose: in fact, as we shall see in Sect. 5, depending on the problem at hand one methodology might be better suited than the others.

3 Numerical Experiments: Effectiveness of the Pruning Strategy

We provide empirical evidence that the sparsity introduced by MINNs can be of great help in learning maps that involve functional data, such as nonlinear operators, showing a reduced computational cost and better generalization capabilities. We first detail the setting of each experiment alone, specifying the corresponding operator to be learned and the adopted MINN architecture. Then, at the end of the current Section, we discuss the numerical results.

Throughout all our experiments, we adopt a standardized approach for designing and training the networks. In general, we always employ the 0.1-leakyReLU activation for all the hidden layers, while we do not use any activation at the output. Every time a mesh-informed architecture is introduced, we also consider its dense counterpart, obtained without imposing the sparsity constraints. Both networks are then trained following the same criteria, so that a fair comparison can be made. As loss function, we always consider the mean square error computed with respect to the L^2 norm, cf. Eq. (4).

The optimization of the loss function is performed via the L-BFGS optimizer, with learning rate always equal to 1 and no batching. What may change from case to case are the network architecture, the number of epochs, and the size of the training set. After training, we compare mesh-informed and dense architectures by evaluating their performance on 500 unseen samples (test set), which we use to compute an unbiased estimate of the average relative error, cf. Eq. (5).

All the code was written in Python 3, mainly relying on the `FEniCS` and `mshr` libraries for the construction of Finite Element spaces and the numerical solution of PDEs. The implementation of neural network models, instead, was handled in `Pytorch`, and their training was carried out on a GPU NVidia Tesla V100 (32GB of RAM).
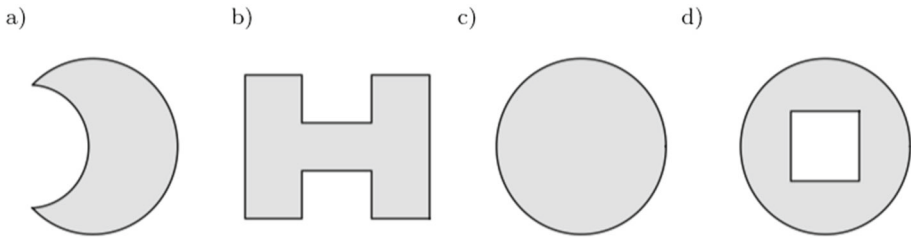
Fig. 5 Domains considered for the numerical experiments in Sect. 3. Panel **a** up to boundaries, Ω is the difference of two circles, $B(\mathbf{0}, 1)$ and $B(\mathbf{x}_0, 0.7)$, where $\mathbf{x}_0 = (-0.75, 0)$. Panel **b** A polygonal domain obtained by removing the rectangles $[-0.75, 0.75] \times [0.5, 1.5]$ and $[-0.75, 0.75] \times [-1.5, -0.5]$ from $(-2, 2) \times (-1.5, 1.5)$. Panel **c** the unit circle $B(\mathbf{0}, 1)$. Panel **d** Ω is obtained by removing a square, namely $[-0.4, 0.4]^2$, from the unit circle $B(\mathbf{0}, 1)$

3.1 Description of the Benchmark Operators

Learning a Parametrized Family of Functions

Let Ω be the domain defined as in Fig. 5a. For our first experiment, we consider a variation of a classical problem concerning the calculation of the *signed distance function* of Ω. This kind of functions are often encountered in areas such as computer vision [43] and real-time rendering [1]. In particular, we consider the following operator,

$$\mathcal{G} : \Theta \subset \mathbb{R}^3 \to L^2(\Omega)$$
$$\mathcal{G} : \boldsymbol{\mu} \to u_{\boldsymbol{\mu}}(\mathbf{x}) := \min_{\substack{\mathbf{y} \in \partial\Omega, \\ y_2 > \mu_1}} |\mathbf{y} - \mathbf{A}_{\mu_3}\mathbf{x}| e^{x_1\mu_2}$$

where $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3)$ is a finite dimensional vector, and $\mathbf{A}_{\mu_3} = \mathrm{diag}(1, \mu_3)$. In practice, the value of $u_{\boldsymbol{\mu}}(\mathbf{x})$ corresponds to the (weighted) distance between the dilated point $\mathbf{A}_{\mu_3}\mathbf{x}$ and the truncated boundary $\partial\Omega \cap \{\mathbf{y} : y_2 > \mu_1\}$.

As input space we consider $\Theta := [0, 1] \times [-1, 1] \times [1, 2]$, endowed with the uniform probability distribution. Since the input is finite-dimensional, we can think of $\mathcal{G}$ as to the parametrization of a 3-dimensional hypersurface in $L^2(\Omega)$. We discretize Ω using P1 triangular Finite Elements with mesh stepsize $h = 0.02$, resulting in the high-fidelity space $V_h := \cong \mathbb{R}^{13577}$. To learn the discretized operator $\mathcal{G}_h$, we employ the following MINN architecture

$$\mathbb{R}^3 \to \mathbb{R}^{100} \to V_{9h} \xrightarrow{r=0.4} V_{3h} \xrightarrow{0.2} V_h,$$

where the supports are defined according to the Euclidean distance. The corresponding dense counterpart, which servers as benchmark, is obtained by removing the sparsity constraints (equivalently, by letting the supports go to infinity). We train the networks on 50 samples and for a total of 50 epochs.

Learning a Local Nonlinear Operator

As a second experiment, we learn a nonlinear operator that is local with respect to the input. Let Ω be as in Fig. 5b. We consider the *infinitesimal area* operator $\mathcal{G} : H^1(\Omega) \to L^2(\Omega)$,

$$\mathcal{G} : u \to \sqrt{1 + |\nabla u|^2}.$$

Note in fact that, if we associate to each $u \in H^1(\Omega)$ the Cartesian surface

$$S_u := \{(x_1, x_2, u(x_1, x_2))\} \subset \mathbb{R}^3,$$

then $\mathcal{G}u$ yields a measure of the area that is locally spanned by that surface, in the sense that

$$\int_{S_u} \phi(\mathbf{s}) d\mathbf{s} = \int_\Omega \phi(u(\mathbf{x})) (\mathcal{G}u) (\mathbf{x}) d\mathbf{x}$$

for any continuous map $\phi : S_u \to \mathbb{R}$. Over the input space $\Theta := H^1(\Omega)$ we consider the probability distribution $\mathbb{P}$ induced by a Gaussian process with mean zero and covariance kernel

$$\text{Cov}(\mathbf{x}, \mathbf{y}) = \frac{1}{|\Omega|} \exp\left(-\frac{1}{2}|\mathbf{x} - \mathbf{y}|^2\right).$$

We discretize Ω using a triangular mesh of stepsize $h = 0.045$ and P1 Finite Elements, which results in a total of $N_h = 11266$ vertices. To sample from the Gaussian process, we truncate its Karhunen-Loéve expansion at $k = 100$, as that suffices to capture the complexity of the input field (actually, we mention that 40 modes would be enough to explain 95% of the field variance: here, we set $k = 100$ to not oversimplify). Conversely, the output of the operator is computed numerically by exploiting the high-fidelity mesh as a reference.

To learn $\mathcal{G}_h$ we use the MINN architecture below,

$$V_h \xrightarrow{r=0.15} V_{3h} \xrightarrow{r=0.3} V_{3h} \xrightarrow{r=0.15} V_h,$$

where the supports are intended with respect to the Euclidean metric. We train our model over 500 snapshots and for a total of 50 epochs.

Learning a Nonlocal Nonlinear Operator

Since MINNs are based on local operations, it is of interest to assess whether they can also learn nonlocal operators. To this end, we consider the problem of learning the Hardy-Littlewood Maximal Operator $\mathcal{G} : L^2(\Omega) \to L^2(\Omega)$,

$$(\mathcal{G}f)(\mathbf{x}) := \sup_{r>0} \fint_{|\mathbf{y}-\mathbf{x}|<r} |f(\mathbf{y})| d\mathbf{y}$$

which is known to be a continuous nonlinear operator from $L^2(\Omega)$ onto itself [42]. Here, we let $\Omega := B(\mathbf{0}, 1) \subset \mathbb{R}^2$ be the unit circle. Over the input space $\Theta := L^2(\Omega)$ we consider the probability distribution $\mathbb{P}$ induced by a Gaussian process with mean zero and covariance kernel

$$\text{Cov}(\mathbf{x}, \mathbf{y}) = \exp\left(-|\mathbf{x} - \mathbf{y}|^2\right).$$

As a high-fidelity reference, we consider a discretization of Ω via P1 triangular Finite Elements of stepsize $h = 0.033$, resulting in a state space V_h with $N_h = 7253$ degrees of freedom. As for the previous experiment, we sample from $\mathbb{P}$ by considering a truncated Karhunen–Loève expansion of the Gaussian process (100 modes). Conversely, the true output $u \to \mathcal{G}_h(u)$ is computed approximately by replacing the supremum over r with a maximum across 50 equally spaced radii in $[h, 2]$. To learn $\mathcal{G}_h$ we use a MINN of depth 2 with a dense layer in the middle,

$$V_h \xrightarrow{r=0.25} V_{2h} \to V_{2h} \xrightarrow{r=0.25} V_h.$$

The idea is that nonlocality can still be enforced through the use of fully connected blocks, but this are only inserted at the lower fidelity levels to reduce the computational cost. We train the architectures over 500 samples and for a total of 50 epochs. Also in this case, we build the mesh-informed layers upon the Euclidean distance.

Learning the Solution Operator of a Nonlinear PDE

For our final experiment, we consider the case of a parameter dependent PDE, which is a framework of particular interest in the literature of Reduced Order Modeling. In fact, learning the solution operator of a PDE model by means of neural networks allows one to replace the original numerical solver with a much cheaper and efficient surrogate, which enables expensive multi-query tasks such as PDE constrained optimal control, Uncertainty Quantification or Bayesian Inversion.

Here, we consider a steady version of the porous media equation, defined as follows

$$- \nabla \cdot \left(|u|^2 \nabla u \right) + u = f. \tag{7}$$

The PDE is understood in the domain Ω defined in Fig. 5d, and it is complemented with homogeneous Neumann boundary conditions. We define $\mathcal{G}$ to be the data-to-solution operator that maps $f \to u$. This time, we endow the input space with the push-forward distribution $\#\mathbb{P}$ induced by the square map $f \to f^2$, where $\mathbb{P}$ is the probability distribution associated to a Gaussian random field with mean zero and covariance kernel

$$\text{Cov}(\mathbf{x}, \mathbf{y}) = \frac{1}{1 + |\mathbf{x} - \mathbf{y}|^2}.$$

To sample from the latter distribution we exploit a truncated Karhunen–Loève expansion of the random field. This time, we set the truncation index to $k = 20$, as that is sufficient to capture 99.71% of the volatility in the Gaussian random field. Indeed, the total variance of the latter is

$$\int_{\Omega} \text{Cov}(\mathbf{x}, \mathbf{x}) d\mathbf{x} = \int_{\Omega} 1 d\mathbf{x} = \pi - (0.8)^2 \approx 2.50159,$$

while, by direct computation, the first 20 eigenvalues of the covariance operator sum up to 2.49458. For the high-fidelity discretization, we consider a mesh of stepsize $h = 0.03$ and P1 Finite Elements, resulting in $N_h = 5987$. Finally, we employ the MINN architecture below,

$$V_h \xrightarrow{r=0.4} V_{4h} \to V_{2h} \xrightarrow{r=0.2} V_h, \tag{8}$$

where the supports are defined according to the Euclidean distance. We train our model over 500 snapshots and for a total of 100 epochs. Note that, as in our third experiment, we employ a dense block at the center of the architecture. This is because the solution operator to a boundary value problem is typically nonlocal (consider, for instance, the Green formula for the Poisson equation).

3.2 Numerical Results

Table 1 reports the numerical results obtained across the four experiments. In general, MINNs perform better with respect to their dense counterpart, with relative errors that are always below 5%. As the operator to be learned becomes more and more involved, fully connected DNNs begin to struggle, eventually reaching an error of 15% in the PDE example.

Table 1 Comparison of Mesh-Informed Neural Networks and Fully Connected DNNs for the test cases in Sect. 3

Operator	N_h	Architecture	Test error (%)	Gen. error (%)
Low-dimensional manifold	13,577	Mesh-Informed	4.14	3.08
		Dense	4.78	4.07
Local area operator	11,266	Mesh-Informed	1.49	1.36
		Dense	3.89	2.54
H–L maximal operator	7253	Mesh-Informed	3.65	1.49
		Dense	6.70	3.09
Nonlinear PDE solver	5987	Mesh-Informed	4.29	3.03
		Dense	15.67	6.67

N_h = number of vertices in the (finest) mesh. Gen. error = Generalization error, defined as the gap between training and test errors. All reported errors are intended with respect to the L^2-norm, see Eq. (5)

In contrast, MINNs are able to keep up and maintain their performance. This is also due to the fact that, having less parameters, MINNs are unlikely to overfit, instead they can generalize well even in poor data regimes (cf. last column of Table 1 Consider for instance the first experiment, which counted as little as 50 training samples. There, the dense model returns an error of 4.78%, of which 4.07% is due to the generalization gap. This means that the DNN model actually surpassed the MINN performance over the training set, as their training errors are respectively of 0.71% and 1.06%. However, the smaller generalization gap allows the sparse architecture to perform better over unseen inputs.

Figures 6, 7, 8 and 9 reports some examples of approximation on unseen input values. There, we note that dense models tend to have noisy outputs (Figs. 7 and 9) and often miscalculate the range of values spanned by the output (Figs. 6 and 8). Conversely, MINNs always manage to capture the main features present in the actual ground truth. This goes to show that MINNs are built upon an effective pruning strategy, thanks to which they are able to overcome the limitations entailed by dense architectures. This phenomenon can be further appreciated in the plot reported in Fig. 10. The latter refers to the nonlinear PDE example, Eq. (7), where our MINN architecture was of the form

$$V_h \xrightarrow{r=\delta} V_{4h} \to V_{2h} \xrightarrow{r=\frac{1}{2}\delta} V_h, \tag{9}$$

with $\delta = 0.4$. Figure 10 shows how the choice of the support size, δ, affects the performance of the MINN model. Since, here, Ω has a diameter of 2, for $\delta = 4$ the architecture is formally equivalent to a dense DNN. As the support decreases, the first and the last layer become sparser, and, in turn, the architecture starts to generalize more, ultimately reducing the test error by nearly 11%. However, if the support is too small, e.g. $\delta \leq 0.125$, this may have a negative effect on the expressivity of the architecture, which now has too few degrees of freedom to properly approximate the operator of interest. In general, the best support size is to be found in the middle, $0 \leq \delta^* \leq \text{diam}(\Omega)$: of note, in this case, our original choice happens to be nearly optimal.

Aside from the improvement in performance, Mesh-Informed Neural Networks also allow for a significant reduction in the computational cost (cf. Table 2). In general, MINNs are ten to a hundred times lighter with respect to fully connected DNNs. While this is not particularly relevant once the architecture is trained (the most heavy DNN weights as little as 124 Megabytes), it makes a huge difference during the training phase. In fact, additional
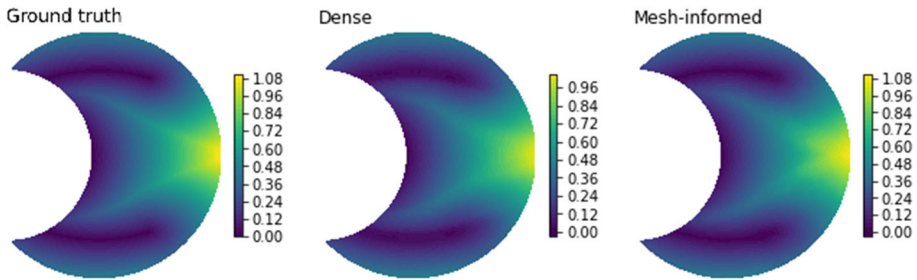
Fig. 6 Comparison of DNNs and MINNs when learning a low-dimensional manifold $\mu \rightarrow u_\mu \in L^2(\Omega)$, cf. Sect. 3.1. The reported results correspond to the approximations obtained on an unseen input value $\mu^* = [0.42, 0.04, 1.45]$.
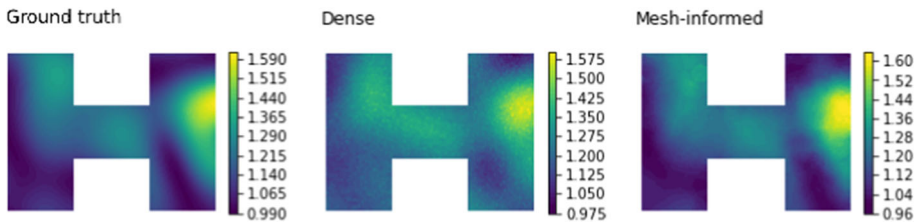

Fig. 7 Comparison of DNNs and MINNs when learning the local operator $u \rightarrow \sqrt{1 + |\nabla u|^2}$, cf. Sect. 3.1. The reported results correspond to the approximations obtained for an input instance outside of the training set
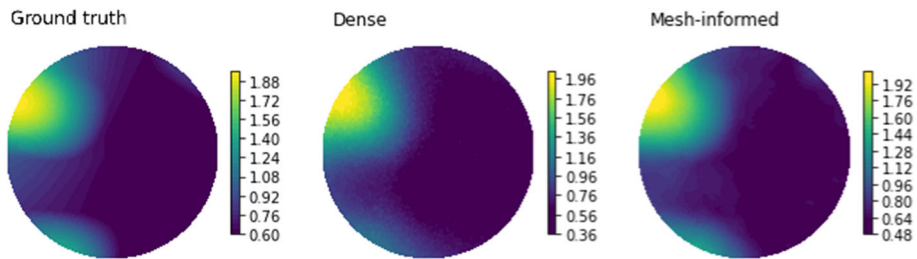

Fig. 8 Comparison of DNNs and MINNs when learning the Hardy-Littlewood Maximal Operator, cf. Sect. 3.1. The pictures correspond to the results obtained for an unseen input instance
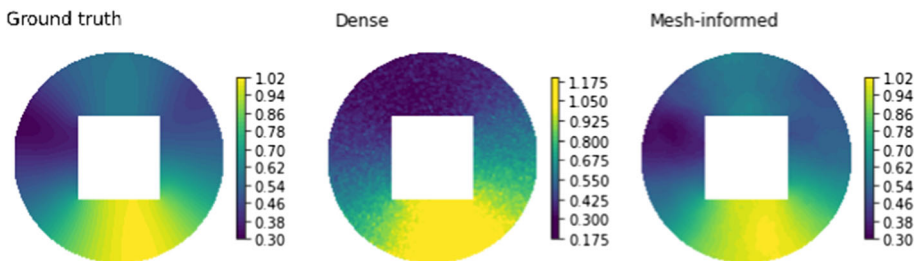

Fig. 9 Comparison of DNNs and MINNs when learning the solution operator $f \rightarrow u$ of a nonlinear PDE, cf. Sect. 3.1. The reported results correspond to the approximations obtained for an input instance f outside of the training set
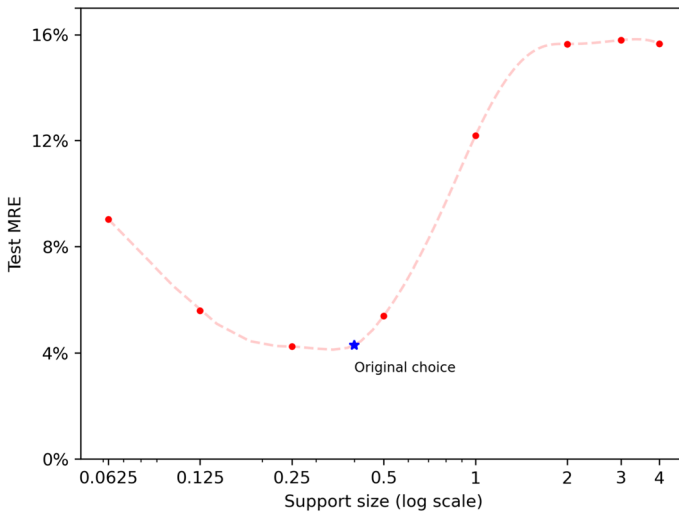
Fig. 10 Relationship between model accuracy and support size in mesh-informed layers. Case study: nonlinear PDE, see Eq. (7); MINN architecture: see Eq. (9). Red dots correspond to different choices for the support of the mesh-informed layers; cubic splines are used to draw the general trend (red dashed line). In blue, our original choice for the architecture, see Eq. (8). Errors are computed according to the L^2-norm. The x-axis is reported in logarithmic scale (Color figure online)

Table 2 Comparison of Mesh-Informed Neural Networks and Fully Connected DNNs in terms of their computational cost. dof = degrees of freedom, i.e. number of parameters to be optimized during the training phase

Architecture	dof(M)	Training speed (s/ep)	Memory usage	
			Static (Mb)	Training (Gb)
Mesh-Informed	0.3	0.64	1.3	0.01
Dense	21.7	2.47	86.9	3.49
Mesh-Informed	0.3	1.21	1.0	0.04
Dense	31.0	3.89	124.1	4.96
Mesh-Informed	1.0	0.68	3.9	0.16
Dense	12.8	1.66	51.2	2.05
Mesh-Informed	1.4	0.70	5.4	0.22
Dense	12.5	1.31	50.0	2.00

Memory usage (static) = bytes required to store the architecture. Memory usage (training) = bytes required to run a single epoch of the training phase. s/ep = seconds per epoch, M = millions, Mb = Megabytes, Gb = Gigabytes

resources are required to optimize a DNN model, as one needs to keep track of all the operations and gradients in order to perform the so-called *backpropagation* step. This poses a significant limitation to the use of dense architectures, as the entailed computational cost can easily exceed the capacity of modern GPUs. For instance, in our experiments, fully connected DNNs required more than 2 GB of memory during training, while, depending on the operator to be learned, 10 to 250 MB were sufficient for MINNs. Clearly, one could also alleviate the computational burden by exploiting cheaper optimization routines, such as first order

optimizers and batching strategies, however this typically prevents the network from actually reaching the global minimum of the loss function. In fact, we recall that the optimization of a DNN architecture is, in general, a non-convex and ill-posed problem. Of note, despite being 10 to 100 times lighter, MINNs are only 2 to 4 times faster during training. We believe that these results can be improved, possibly by optimizing the code used to implement MINNs.

4 Numerical Experiments: Handling General Nonconvex Domains

In the previous Section, when building our mesh-informed architectures, we always referred to the Euclidean distance. This is because, for our analysis, we considered spatial domains that were still quite simple in terms of shape and topology. In this sense, while most of those domains were nonconvex, the Euclidean metric was still satisfactory for quantifying distances.

In this Section, we would like to investigate this further, by testing MINNs over geometries that are far more complicated. We shall provide two examples: one concerning the flow of a Newtonian fluid through a system of channels, and one describing a reaction-diffusion equation solved on a 2D section of the human brain. In both cases, we shall not provide comparisons with other Deep Learning techniques: the only purpose for this Section is to assess the ability of MINNs in handling complicated geometries.

4.1 Stokes Flow in a System of Channels

As a first example, we consider the solution operator to the following parametrized (stationary) Stokes equation,

$$
\begin{cases}
-\Delta \mathbf{u} + \nabla p = 0 & \text{in } \Omega \\
-\nabla \cdot \mathbf{u} = 0 & \text{in } \Omega \\
p = 0 & \text{on } \Gamma_{\text{out}} \\
\mathbf{u} = 0 & \text{on } \Omega \setminus (\Gamma_{\text{in}} \cup \Gamma_{\text{in}} \cup_{i=1}^{4} \Gamma_i) \\
\mathbf{u} = \mathbf{f}_y & \text{on } \Gamma_{\text{in}} \\
\mathbf{u} = \mathbf{g}_{a_i}^{(i)} & \text{on } \Gamma_i, \quad i = 1, \dots, 4,
\end{cases}
\tag{10}
$$

where p and $\mathbf{u}$ are respectively the fluid pressure and velocity, while $\Omega \subset \mathbb{R}^2$ is the domain depicted in Fig. 11. The PDE depends on five scalar parameters. The first one, $y \in [0.1, 0.8]$ is used to parametrize the location of the inflow condition, which is given as

$$
\mathbf{f}_y(\mathbf{x}) = 100 \cdot \left[\mathbf{1}_{y-0.1, y+0.1}(x_2)((x_2 - y)^2 + 0.01), \ 0 \right]
$$

In other words, $\mathbf{f}_y$ describes a jet flow centered at y and directed towards the right. Vice versa, the remaining four parameters, $a_1, \dots, a_4 \in [0, 1]$, are used to model the presence of possible leaks at the corresponding regions $\Gamma_1, \dots, \Gamma_4$. In particular, for each $i = 1, \dots, 4$,

$$
\mathbf{g}^{(i)a_i}(\mathbf{x}) = a_i \eta_i(x_1)(1 - \eta_i(x_1)) \cdot \mathbf{n},
$$

where $\mathbf{n}$ is the external normal, while the η_i's are suitable affine transformations from $\mathbb{R} \to \mathbb{R}$ such that $\eta_i(\{x_1 \mid \mathbf{x} \in \Gamma_i\}) = [0, 1]$. Our output of interest, instead, is the velocity vector field $\mathbf{u} : \Omega \to \mathbb{R}^2$, meaning that the operator under study is
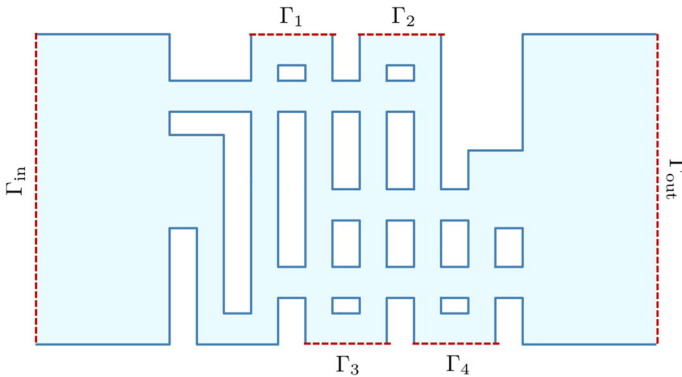
$$
\mathcal{G} : (y, a_1, \dots, a_4) \to \mathbf{u}.
$$

Fig. 11 Spatial domain for the Stokes flow example, Sect. 4.1

As high-fidelity reference, we consider a discretized setting where the domain Ω is endowed with a triangular mesh of stepsize $h = 0.08$, the pressure p is sought in the Finite Element space of piecewise linear polynomials, $Q_h = X_h^1$, while the velocity $\mathbf{u}$ is found in the space V_h of P1-Bubble vector elements [53], i.e. $X_h^1 \times X_h^1 \subset V_h \subset X_h^3 \times X_h^3$. This is done in order to ensure the numerical stability of classical Finite Element schemes associated to (10). Then, our purpose is to approximate the operator

$$\mathcal{G}_h : [0.1, 1.9] \times [0, 1]^4 \to V_h \cong \mathbb{R}^{17'050}$$

with a suitable MINN surrogate, which we build as

$$\mathbb{R}^5 \to \mathbb{R}^{100} \to V_{9h} \xrightarrow{r=1} V_h \xrightarrow{r=0.08} V_h. \tag{11}$$

Note that, even for the hidden state at the third/fourth layer, we consider a space comprised of Bubble elements (however, this is not mandatory). Here, in order to handle the fact that V_h consists of 2-dimensional vector fields, each Lagrangian node is counted twice during the construction of the mesh-informed layers. As a consequence, the mesh-informed layers will allow information to travel locally in space but also interact in between dimensions.

As usual, we employ the 0.1-leakyReLU nonlinearity at the internal layers, while we do not use any activation at output. We train our architecture over 450 randomly sampled snapshots, and for a total of 50 epochs. Following the same criteria as in Sect. 3, we train the network by optimizing the average squared error with respect to the L^2-norm, cf. Eq. 4, without using batching strategies and by relying on the L-BFGS optimizer. This time, however, in order to account for the complicated shape of the spatial domain Ω, we impose the sparsity constraints in the mesh-informed layers according to the geodesic distance. We compute the latter as in Sect. 2.3, where we exploit a coarse mesh of stepsize $h^* = 0.2$ to approximate distances. After training, our MINN surrogate reported an average L^2-error of 5.39% when tested over fifty new parameter instances, a performance that we consider to be satisfactory. In particular, as it can be further appreciated from Fig. 12, the proposed MINN architecture succeeded in learning the main characteristics of the operator under study. Of note, we mention that, in this case, implementing the dense counterpart of architecture 11, as we did back in Sect. 3), is computationally prohibitive. In fact, the latter would have $\approx 300 \cdot 10^6$ degrees of freedom, which, for instance, our GPU cannot handle.
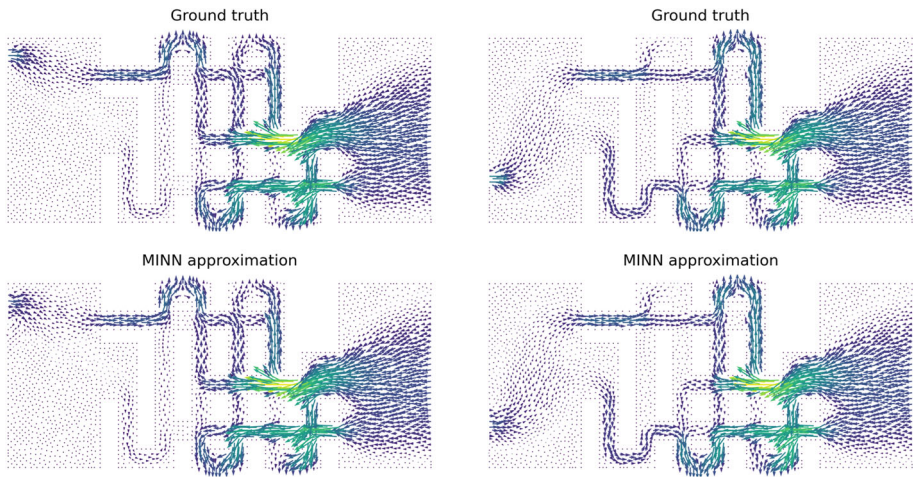
Fig. 12 Ground truth vs MINN approximation for the Stokes flow example, Sect. 4.1. The solutions reported refer to two different configurations of the parameters (not seen during training), respectively on the left/right

4.2 Brain Damage Recovery

As a second example we consider a problem concerning a nonlinear diffusion reaction of Fisher-Kolmogorov–Petrovsky–Piskunov (F-KPP) type [16] in a two-dimensional domain representing a vertical section of a human brain across the sagittal plane. Specifically, we consider the following time-dependent PDE

$$\begin{cases} \dfrac{\partial u}{\partial t} - \dfrac{1}{10}\Delta u = 10^3 u(1-u) & \text{in } \Omega \times (0, +\infty) \\ \nabla u \cdot \mathbf{n} = 0 & \text{in } \partial\Omega \times (0, +\infty) \\ u(\mathbf{x}, 0) = \varphi(\mathbf{x}) & \text{in } \Omega. \end{cases} \tag{12}$$

In the literature, researchers have used the F-KPP equation to model a multitude of biological phenomena, including cell proliferation and wound healing, see e.g. [26, 48]. In this sense, we can think of (12) as an equation that models the recovery of a damaged human brain, even though much more sophisticated models would be required to properly describe such phenomenon [46].

We thus interpret the solution to Eq. (12), $u : \Omega \times [0, +\infty) \to [0, 1]$, as a map that describes the level of healthiness of the brain at a given location at a given time (0 = highly damaged, 1 = full health). Then, the dynamics induced by the F-KPP equation resembles that of a healing process: in fact, for any initial profile $\varphi : \Omega \to [0, 1]$, later modeled as a random field, one has $u(\mathbf{x}, t) \to 1$ for $t \to +\infty$, where the speed of such convergence depends both on φ and on the point $\mathbf{x}$. In light of this, we introduce an additional map, $\tau : \Omega \to [0, +\infty)$, which we define as

$$\tau(\mathbf{x}) := \inf \{t \geq 0 \mid u(\mathbf{x}, t) \geq 0.9\}. \tag{13}$$

In other words, $\tau(\mathbf{x})$ corresponds to the amount of time required by the healing process in order to achieve a suitable recovery at the point $\mathbf{x} \in \Omega$. Our objective is to learn the operator

$$\mathcal{G} : \varphi \to \tau, \tag{14}$$

which maps any given initial condition to the corresponding time-to-recovery map. To tackle this problem, we first discretize Ω through a triangular mesh of stepsize $h = 0.0213$, over which we define the state space of continuous piecewise linear Finite Elements, V_h, with $\dim(V_h) = 2414$. Then, for any fixed initial condition φ, we solve (12) numerically by employing the Finite Element method in space and the backward Euler scheme in time ($\Delta t = 10^{-4}$). In particular, in order to compute an approximation of τ at the mesh vertices, we evolve the PDE up until the first time at which all nodal values of the solution are above 0.9. In this way, for each $\varphi \in V_h$ we are able to compute a suitable $\tau \in V_h$, with $\tau \approx \mathcal{G}(\varphi)$. Precisely, in order to guarantee that φ always takes values in [0,1], and that its realizations have an underlying spatial correlation coherent with the geometry of Ω, we cast our learning problem in a probabilistic setting where we endow the input space with the push-forward measure $\#\mathbb{P}$ induced by the map

$$g \to \frac{1}{2} \tanh{(10g + 5))} + \frac{1}{2}$$

where $\mathbb{P}$ is the probability law of a centered Gaussian random field g defined over Ω with Covariance kernel

$$\mathrm{Cov}(\mathbf{x}, \mathbf{y}) := e^{-100 d(\mathbf{x}, \mathbf{y})^2}$$

d being the geodesic distance across Ω.

As model surrogate, we consider the MINN architecture reported below,

$$V_h \xrightarrow{r=0.05} V_{\frac{5}{2}h} \xrightarrow{r=0.1} V_{\frac{5}{2}h} \xrightarrow{r=0.05} V_h. \tag{15}$$

which we train over 900 random snapshots and for a total of 300 epochs. As for our previous test cases, we use the 0.1-leakyReLU activation at the internal layers and we optimize the weights by minimizing the mean squared L^2-error: to do so, we rely on the L-BFGS optimizer, with default learning rate and no mini-batching. Note that, as for Sect. 4.1, here the supports of the mesh-informed layers are computed following the geodesic distance (stepsize of the auxiliary coarse mesh: $h^* = 0.107$). However, differently from the previous test case, we only employ mesh-informed layers. This is because, intuitively, we expect the phenomenon under study to be mostly local in nature.

After training, our architecture reported an average L^2-error of 7.40% (computed on 100 randomly sampled unseen instances). Figure 13 shows the initial condition of the brain for a given realization of the random field φ, together with its high-fidelity approximation for τ and the corresponding MINN prediction. Once again, we see a good agreement between the actual ground truth and the neural network output. For instance, the proposed MINN model succeeds in recognizing a fundamental feature of the healing process, that is: despite reporting an equivalent damage, two different regions of the domain may require completely different times for their recovery; in particular, those regions that are more isolated will take longer to heal.

5 Numerical Experiments: Comparison with DeepONets and Fourier Neural Operators

Our purpose for the current Section is to compare the performances of MINN architectures with those of two other popular Deep Learning approaches to operator learning, namely
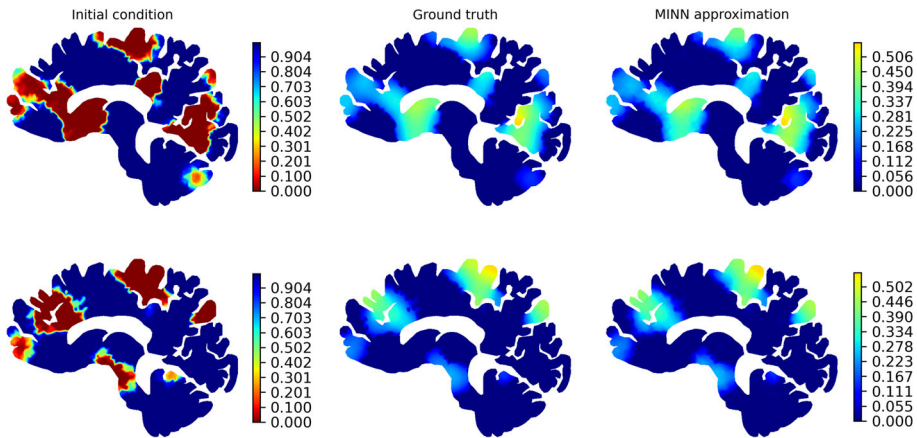
Fig. 13 Initial condition (right), ground truth of the time-to-recovery map (center) and corresponding MINN approximation (right) for an unseen instance of the *brain damage recovery* example, Sect. 4.2. Note: a different colorbar is used for the initial condition and the time-to-recovery map, respectively (Color figure online)

DeepONets and Fourier Neural Operators (FNO). To start with, we compare the three methodologies on a benchmark case study, where the performances of DeepONets and FNO have already been reported [41]. We then move on to consider two additional problems which, despite their simplicity, we find to be of remarkable interest. These concern a dynamical system with chaotic trajectories, and an advection-dominated problem, two notoriously challenging scenarios when it comes to surrogate and reduced order modeling. Here, we shall exploit these problems to showcase the possible advantages that MINNs may offer with respect to the state-of-the-art.

Here, for the sake of simplicity, we shall restrict our analysis to either 1D domains or 2D squares, which we always discretize through uniform grids. This is to ensure a fair comparison between MINNs and other approaches, such as FNOs, whose generalization to more complex geometries is still in the progress of being (see, e.g., the recently proposed Laplace Neural Operators, [7, 9]). We report our results in the subsections below. Before that, however, it may be worth recalling the fundamental ideas behind DeepONets and FNOs.

Let $\mathcal{G}_h : \Theta \to V_h$ be our operator of interest, where Θ is some input space, while $V_h \cong \mathbb{R}^{N_h}$ is a given Finite Element space of dimension N_h, here consisting, for simplicity, of piecewise linear polynomials. As we mentioned in Sect. 2.5.2, DeepONets are grounded on a representation of the form

$$(\mathcal{G}_h f)(\mathbf{x}) \approx \Psi(f) \cdot \phi(\mathbf{x}),$$

where $\Psi : \Theta \to \mathbb{R}^m$ and $\phi : \Omega \to \mathbb{R}^m$ are the branch and the trunk nets, respectively. Let now $\{\mathbf{x}_j\}_{j=1}^{N_h}$ be the nodes of the mesh at output, and let $\mathbf{V} \in \mathbb{R}^{N_h \times m}$ be the matrix

$$\mathbf{V} := \begin{bmatrix} \phi_1(\mathbf{x}_1) & \ldots & \phi_m(\mathbf{x}_1) \\ \vdots & & \vdots \\ \phi_1(\mathbf{x}_{N_h}) & \ldots & \phi_m(\mathbf{x}_{N_h}) \end{bmatrix}, \tag{16}$$

where $\phi_i(\mathbf{x})$ denotes the ith component of $\phi(\mathbf{x})$. We note that the map

$$\Theta \ni f \to \mathbf{V} \cdot \Psi(f) \in \mathbb{R}^{N_h} \cong V_h, \tag{17}$$

formally returns the DeepONet approximation across the overall output mesh. In particular, this opens the possibility of directly replacing the trunk net with a suitable projection matrix $\mathbf{V}$. As a matter of fact, this is the strategy proposed by Lu et al. in [41], where the authors exploit Proper Orthogonal Decomposition (POD) [52] to compute the projection matrix $\mathbf{V}$ in an empirically optimal way. This technique, which the authors call POD-DeepONet, is usually better performing than the vanilla implementation of DeepONet, but it requires fixing the mesh discretization at output. Since the latter condition is always true within our framework, we shall restrict our attention to POD-DeepONet for our benchmark analysis.

For what concerns FNOs instead, the fundamental building block of these architectures is the *Fourier layer*. Similarly to convolutional layers, these architectures are defined to accept inputs with multiple *channels* or features. In our setting, for a given feature dimension c, a Fourier layer is a map $L : (V_h)^c \to (V_h)^c$ of the form

$$L(\boldsymbol{v}) = \rho \left(\mathcal{F}^{-1} \left(\mathbf{R} \cdot \mathcal{F}(\boldsymbol{v}) \right) + W\boldsymbol{v} \right) \tag{18}$$

where

- $\mathcal{F} : (V_h)^c \to \mathbb{R}^{m \times c}$ is the truncated Fourier transform, which takes the c-signals at input, computes the Fourier transform of each of them and only keeps the first m coefficients;
- $\mathcal{F}^{-1} : \mathbb{R}^{m \times c} \to (V_h)^c$ is the inverse Fourier transform, defined for each channel separately;
- $\mathbf{R} = (r_{i,j,k})_{i,j,k} \in \mathbb{R}^{m \times m \times c}$ is a learnable tensor that performs a linear transformation in the Fourier space. For any given input $\mathbf{A} = (a_{j,k})_{j,k} \in \mathbb{R}^{m \times c}$, the action of the latter is given by

$$\mathbf{R} \cdot \mathbf{A} := \left(\sum_{j=1}^{m} r_{i,j,k} a_{j,k} \right)_{i,k} \in \mathbb{R}^{m \times c};$$

- $W : V_h \to V_h$ is a linear map whose action is given by

$$(W\boldsymbol{v})(\mathbf{x}) := \mathbf{W} \cdot \boldsymbol{v}(\mathbf{x}),$$

  where $\mathbf{W} \in \mathbb{R}^{c \times c}$ is a learnable weight matrix;
- ρ is the activation function.

In practice, when approximating a given operator $\mathcal{G}_h : \Theta \to V_h$, the implementation of a complete FNO architecture is often of the form

$$P \circ L_l \circ \cdots \circ L_1 \circ E \circ \tilde{\Phi} : \quad \Theta \to V_h \to (V_h)^c \to \cdots \to (V_h)^c \to V_h. \tag{19}$$

Here, Φ is a preliminary block, possibly consisting of dense layers, that maps the input form Θ to V_h. The FNO that exploits a lifting layer, E, to increase the number of channels, from V_h to $(V_h)^c$. Such lifting is typically performed by composing the signal at input with a shallow (dense) network, $e : \mathbb{R} \to \mathbb{R}^c$, so that $E : v(\mathbf{x}) \to e(v(\mathbf{x}))$. The model then implements l Fourier layers, which constitute the core of the FNO architecture. Finally, a projection layer is used to map the output back from $(V_h)^c$ to V_h. As for the lifting operator, this is achieved through the introduction of a suitable projection network, $p : \mathbb{R} \to \mathbb{R}^c$, acting node-wise, i.e. $P : \boldsymbol{v}(\mathbf{x}) \to p(\boldsymbol{v}(\mathbf{x}))$.

5.1 Benchmark Problem: Diffusion Equation with Random Coefficients

To start, we consider a test case where the performances of DeepONets and FNOs have already been reported, see, respectively [41] and [39]. Let $\Omega = (0, 1)^2$. We wish to learn the

solution operator $\mathcal{G} : L^\infty(\Omega) \to L^2(\Omega)$ of the boundary value problem below

$$\begin{cases} -\nabla \cdot (a\nabla u) = 1 & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{20}$$

where $\mathcal{G} : a \to u$. As usual, we cast our learning problem in a probabilistic setting by endowing the input space, $L^\infty(\Omega)$, with a suitable probability distribution $\mathbb{P}$. In short, the latter is built so that a satisfies

$$\mathbb{P}\left(a(\mathbf{x}) \in \{3, 12\} \ \forall \mathbf{x} \in \Omega\right) = 1,$$

meaning that the PDE is characterized by a random permeability field with piecewise constant realizations. For a more accurate description about the construction of such $\mathbb{P}$, we refer to [39, 41].

Following the same lines of [41], we discretize the spatial domain with a uniform 29x29 grid, which we can equivalently think of as a triangular rectangular mesh, $\mathcal{M} = \{\mathbf{x}_i\}_{i=1}^{841}$, of uniform stepsize $h = \sqrt{2}/28 \approx 0.05$. Both for training and testing, we refer to the dataset made available by the authors in [41], which consists of 1000 training snapshots and 200 testing instances, respectively. As detailed in [39], the latter were obtained by repeatedly solving Problem (20) via finite-differences. Consequently, for each random realization of the permeability field, we have access to the nodal values $\mathbf{a} = [a(\mathbf{x})]_{\mathbf{x}\in\mathcal{M}} \in \mathbb{R}^{N_h}$ of the random field, and the corresponding approximations of the PDE solution, $\mathbf{u} \approx [u(\mathbf{x})]_{\mathbf{x}\in\mathcal{M}} \in \mathbb{R}^{N_h}$, where $N_h = 841$. In this sense, we can think of the discrete operator as $\mathcal{G}_h : V_h \to V_h$, with $V_h = X_h^1$, even though it would be more natural to have X_h^0 as input space to account for the discontinuities in a.

It is also in light of these considerations that we propose to learn to operator with the MINN architecture below,

$$X_h^1 \xrightarrow{r=0.1} X_{4h}^0 \xrightarrow{r=0.4} X_{4h}^0 \xrightarrow{r=0.2} X_h^1 \xrightarrow{r=0.2} X_h^1 \xrightarrow{r=0.1} X_h^1 \xrightarrow{r=0.2} X_h^1. \tag{21}$$

The idea is that the first two layers should act as a preprocessing of the input. In particular, aside from the dimensionality reduction, they serve the additional purpose of recasting the original signal over the space of P0 Finite Elements, which we find to be better suited for representing permeability fields.

This time, we employ the 0.3-leakyReLU activation for all the layers, including the last one. This allows us to enforce, at least in a relaxed fashion, the fact that any PDE solution to (20) should be nonnegative. We train our model for 300 epochs by minimizing the mean squared L^2-error through the L-BFGS optimizer (no batching, default learning rate), which on our GPU takes only 1 min an 56 s.

After training, our model reports an average L^2-error of 2.78%, which we find to be satisfactory as it compares very well with both DeepONets and FNOs (cf. Table 3). In general, as it can be observed from Fig. 14, the proposed MINN architecture can reproduce the overall behavior of the PDE solutions fairly well, but it fails in capturing some of their local properties. Considering that our model is twice as accurate over the training snapshots than it is over the test data (average relative error: 1.38% vs 2.78%), this might be due to a reduced amount of training instances.

Nonetheless, our performances remain comparable with those achieved by the state-of-art. Here, DeepONets attain the best accuracy thanks to their direct usage of the POD projection, which is known to be particularly effective for elliptic problems [52]. Indeed, as shown in [41], the same approach yields an average L^2-error of 2.91% if one replaces the POD basis with a classical trunk-net architecture. FNOs, instead, report the worst performance,

Table 3 Comparison of Mesh-Informed Neural Networks, DeepONets and Fourier Neural Operators (FNO), for the diffusion equation example, Sect. 5.1

	MINN	FNO	POD-DeepONet
L^2-relative error	2.78%	4.83%	2.32%
Layers	6 Mesh-informed	4 Fourier	2 Dense
		2 Feature maps	2 Convolutional
Hyperparameters	$h_{\text{coarse}} = 4h$	$c = 32$	$m_{\text{POD}} = 115$
		$m = 12$	

All errors correspond to the average performance reported across the test set. For POD-DeepONet, the number of layers refers to the branch net only. h = stepsize of the reference mesh, h_{coarse} = stepsize of the coarser mesh, c = no. of features for the Fourier layers, m = no. of Fourier modes, m_{POD} = no. of POD basis functions
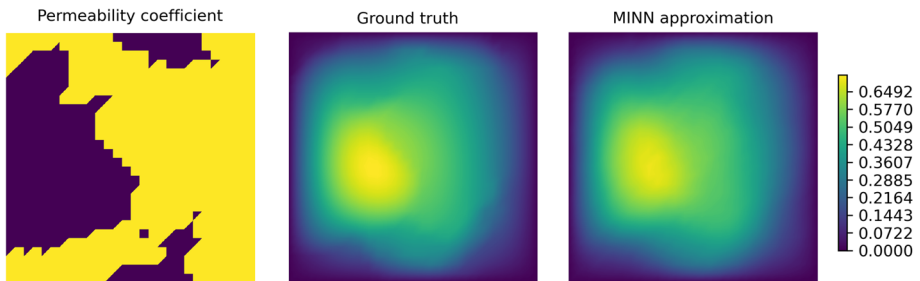

Fig. 14 Input field, ground truth and MINN approximation for a test instance of Problem (20), Sect. 5.1

by their accuracy can be easily increased if one exploits suitable strategies such as *output normalization*. The latter consists in the construction of a surrogate model of the form

$$\mathcal{G}_h(a) \approx \sigma \tilde{\Phi}(a) + \bar{u},$$

where $\tilde{\Phi}$ is an FNO architecture to be learned, while

$$\bar{u} = \bar{u}(\mathbf{x}) \approx \mathbb{E}[u] \quad \text{and} \quad \sigma^2 = \sigma^2(\mathbf{x}) \approx \mathbb{E}|u - \mathbb{E}[u]|^2$$

are the pointwise average and variance of the solution field, respectively (both to be estimated directly from the training data). Then, this trick allows one to obtain a better FNO surrogate with a relative error of 2.41% [41].

5.2 Dealing with Chaotic Trajectories: The Kuramoto–Sivashinsky Equation

As a second example, we test the three methodologies in the presence of chaotic behaviors, here arising from the Kuramoto–Sivashinsky equation, a time-dependent nonlinear PDE that was first introduced to model thermal instabilities and flames propagation [35]. More precisely, we consider the periodic boundary value problem below,

$$\begin{cases} \dfrac{\partial u}{\partial t} = -\nu \dfrac{\partial^4 u}{\partial x^4} - \dfrac{\partial^2 u}{\partial x^2} - u \dfrac{\partial u}{\partial x} & \text{in } \mathbb{R} \times (0, T] \\ u(x, t) = u(x + \ell, t) & (x, t) \in \mathbb{R} \times (0, T] \\ u(x, 0) = u_0(x) & x \in \mathbb{R} \end{cases} \tag{22}$$

where $\ell = T = 100$ and $u_0(x) := \pi + \cos(2\pi x/\ell) + 0.1\cos(4\pi x/\ell)$ are given, while $\nu \in [1, 3.5]$ is a parameter. We wish to learn the operator $\mathcal{G}$ that maps ν onto the corresponding PDE solution $u = u(x, t)$. To this end, it is convenient to define the spacetime domain $\Omega := (0, \ell) \times (0, T)$, which we equip with a uniform grid of dimension 100x100. As a high-fidelity reference, we then consider a spectral method combined with a modified Crank-Nicolson scheme for time integration ($\Delta t = 5 \cdot 10^{-4}$: trajectories are later subsampled to fit the uniform grid over Ω). This allows us to define the discrete operator as

$$\mathcal{G}_h : \Theta \to V_h,$$

with where $\Theta := [1, 3.5]$ and $V_h := X_h^1(\Omega)$, so that $\dim(X_h^1) = N_h = n_h^2 = 10'000$. We use the numerical solver to compute a total of $N = 1000$ PDE solutions, $\{(\nu_i, u_h^i)\}_{i=1}^N$, $N_{\text{train}} = 500$ to be used for training and $N_{\text{test}} = 500$ for testing.

We train all the models according to the loss function below

$$\mathcal{L}(\Phi) := \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left(\frac{1}{n_h^2} \sum_{j,k=1}^{n_h} |u_h^i(x_j, t_k) - \Phi(\nu_i)(x_j, t_k)|^2 \right)$$

which we minimize iteratively, for a total of 500 epochs, using the L-BFGS optimizer. Similarly, in order to emphasize the difference between the space and the time dimension, we evaluate the accuracy of the models in terms of the relative error below,

$$\mathcal{E}(\Phi) :=$$

$$= \frac{1}{N_{\text{test}}} \sum_{i=N_{\text{test}}}^{N} \left(\frac{\frac{1}{n_h} \sum_{k=1}^{n_h} \sqrt{\frac{1}{n_h} \sum_{j=1}^{n_h} |u_h^i(x_j, t_k) - \Phi(\nu_i)(x_j, t_k)|^2}}{\frac{1}{n_h} \sum_{k=1}^{n_h} \sqrt{\frac{1}{n_h} \sum_{j=1}^{n_h} |u_h^i(x_j, t_k)|^2}} \right)$$

$$\approx \frac{1}{N_{\text{test}}} \sum_{i=N_{\text{test}}}^{N} \left(\frac{\|u_h^i - \Phi(\nu_i)\|_{L^1((0,T);\ L^2(0,L))}}{\|u_h^i\|_{L^1((0,T);\ L^2(0,L))}} \right), \tag{23}$$

where norms are intended in the Bochner sense [14]. For the implementation of the three approaches, we proceed as follows:

(i) POD-DeepONet: we exploit the training data to construct a POD basis $\mathbf{V}$ consisting of $m = 200$ modes, as those should be sufficient for capturing most of the variability in the solution space. We then construct the branch net as a classical DNN from $\mathbb{R} \to \mathbb{R}^m$, with 3 hidden layers of width 500, each implementing the 0.1-leakyReLU activation;

(ii) FNO: we use a combined architecture with a dense block at the beginning and three Fourier layers at the end. The dense block consists of 2 hidden layers of width 500, and an output layer with N_h neurons, all complemented with the 0.1-leakyReLU activation. Then, the model is followed by three Fourier layers, which, for simplicity, are identical in structure. In particular, following the same rule of thumb proposed by the authors [39], we implement a Fourier block with $c = 32$ features and $m = 12$ Fourier modes per layer. Once again, all layers (except for the last one) use the 0.1-leakyReLU as nonlinearity;

(iii) MINN: we propose the following architecture, obtained through a combination of dense and mesh-informed layers

$$\mathbb{R} \to \mathbb{R}^{50} \to V_{3h} \xrightarrow{r=20} V_h \xrightarrow{r=6} V_h,$$

where we recall that the stepsize of the (finer) spacetime mesh is $h = \sqrt{2}$. This time, however, we do not use the Euclidean metric for the computation of the supports, but

Table 4 Models comparison for the Kuramoto Sivashinsky example, Sect. 5.2

	MINN	FNO	POD-DeepONet
(L^1 of L^2)-relative error	5.87%	9.41%	6.97%
Layers	2 Mesh-informed	3 Fourier	4 Dense
	2 Dense	3 Dense	
		2 Feature maps	
Hyperparameters	$h_{\text{coarse}} = 3h$	$c = 32$	$m_{\text{POD}} = 200$
		$m = 12$	

Error are computed according to Eq. (23). Table entries for *Layers* and *Hyperparameters* read as in Table 3

we rely on a different distance function that can account for the periodicity of the spatial component. In fact, Eq. (22) is better understood in the periodic domain $\tilde{\Omega} := (\mathbb{R}/\ell\mathbb{Z}) \times [0, T]$, than it is over Ω. Here, $\mathbb{R}/\ell\mathbb{Z}$ is the quotient of $\mathbb{R}$ with respect to the equivalence relation below,

$$x \sim x' \iff (x - x')\ell^{-1} \in \mathbb{Z}.$$

To account for this, we compute the supports of the mesh-informed layers according to the following distance function $d : \Omega \times \Omega \to [0, +\infty)$,

$$d((x, t), (x', t')) := \sqrt{\left(\frac{\ell}{2} - \left|\frac{\ell}{2} - |x - x'|\right|\right)^2 + (t - t')^2}.$$

This ensures the wished behavior over the unwrapped domain Ω. In fact, for instance, one has

$$d((1, 0), (3, 0)) = d((1, 0), (99, 0)),$$

as we recall that $\ell = 100$.

After training, all model surrogates report relative errors below 10%, with MINNs achieving the best performance, cf. Table 4. When tested over parametric instances outside of the training set, the three approaches propose similar but different predictions, cf. Fig. 15. The FNO-surrogate manages to capture the macroscopic features of the solution, but returns a much noisier output. Conversely, the approximation proposed by POD-DeepONet is cleaner and better resembles the original ground truth. Still, even the latter model fails to capture some of the local features of the PDE solution, which, in contrast, the MINN architecture is able to recover. The interested reader may also find additional insights in "Appendix A".

Nonetheless, we must acknowledge that our analysis might be affected by the several design choices that we had to make for the architectures. In fact, even though we did our best in implementing and tuning the hyperparameters for the three approaches, we cannot consider our results to be universal.

5.3 Dealing with the Kolmogorov Barrier: The Advection Equation

As a last example, we consider a situation in which the problem at hand exhibits a slow decay of the *Kolmogorov n-width*. The latter is a quantity of particular interest in the areas of surrogate and Reduced Order Modeling, as it measures to which extent a given phenomenon
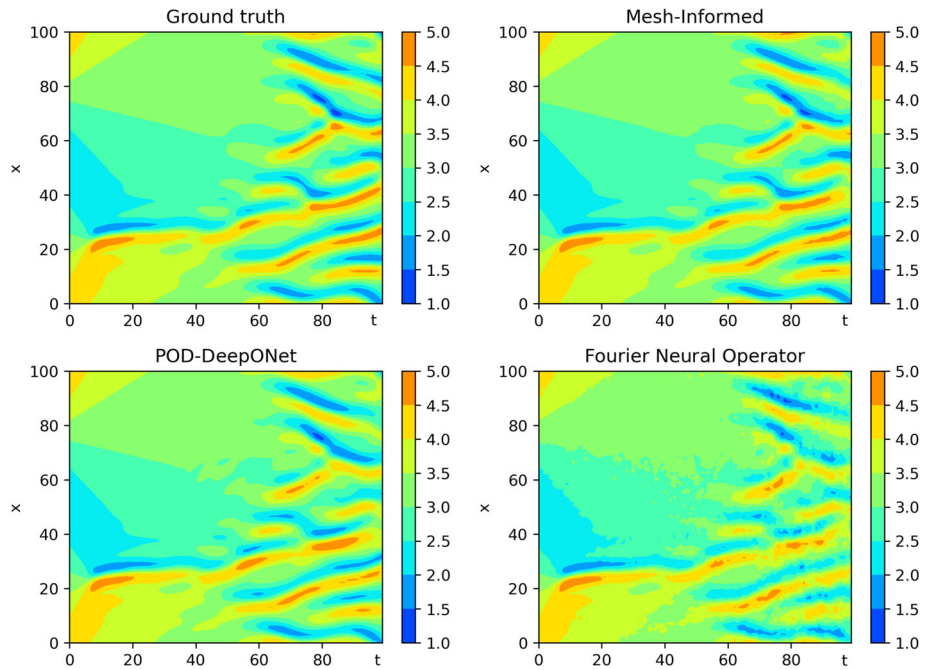
Fig. 15 Ground truth and corresponding approximations via MINN, POD-DeepONet and FNO, for a test instance of the Kuramoto–Sivashinsky example, Sect. 5.2. The solutions are plotted over the spacetime domain $\Omega = (0, \ell) \times (0, T)$.

can be described via linear superposition of suitable modes. More precisely, given a compact subset S of a normed state space $(V, \| \cdot \|)$, the Kolmogorov n-width of S is defined as

$$d_n(S) := \inf_{\substack{H \subseteq V \\ \dim(H) = n}} \sup_{u \in S} \inf_{v \in H} \|u - v\|.$$

For instance, if $d_n(S) < \varepsilon$ for some $n \in \mathbb{N}$, then there will be n modes, $v_1, \ldots, v_n \in V$, capable of representing all the elements in S with an error that is smaller than ε. Vice versa, if $d_n(S)$ decays slowly with n, then one is forced to consider a larger number of modes to obtain a reasonable approximation. As we shall discuss in a moment, this fact can have a huge impact for operator learning problems, especially for those approaches based on linear projection techniques, such as DeepONets.

To see this, consider a continuous operator $\mathcal{G} : \Theta \to L^2(\Omega)$, with $\Theta \subset \mathbb{R}^p$ a compact subset and Ω a bounded domain. Let $S := \mathcal{G}(\Theta)$ be the image of Θ through the operator $\mathcal{G}$. Then, it is straightforward to see that

$$d_n(S) < \varepsilon \iff$$

$$\exists \{v_i\}_{i=1}^n \subset L^2(\Omega), \ \{\phi_i\}_{i=1}^n \subset \mathcal{C}(\Theta) \ \text{s.t.} \ \sup_{\boldsymbol{\mu} \in \Theta} \left\| \mathcal{G}(\boldsymbol{\mu}) - \sum_{i=1}^n \phi_i(\boldsymbol{\mu}) v_i \right\|_{L^2(\Omega)} < \varepsilon,$$

$$(24)$$

furthermore, since the projection coefficients are optimal on any given (orthonormal) basis, it is not restrictive to set $\phi_i(\boldsymbol{\mu}) := \langle \mathcal{G}(\boldsymbol{\mu}), v_i \rangle_{L^2(\Omega)}$. The equivalence in (24) shows that the

effectiveness of a separation of variables approach, such as the one adopted by DeepONets and POD-DeepONets, is confined by the behavior of the Kolmogorov n-width. In particular, these approaches may encounter some difficulties if $d_n(S)$ happens to decay slowly.

In light of these considerations, we propose a final case study based on the advection equation, which, despite its simplicity, constitutes a prototypical example of slow decay in the Kolmogorov n-width. More precisely, let us consider the evolution equation below

$$
\begin{cases}
\dfrac{\partial u}{\partial t} + c\dfrac{\partial u}{\partial x} = 0 & \text{in } \Omega \times (0, T] \\[2mm]
u(0, t) = u_0(-ct) & t \in (0, T] \\[1mm]
u(x, 0) = u_0(x) & x \in \Omega
\end{cases}
\tag{25}
$$

where $\Omega = (0, 1)$, $T = 1$ and

$$
u_0(x) := \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}x^2},
$$

with $\sigma = 0.005$, are given, while the wave velocity, c, is regarded as a model parameter. Our goal is to approximate the operator

$$
\mathcal{G} : c \to u(\cdot, T),
$$

that maps the wave speed to the terminal state of the system. Equivalently, we may write $(\mathcal{G}c)(x) = u(x - c)$, as the solution to (25) is known in closed form. We exploit the latter formula to generate a collection of 1000 random snapshots, where the input parameter is sampled uniformly from $\Theta := [0.05, \ 0.95]$. We keep 750 of such snapshots for training a leave the remaining 250 for testing. For the implementation of the three Deep Learning approaches we proceed as follows:

i) POD-DeepONet: we set up two possible architectures. For the first one, we use 50 POD modes and a DNN with 2 hidden layers of width 100 as branch net. As a second architecture, instead, we consider a similar model where the number of POD basis is increased to 200;

ii) FNO: we construct a model by concatenating a shallow network of width 100, which maps $\mathbb{R} \to \mathbb{R}^{N_h}$, with a Fourier block composed of two Fourier layers. We design the latter along the same lines adopted by the authors in [39], i.e. by letting the number of Fourier modes be equal to 16 and by introducing a total of 64 hidden features.

iii) MINN: we use the hybrid architecture below

$$
\mathbb{R} \to \mathbb{R}^{100} \to \mathbb{R}^{100} \to V_{20h} \xrightarrow{r=0.1} V_{4h} \xrightarrow{r=0.01} V_h.
$$

As usual, we equip the internal layers of all our architectures with the 0.1-leakyReLU activation. Similarly, we train the three models by minimizing the mean squared L^2-error, as we did for all our previous experiments. This time, we rely on the Adam optimizer for the minimization of the loss function, which we run for a total of 50'000 epochs, starting with a learning rate of 10^{-3} and halving it every 10'000 iterations. Here, in fact, for all of the approaches we found the Adam optimizer to yield better results with respect to L-BFGS.

Results are in Table 5, Fig. 16 and "Appendix A". Unsurprisingly, the two POD-DeepONets achieve the highest errors, which is to be expected considering the nature of the problem itself. In the first case, this is due to the poorness of the POD basis, which, with as little as 50 modes, is unable to capture the overall phenomenon. In the second case, instead, this

Table 5 Test errors for the advection equation, Sect. 5.3

	MINN	FNO	POD-DeepONet
L^2-relative error	4.18%	11.08%	42.63% (50 basis)
			45.87% (200 basis)
Layers	2 Mesh-informed	3 Fourier	4 Dense
	3 Dense	3 Dense	
		2 Feature maps	
Hyperparameters	$h_{coarse} = 20h$	$c = 64$	$m_{POD} = 50 - 200$
	$h_{inter} = 4h$	$m = 16$	

Table entries as in Table 3. Here, h_{inter} = intermediate resolution
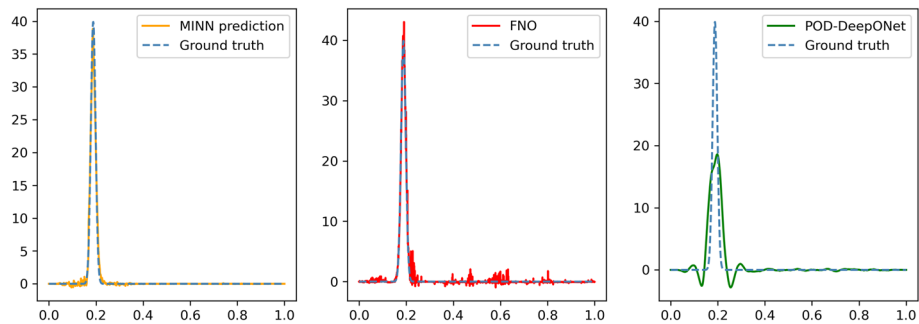
Fig. 16 Model predictions of MINN (left), FNO (center) and POD-DeepONet (right) for an unobserved parametric instance of the advection equation problem, Sect. 5.3. For POD-DeepONet, only the output of the best model is reported (50 basis)

is caused by large number of POD coefficients to be learned by the branch net. In fact, in our analysis, the largest POD basis (200 modes) reached an average projection error below 0.01%: therefore, here, the branch net is the only one responsible for the bad performance of the model.

Conversely, things improve a lot if we move to fully nonlinear methods, such as FNOs and MINNs. For instance, FNO is at least four times more accurate than POD-DeepONet, with an average relative error of 11.08%. Still, its performance is not comparable with the one achieved by our MINN surrogate (relative error: 4.18%). This difference may be further appreciated by the plots reported in Fig. 16: while all the models predict the location of the wave correctly, only the MINN surrogate manages to capture both the shape and the magnitude of the signal; in contrast the approximations proposed by FNO and POD-DeepONet present spurious oscillations and are either noisy or off-scale. This goes to show that even an apparently simple phenomenon can give rise to highly nontrivial complications.

6 An Application to Uncertainty Quantification

We finally consider an application to Uncertainty Quantification (UQ) involving a partial differential equation. UQ is an essential aspect of robust modelling, which often involves expensive numerical and statistical routines. In this Section, we provide an example on how
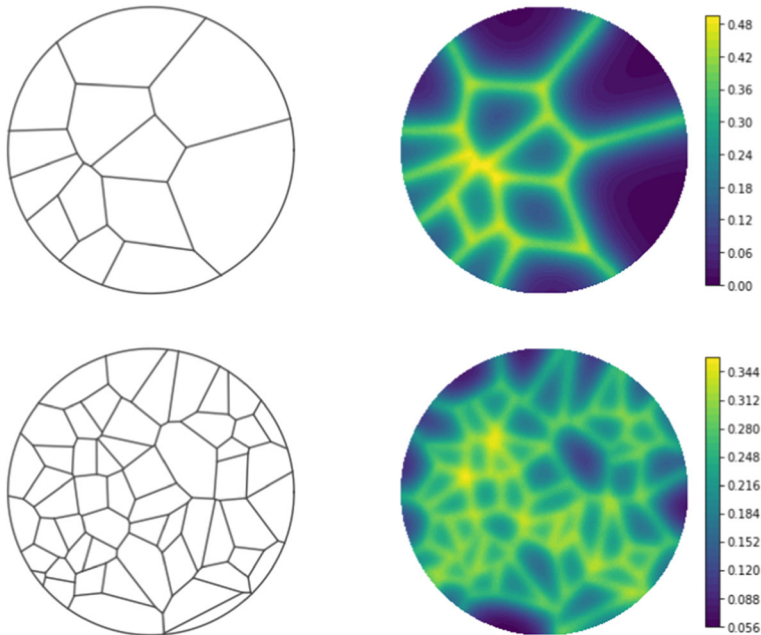
Fig. 17 Forward UQ problem (Sect. 6). Topology of the microvascular network Λ (left) and corresponding oxygen distribution in the tissue $u = u_\Lambda$ (right). The top and the bottom rows corresponds respectively to a poorly and a highly vascularized tissue (resp. $\lambda = 1$ and $\lambda = 3$). Globally, the two networks provide the same amount of oxygen (cf. Eq. 26), but their topology significantly affects the values of u in the tissue. In the first case (top row), nearly 31% of the tissue has an oxygen level below the threshold value $u_* := 0.1$. Conversely, only 3% of the tissue reports a low oxygen concentration in the second example

MINNs can alleviate these costs by serving as model surrogates in the computational pipeline. In particular, starting from a suitable PDE model, we address a problem concerning oxygen transfer in biological tissues.

6.1 Model Description

Oxygen is a fundamental constituent of most biological processes. In humans, oxygen is delivered by the circulatory system from the lungs to the rest of the body. At the small scales, cells receive oxygen from the vascular network of capillaries that spread all over the body. An efficient oxygen transfer is fundamental to ensure a healthy micro-environment and abnormal values in oxygen concentration are often associated to pathological scenarios. In particular, *hypoxia*, that is the shortage of oxygen supplies, plays an important role in the development and treatment of tumors. It has been shown that hypoxic tissue opposes a resistance to chemotherapy and radiotherapy [8, 44]. These issues are caused by perturbed properties of the tumor blood vessels in terms of morphology and phenotype. Here, we aim at developing a methodology to assess the role of vascular morphology on tissue hypoxia. More precisely, we wish to address the following question: how does the topology of the vascular network relate to the size of the tissue under hypoxia? We answer this question in the simplified setting that we describe below.

Within an idealized setting, we consider a portion of a vascularized tissue $\Omega := \{\mathbf{x} \in \mathbb{R}^2 : |x| < 1\}$. Let $\Lambda \subset \overline{\Omega}$ be a graph representing the vascular network of capillaries (cf.

Fig. 17) and let $u : \Omega \to [0, 1]$ be the oxygen concentration in the tissue, normalized to the unit value. We model the oxygen transfer from the network to the tissue with the following equations,

$$\begin{cases} -\alpha \Delta u + u = (1 - u)\delta_\Lambda & \text{in } \Omega \\ -\alpha \nabla u \cdot \mathbf{n} = \beta u & \text{on } \partial\Omega \end{cases}$$

where $\alpha = 0.1$ and $\beta = 0.01$ are respectively a fixed diffusion and resistance coefficient, while δ_Λ is the unique singular measure for which

$$\int_\Omega v(\mathbf{x})\delta_\Lambda(d\mathbf{x}) = \frac{1}{|\Lambda|} \int_\Lambda v(\mathbf{s})d\mathbf{s}$$

for all $v \in \mathcal{C}(\overline{\Omega})$. Here, we denote by $|\Lambda| := \int_\Lambda 1 d\mathbf{s}$ the total length of the vascular graph. The first equation in (6.1) describes the diffusion and consumption of oxygen, balanced according to the amount released from the vascular network on the right hand side. Finally, the model is closed using resistance boundary conditions of Robin type.

We understand (6.1) in the weak sense, meaning that define $u = u_\Lambda$ as the unique solution to the problem below

$$\int_\Omega \alpha \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x})d\mathbf{x} + \int_\Omega u(\mathbf{x})v(\mathbf{x})d\mathbf{x} + \int_{\partial\Omega} \beta u(\mathbf{s})v(\mathbf{s})d\mathbf{s} =$$
$$= \frac{1}{|\Lambda|} \int_\Lambda (1 - u(\mathbf{s}))v(\mathbf{s})d\mathbf{s} \tag{26}$$

where the above is to be satisfied for all $v \in H^1(\Omega)$.

6.2 Uncertainty Quantification Setting

As we mentioned previously, we are interested in the relationship between Λ and u. To this end, we introduce the parameter space

$$\Theta := \{\Lambda \subset \overline{\Omega} : \ \Lambda \text{ is the union of finitely many segments}\}$$

which consists of all vascular networks. Note that, due to the normalizing factor $1/|\Lambda|$ in (26), all the vascular networks actually provide the same global amount of oxygen. However, as we will see later on, only those vascular graphs that are sufficiently spread across the domain can ensure a proper oxygen supply to the whole tissue (cf. Fig. 17). In other words, we explore the influence of the distribution of the network, rather than its density, on the oxygen level.

The next subsection is devoted to prescribing a suitable discretization of (26) to work with, and to introduce a class of probability measures $\{\mathbb{P}_\lambda\}_\lambda$ defined over Θ. The idea is the following. We will use a macro-scale parameter λ to describe the general perfusion of the tissue. Higher values of λ will correspond to a highly vascularized tissue. This means that the topology of the vascular network will still be uncertain, but the corresponding probability distribution $\mathbb{P}_\lambda$ will favor dense graphs. Conversely, lower values of λ will describe scenarios where capillaries are more sparse (see Fig. 17, top vs bottom row). This will then bring us to consider the family of random variables

$$Q_\lambda := \frac{1}{|\Omega|}|\{u_\Lambda < 0.1\}| \text{ with } \Lambda \sim \mathbb{P}_\lambda, \tag{27}$$

that measure the portion of the tissue under the oxygen threshold 0.1, which we take as the value under which hypoxia takes place. Our interest will be to estimate the probability density function of each Q_λ and to provide a robust approximation of their expected value $\mathbb{E}[Q_\lambda]$. While these tasks can be achieved using classical Monte Carlo, the computational cost is enormous as it implies solving Eq. (26) repeatedly. To alleviate this burden, we will replace the original PDE solver with a suitable MINN architecture trained to learn the map $\Lambda \to u_\Lambda$.

6.3 Discretization and Implementation Details

For the random generation of vascular networks we exploit Voronoi diagrams [3]. Let $\mathcal{P} := \{P \subset \Omega \mid P \text{ finite}\}$ be the collection of all points tuples in Ω. To any $P \in \mathcal{P}$, we associate the vascular graph $\Lambda(P)$ defined by the edges of the Voronoi cells generated by P. In this way, we obtain a correspondence $\mathcal{P} \to \Theta$ given by $P \to \Lambda(P)$, that we can exploit to prescribe probability measures over Θ. To this end, let $\lambda > 0$ and let X_λ be a Poisson point process over Ω having a uniform intensity of 10λ. We denote by $\tilde{\mathbb{P}}_\lambda$ the probability measure induced by X_λ over $\mathcal{P}$. Then, we define $\mathbb{P}_\lambda := \#\tilde{\mathbb{P}}_\lambda$ as the push-forward measure obtained via the action $P \to \Lambda(P)$. This ensures the wished behavior: higher values of λ tend to generate more points in the domain and, consequently, denser graphs.

We now proceed to discretize the variational problem. As a first step, we note that the vascular graph Λ is not given in terms of a parametrization, which makes it harder to compute integrals of the form $\int_\Lambda v(\mathbf{s})d\mathbf{s}$. As an alternative, we consider the smoothed approximation below,

$$\int_\Lambda v(\mathbf{s})d\mathbf{s} \approx \int_\Omega v(\mathbf{x})\phi_\Lambda(\mathbf{x})d\mathbf{x}, \tag{28}$$

where

$$\phi_\Lambda := \frac{1}{\epsilon^2}\max\{\epsilon - \text{dist}(\mathbf{x}, \Lambda), 0\}.$$

Here, $\epsilon > 0$ is a smoothness parameter that we fix to $\epsilon = 0.05$. It is not hard to prove that, for each $v \in \mathcal{C}(\overline{\Omega})$ fixed, the right-hand side of Eq. (28) converges to the left hand-side as $\epsilon \to 0$: for a detailed proof, the reader may refer to "Appendix B". Then, our operator of interest becomes $\mathcal{G} : \phi_\Lambda \to u_\Lambda$, and we can proceed with our usual discretization via P1 Finite Elements. To this end, we discretize the domain using a triangular mesh of stepsize $h = 0.03$, which results in $N_h = 7253$ degrees of freedom. Then, we allow λ to vary uniformly in [1,10] and we generate a total of 4500 training snapshots according to the probability distributions introduced previously. We exploit these snapshots to train the MINN model below,

$$V_h \xrightarrow{r=0.1} V_{3h} \to V_{3h} \xrightarrow{r=0.1} V_h,$$

where the architecture has been defined in analogy to the one employed for the nonlinear PDE in Sect. 3.1. The network is trained for a total of 50 epochs and using the same criteria presented in Sect. 3.

6.4 Results

Once trained, the Mesh-Informed Neural Network reported an average L^2-error of 4.99%, with errors below 10% for 488 out of 500 test instances. We considered these results satisfac-
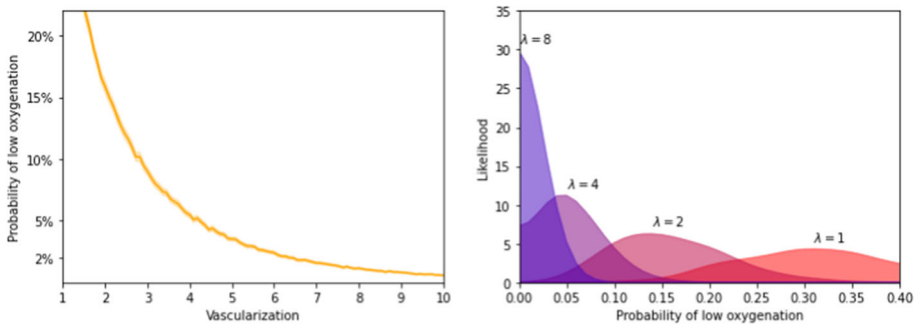
Fig. 18 Results for the UQ problem in Sect. 6. Left panel: expected probability of low oxygenation, $\mathbb{E}[Q_\lambda]$, as a function of the vascularization level λ. Confidence bands are computed pointwise using a 99% confidence level. Right panel: probability distribution of Q_λ for different values of λ. Colors fade from red to purple as λ grows (Color figure online)

tory and we proceeded to sample a total of 100'000 solutions using our DNN model. More precisely, we considered 100 equally spaced values of λ in [1,10], and for each of those we sampled 1'000 independent solutions. From there, we obtained an i.i.d. sample of size 1'000 for each of the Q_{λ_i}, where $\lambda_i = \{1 + i/11\}_{i=0}^{99}$. Results are in Fig. 18.

The left panel of Fig. 18 shows the approximation of the map $\lambda \to \mathbb{E}[Q_\lambda]$. As the tightness of the 99% confidence bands suggests, the estimate is rather robust. Spurious oscillations are most likely due to the numerical errors introduced by the MINN model, rather than from statistical noise. Coherently with the physical interpretation of λ, we see that the probability of low oxygenation decreases with the vascular density. Interestingly, although the total intensity of the source term is normalized to the same level in any configuration, the networks with higher gaps between neighboring edges are prone to spots of low oxygen concentration. Not only, the decay appears to be exponential. Further investigations seem to confirm this intuition, as we obtain an R^2-coefficient of 0.987 when trying to relate λ and $\log \mathbb{E}[Q_\lambda]$ via linear regression. Conversely, the right panel of Fig. 18 shows how the probability distribution of Q_λ changes according to λ. The densities are more spread out when λ is near 1, while they shrink towards zero as λ increases. This is coherent with the physical intuition, and we would expect the density of Q_λ to converge to a Dirac delta as $\lambda \to +\infty$.

In real scenarios where the physical complexity of a vascularized tissue is appropriately described as in [44], this analysis would be computationally viable only with the employment of the MINN model as a surrogate for the numerical solver. In the case presented here, both the full order model and the surrogate model are computationally inexpensive. However, the former required around 2 min to generate 1'000 PDE solutions. Conversely, the trained DNN model was able to provide the same number of solutions in as little as 3 milliseconds, corresponding to a speed up factor of approximately 40. For multiphysics models where a simulation of a single point in the parameter space could cost hours of wall computational time, such gain could enable approaches that would be otherwise unreasonable. Even in the present simplified setting, such a boost also makes up for the computational effort required to train the network. In fact: (i) collecting the training snapshots took 575.96 s, (ii) training the MINN model required 125.32 s, (iii) generating the 100'000 new solutions took 0.3 s. In contrast, the numerical solver would generate at most $\approx 5'500$ solutions within the same amount of time. These considerations support the interest in further developing model order reduction techniques based on deep neural networks that are robust for general spatial domains, such as MINNs. In fact, we are currently developing model reduction techniques applied to

realistic models of the vascular microenvironment that leverage on the DL-ROM framework, previously developed in [18–20], combined with the efficiency of MINNs.

7 Conclusions

In this paper, we have introduced Mesh-Informed Neural Networks (MINNs), a novel class of sparse DNN models that can be used to learn general operators between infinite dimensional spaces. The approach is based on an *a priori* pruning strategy that is obtained by embedding the hidden states into discrete functional spaces of different resolution. Despite being very easy to implement, MINNs show remarkable advantages with respect to dense architectures, such as a massive reduction in the computational costs and an increased ability to generalize over unseen samples. This is coherent with the results available in the pruning literature [5], even though the setting differs from the one considered thereby.

We have tested MINNs over a large variety of scenarios, going from low dimensional manifolds to parameter dependent PDEs, showing that these architectures can be a competitive alternative for learning nonlinear operators in the presence of complex spatial domains. This opens a wide new range of research directions that we wish to investigate further in future works. For instance, one could test the use of MINNs in more sophisticated Deep Learning based Reduced Order Models for PDEs (DL-ROMs), such as those in [18–20, 38]. In addition, considering how MINNs are actually built, it would be also interesting to see whether one can take advantage of multi-fidelity strategies during the training phase, as in [30]. Finally, another intriguing research question is whether one can characterize the approximation properties of such architectures, similarly to what other researchers have already done for FNOs and DeepONets, see e.g. [32] and [37], respectively.

Data Availibility Enquiries about data availability should be directed to the authors.

Declarations

Competing interests The authors have not disclosed any competing interests.

Appendix A: Supplementary Material for Sect. 5
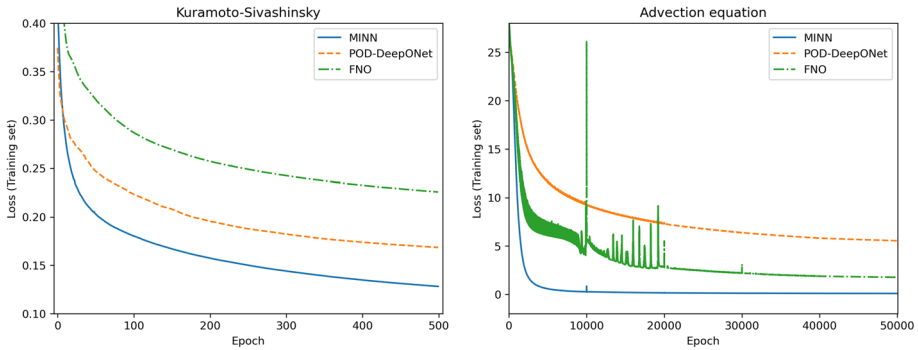
See Fig. 19 and Table 6.

Fig. 19 Loss optimization for MINN, POD-DeepONet and FNO, for the two case studies reported in Sects. 5.2 and 5.3. For the advection equation, only the training of the best POD-DeepONet is reported (50 basis)

Table 6 Training times for MINN, POD-DeepONet and FNO, for the two case studies reported in Sects. 5.2 and 5.3

Case study	MINN	POD-DeepONet	FNO
Kuramoto–Sivashinsky Sect. 5.2	12 m 7 s	3 m 12 s	1 h 33 m 17 s
Advection equation Sect. 5.3	5 m 6 s	4 m 7 s	6 h 56 m 29 s

For the advection equation, only the training time of the best POD-DeepONet is reported (50 basis)

Appendix B: Auxiliary Result for Sect. 6

Lemma 1 *Let Ω be a Lipschitz domain. Let $\Lambda \subset \overline{\Omega}$ be the union of finitely many segments, where each segment intersects $\partial\Omega$ in at most two points (the extremes). For any fixed $v \in \mathcal{C}(\overline{\Omega})$ one has*

$$\frac{1}{\epsilon^2} \int_{\Omega} v(\mathbf{x}) \max\{\epsilon - \operatorname{dist}(\mathbf{x}, \Lambda), 0\}\, d\mathbf{x} \to \int_{\Lambda} v(\mathbf{s})d\mathbf{s} \quad as\ \epsilon \downarrow 0^+.$$

Proof Let $\varphi_{\Lambda}^{\epsilon}$ be the (unscaled) kernel

$$\varphi_{\Lambda}^{\epsilon}(\mathbf{x}) := \max\{\epsilon - \operatorname{dist}(\mathbf{x}, \Lambda), 0\}.$$

By definition, we note that $\varphi_{\Lambda}^{\epsilon}$ vanishes outside of the set $\Lambda + B(0, \epsilon) := \{\mathbf{x} + \epsilon\mathbf{v} \mid \mathbf{x} \in \Lambda,\ \mathbf{v} \in B(0, 1)\}$. We now proceed in three steps.

Step 1 We start by proving that the lemma holds whenever Λ is composed by a single segment. Without loss of generality, we let $\Lambda = [0, 1] \times \{0\}$. For the sake of simplicity, we further assume that $\Lambda \cap \partial\Omega = \emptyset$. The case in which Λ has an extreme on the boundary can be handled similarly by exploiting the Lipschitz regularity of $\partial\Omega$. Let $\epsilon < \operatorname{dist}(\Lambda, \partial\Omega)$, so that $\Lambda + B(0, \epsilon) \subset \Omega$. By direct computation we have

$$\int_{\Omega} v(\mathbf{x})\varphi_{\Lambda}^{\epsilon}(\mathbf{x})d\mathbf{x} = \frac{1}{\epsilon^2}\int_{\Lambda + B(0,\epsilon)} v(\mathbf{x})\varphi_{\Lambda}^{\epsilon}(\mathbf{x})d\mathbf{x} =$$

$$= \frac{1}{\epsilon^2}\int_{A_{\epsilon} \cup B_{\epsilon}} v(\mathbf{x})\varphi_{\Lambda}^{\epsilon}(\mathbf{x})d\mathbf{x} + \frac{1}{\epsilon^2}\int_{[0,1]\times[-\epsilon,\epsilon]} v(\mathbf{x})\varphi_{\Lambda}^{\epsilon}(\mathbf{x})d\mathbf{x}$$

where A_{ϵ} and B_{ϵ} are two half circles of radius ϵ respectively centered at the extremes of the segment Λ. It is easy to see that the first contribute vanishes as $\epsilon \downarrow 0^+$. In fact, since

$$||\varphi_\Lambda^\epsilon||_\infty = \epsilon,$$

$$\left| \frac{1}{\epsilon^2} \int_{A_\epsilon \cup B_\epsilon} v(\mathbf{x})\varphi_\Lambda^\epsilon(\mathbf{x})d\mathbf{x} \right| \leq \frac{1}{\epsilon^2}||v||_\infty \cdot \epsilon|A_\epsilon \cup B_\epsilon| = \pi\epsilon||v||_\infty.$$

Conversely, for the second term we have

$$\int_0^1 \frac{1}{\epsilon^2} \int_{-\epsilon}^\epsilon v(x_1, x_2)\varphi_\Lambda^\epsilon(x_1, x_2)dx_2dx_1 = \int_0^1 \frac{1}{\epsilon^2} \int_{-\epsilon}^\epsilon v(x_1, x_2)(\epsilon - |x_2|)dx_2dx_1 =$$

$$= \int_0^1 \frac{1}{\epsilon^2} \int_{-1}^1 v(x_1, \epsilon z)(\epsilon - \epsilon|z|)\epsilon dzdx_1 = \int_0^1 \int_{-1}^1 v(x_1, \epsilon z)(1 - |z|)dzdx_1.$$

By letting $\epsilon \downarrow 0^+$ we then get

$$\int_0^1 \int_{-1}^1 v(x_1, 0)(1 - |z|)dzdx_1 = \left(\int_\Lambda v(\mathbf{s})d\mathbf{s} \right)\left(\int_{-1}^1 (1 - |z|)dz \right) = \int_\Lambda v(\mathbf{s})d\mathbf{s}.$$

Step 2 Let $\Lambda = L_1 \cup \cdots \cup L_n$ be given by the union of n segments. For any $i = 1, \ldots, n$, let $\hat{L}_i := \{\mathbf{x} \in \Omega \mid \text{dist}(\mathbf{x}, L_i) < \text{dist}(\mathbf{x}, \Lambda \backslash L_i)\}$. We prove the following auxiliary result,

$$|(\Omega \backslash \hat{L}_i) \cap (L_i + B(0, \epsilon))| = o(\epsilon^2).$$

To see this, we note that, upto sets of measure zero,

$$(\Omega \backslash \hat{L}_i) \cap (L_i + B(0, \epsilon)) = (\hat{L}_1 \cup \ldots \hat{L}_{i-1} \cup \hat{L}_{i+1} \cup \ldots \hat{L}_n) \cap (L_i + B(0, \epsilon)).$$

It is then sufficient to prove that $|\hat{L}_j \cap (L_i + B(0, \epsilon))| = o(\epsilon^2)$ for all j independently. If $L_i \cap L_j = \emptyset$, the proof is trivial. Conversely, if the two segments intersect, let θ be the angle between the two lines. It is easy to see that the intersection $\hat{L}_j \cap (L_i + B(0, \epsilon))$ is contained in a triangle of height ϵ and width $\epsilon / \tan(\theta/2) + \epsilon \tan(\theta/2)$. The conclusion follows.

Step 3 Let $\Lambda = L_1 \cup \cdots \cup L_n$ and define the regions $\hat{L}_1, \ldots \hat{L}_n$ as in the previous step. Fix any $v \in \mathcal{C}(\Omega)$. Then

$$\frac{1}{\epsilon^2} \int_\Omega v(\mathbf{x})\varphi_\Lambda^\epsilon(\mathbf{x})d\mathbf{x} = \frac{1}{\epsilon^2} \sum_{i=1}^n \int_{\hat{L}_i} v(\mathbf{x})\varphi_\Lambda^\epsilon(\mathbf{x})d\mathbf{x} = \sum_{i=1}^n \frac{1}{\epsilon^2} \int_{\hat{L}_i} v(\mathbf{x})\varphi_{L_i}^\epsilon(\mathbf{x})d\mathbf{x}.$$

Therefore, we can prove the original claim by showing that $\frac{1}{\epsilon^2} \int_{\hat{L}_i} v(\mathbf{x})\varphi_{L_i}^\epsilon(\mathbf{x})d\mathbf{x} \rightarrow \int_{L_i} v(\mathbf{s})d\mathbf{s}$ for each i. At this purpose, fix any $i = 1, \ldots, n$. We have

$$\left| \frac{1}{\epsilon^2} \int_\Omega v(\mathbf{x})\varphi_{L_i}^\epsilon(\mathbf{x})d\mathbf{x} - \frac{1}{\epsilon^2} \int_{\hat{L}_i} v(\mathbf{x})\varphi_{L_i}^\epsilon(\mathbf{x})d\mathbf{x} \right| \leq \frac{1}{\epsilon^2} \int_{\Omega \backslash \hat{L}_i} |v(\mathbf{x})||\varphi_{L_i}^\epsilon(\mathbf{x})|d\mathbf{x} \leq$$

$$\leq \frac{1}{\epsilon^2}||v||_\infty \cdot \epsilon|(\Omega \backslash \hat{L}_i) \cap (L_i + B(0, \epsilon))| = o(\epsilon)$$

and the conclusion follows from Step 1. □

References

1. Akenine-Moller, T., Haines, E., Hoffman, N.: Real-Time Rendering. AK Peters/CRC Press, Boca Raton (2019)
2. Augasta, M., Kathirvalavakumar, T.: Pruning algorithms of neural networks—a comparative study. Open Comput. Sci. **3**(3), 105–115 (2013)

3. Aurenhammer, F.: Voronoi diagrams-a survey of a fundamental geometric data structure. ACM Comput. Surv. (CSUR) **23**(3), 345–405 (1991)

4. Bhattacharya, K., Hosseini, B., Kovachki, N.B., Stuart, A.M.: Model reduction and neural networks for parametric PDEs. SMAI J. Comput. Math. **7**, 121–157 (2021)

5. Blalock, D., Ortiz, J.J.G., Frankle, J., Guttag, J.: What is the state of neural network pruning?. In: Proceedings of Machine Learning and Systems 2020 (MLSys 2020) (2020)

6. Berner, J., Grohs, P., Kutyniok, G., Petersen, P.: The modern mathematics of deep learning. arXiv preprint arXiv:2105.04026 (2021)

7. Cao, Q., Goswami, S., Karniadakis, G.E.: LNO: Laplace Neural Operator for Solving Differential Equations. arXiv preprint arXiv:2303.10528 (2023)

8. Cattaneo, L., Zunino, P.: A computational model of drug delivery through microcirculation to compare different tumor treatments. Int. J. Numer. Methods Biomed. Eng. **30**(11), 1347–1371 (2014)

9. Chen, G., Liu, X., Li, Y., Meng, Q., Chen, L.: Laplace Neural Operator for Complex Geometries. arXiv preprint arXiv:2302.08166 (2023)

10. Chen, W., Wang, Q., Hesthaven, J.S., Zhang, C.: Physics-informed machine learning for reduced-order modeling of nonlinear problems. J. Comput. Phys. **446**, 110666 (2021)

11. Daubechies, I., DeVore, R., Foucart, S., Hanin, B., Petrova, G.: Nonlinear Approximation and (Deep) ReLU Networks. Construct. Approx. **1**, 1–46 (2021)

12. Elbrächter, D., Grohs, P., Jentzen, A., Schwab, C.: DNN expression rate analysis of high-dimensional PDEs: application to option pricing. Construct. Approx. **1**, 1–69 (2021)

13. Ern, A., Guermond, J.L.: Simplicial Finite Elements. In: Finite Elements I. Texts in Applied Mathematics, Vol. 72. Springer, Cham (2021)

14. Evans, L.C.: Partial Differential Equations, vol. 19. American Mathematical Society, Providence (2022)

15. Feng, Y., Feng, Y., You, H., Zhao, X., Gao, Y.: Meshnet: mesh neural network for 3d shape representation. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, No. 01, pp 8279–8286 (2019)

16. Fisher, R.A.: The wave of advance of advantageous genes. Ann. Eugen. **7**(4), 355–369 (1937)

17. Franco, N.R., Fresca, S., Manzoni, A., Zunino, P.: Approximation bounds for convolutional neural networks in operator learning. Neural Netw. **161**, 129–141 (2023)

18. Franco, N., Manzoni, A., Zunino, P.: A deep learning approach to reduced order modelling of parameter dependent partial differential equations. Math. Comput. **92**(340), 483–524 (2023)

19. Fresca, S., Dede, L., Manzoni, A.: A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. J. Sci. Comput. **87**(2), 1–36 (2021)

20. Fresca, S., Manzoni, A.: POD-DL-ROM: enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition. Comput. Methods Appl. Mech. Eng. **388**, 114181 (2022)

21. Gao, H., Sun, L., Wang, J.X.: PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. J. Comput. Phys. **428**, 110079 (2021)

22. Geist, M., Petersen, P., Raslan, M., Schneider, R., Kutyniok, G.: Numerical solution of the parametric diffusion equation by deep neural networks. J. Sci. Comput. **88**(1), 1–37 (2021)

23. Gladstone, R.J., Rahmani, H., Suryakumar, V., Meidani, H., D'Elia, M., Zareei, A.: GNN-based physics solver for time-independent PDEs. arXiv preprint arXiv:2303.15681 (2023)

24. Goswami, S., Kontolati, K., Shields, M.D., Karniadakis, G.E.: Deep transfer operator learning for partial differential equations under conditional shift. Nat. Mach. Intell. **1**, 1–10 (2022)

25. Gribonval, R., Kutyniok, G., Nielsen, M., Voigtlaender, F.: Approximation spaces of deep neural networks. Constr. Approx. **55**(1), 259–367 (2022)

26. Habbal, A., Barelli, H., Malandain, G.: Assessing the ability of the 2D Fisher-KPP equation to model cell-sheet wound closure. Math. Biosci. **252**, 45–59 (2014)

27. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)

28. Gruber, A., Gunzburger, M., Ju, L., Wang, Z.: A comparison of neural network architectures for data-driven reduced-order modeling. Comput. Methods Appl. Mech. Eng. **393**, 114764 (2022)

29. Gühring, I., Raslan, M.: Approximation rates for neural networks with encodable weights in smoothness spaces. Neural Netw. **134**, 107–130 (2021)

30. Guo, M., Manzoni, A., Amendt, M., Conti, P., Hesthaven, J.S.: Multi-fidelity regression using artificial neural networks: efficient approximation of parameter-dependent output quantities. Comput. Methods Appl. Mech. Eng. **389**, 114378 (2022)

31. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, pp. 1026–1034 (2015)

32. Kovachki, N., Lanthaler, S., Mishra, S.: On universal approximation and error bounds for Fourier neural operators. J. Mach. Learn. Res. **22**(1), 13237–13312 (2021)
33. Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., Anandkumar, A.: Neural operator: learning maps between function spaces. arXiv preprint arXiv:2108.08481 (2021)
34. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. NIPS (2012)
35. Kuramoto, Y.: Diffusion-induced chaos in reaction systems. Prog. Theor. Phys. Suppl. **64**, 346–367 (1978)
36. Kutyniok, G., Petersen, P., Raslan, M., Schneider, R.: A theoretical analysis of deep neural networks and parametric PDEs. Construct. Approx. **1**, 1–53 (2021)
37. Lanthaler, S., Mishra, S., Karniadakis, G.E.: Error estimates for deeponets: A deep learning framework in infinite dimensions. Trans. Math. Appl. **6**(1), tnac001 (2022)
38. Lee, K., Carlberg, K.T.: Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. J. Comput. Phys. **404**, 108973 (2020)
39. Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A.: Fourier Neural Operator for Parametric Partial Differential Equations. arXiv preprint arXiv:2010.08895 (2020)
40. Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.E.: Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nat. Mach. Intell. **3**(3), 218–229 (2021)
41. Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., Karniadakis, G.E.: A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. Comput. Methods Appl. Mech. Eng. **393**, 114778 (2022)
42. Park, Y.J.: On the Continuity of the Hardy–Littlewood Maximal Function. J. Chungcheong Math. Soc. **31**(1), 43–46 (2018)
43. Perera, S., Barnes, N., He, X., Izadi, S., Kohli, P., Glocker, B.: Motion segmentation of truncated signed distance function based volumetric surfaces. In: 2015 IEEE Winter Conference on Applications of Computer Vision, pp. 1046–1053. IEEE (2015)
44. Possenti, L., Cicchetti, A., Rosati, R., Cerroni, D., Costantino, M.L., Rancati, T., Zunino, P.: A mesoscale computational model for microvascular oxygen transfer. Ann. Biomed. Eng. **49**(12), 3356–3373 (2021)
45. Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., Battaglia, P.W.: Learning Mesh-Based Simulation with Graph Networks. arXiv preprint arXiv:2010.03409 (2020)
46. Ravasio, A., Cheddadi, I., Chen, T., Pereira, T., Ong, H.T., Bertocchi, C., et al.: Gap geometry dictates epithelial closure efficiency. Nat. Commun. **6**(1), 7683 (2015)
47. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Networks **20**(1), 61–80 (2008)
48. Sherratt, J. A., Murray, J.D.: Models of epidermal wound healing. Proc. R. Soc. Lond. Ser. B: Biol. Sci. **241**(1300), 29–36 (1990)
49. Wang, S., Wang, H., Perdikaris, P.: Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. Sci. Adv. **7**(40), eabi8605 (2021)
50. Wu, Y., Schuster, M., Chen, Z., Le, Q., Norouzi, M., Macherey, W. et al.: Google's neural machine translation system: bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144 (2016)
51. Xu, J., Pradhan, A., Duraisamy, K.: Conditionally parameterized, discretization-aware neural networks for mesh-based modeling of physical systems. Adv. Neural. Inf. Process. Syst. **34**, 1634–1645 (2021)
52. Quarteroni, A., Manzoni, A., Negri, F.: Reduced Basis Methods for Partial Differential Equations: An Introduction, vol. 92. Springer, Berlin (2015)
53. Quarteroni, A., Valli, A.: Numerical Approximation of Partial Differential Equations, vol. 23. Springer, Berlin (2008)

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