Finite Element Operator Learning for Solving Parametric PDEs without Labeled Data

Anonymous authors
Paper under double-blind review

Abstract

Partial differential equations (PDEs) underlie our understanding and prediction of natural phenomena across numerous fields, including physics, engineering, and finance. However, solving parametric PDEs is a complex task that necessitates efficient numerical methods. In this paper, we propose a novel approach for solving parametric PDEs using a Finite Element Operator Network (FEONet). Our proposed method leverages the power of deep learning in conjunction with traditional numerical methods, specifically the finite element method, to solve parametric PDEs in the absence of any paired input-output training data. We demonstrate the effectiveness of our approach on several benchmark problems and show that it outperforms existing state-of-the-art methods in terms of accuracy, generalization, and computational flexibility. Our FEONet framework shows potential for application in various fields where PDEs play a crucial role in modeling complex domains with diverse boundary conditions and singular behavior. Furthermore, we provide theoretical convergence analysis to support our approach, utilizing finite element approximation in numerical analysis.

1 Introduction

Solving partial differential equations (PDEs) is vital as they serve as the foundation for understanding and predicting the behavior of a range of natural phenomena Courant & Hilbert (1953); Bender (2000). From fluid dynamics, heat transfer, to electromagnetic fields, PDEs provide a mathematical framework that allows us to comprehend and model these complex systems Gershenfeld & Gershenfeld (1999); Lin & Segel (1988). Their significance extends beyond the realm of physics and finds applications in fields such as finance, economics, and computer graphics Simon et al. (1994); Mortenson (1999). In essence, PDEs are fundamental in advancing our understanding of the world around us and have a broad impact across various industries and fields of study.

Numerical methods are essential for approximating solutions to PDEs when exact solutions are unattainable. This is especially significant for complex systems that defy solutions using traditional methods Burden et al. (2015); Atkinson (1991). Various techniques, such as finite difference, finite element, and finite volume methods, are utilized to develop these numerical methods (Larson & Bengzon, 2013; Strikwerda, 2004; LeVeque, 2002). In particular, the finite element method (FEM) is widely employed in engineering and physics, where the domain is divided into small elements and the solution within each element is approximated using polynomial functions. The mathematical analysis of the FEM has undergone significant development, resulting in a remarkable enhancement in the method's reliability. The FEM is particularly advantageous in handling irregular geometries and complex boundary conditions, making it a valuable tool for analyzing structures, heat transfer, fluid dynamics, and electromagnetic problems (Zienkiewicz & Taylor, 2000; Hughes, 2000). However, it is important to note that the implementation of numerical methods often comes with a significant computational cost.

The use of machine learning in solving PDEs has gained significant traction in recent years. By integrating deep neural networks and statistical learning theory into numerical PDEs, a new field known as scientific machine learning has emerged, presenting novel research opportunities. The application of machine learning to PDEs can be traced back to the 1990s when neural networks were employed to approximate solutions (Lagaris et al., 1998). More recently, the field of physics-informed neural networks (PINNs) has been developed, where neural networks are trained to learn the underlying physics of a system, enabling us to solve PDEs and other physics-based problems (Lu et al., 2021b; Karniadakis et al., 2021; Yang et al., 2021; Meng et al., 2020; Raissi et al., 2019). Despite their advantages, these approaches come with limitations. One prominent drawback of PINN methods is that they are trained on a single instance of input data, encompassing initial conditions, boundary conditions, and external force terms. As a consequence, if the input data changes, the entire training process must be repeated, posing challenges in making real-time predictions for varying input data. This limitation restricts the applicability of PINNs in dynamic and adaptive systems where the input data is subject to change.

Operator learning, a recent research area, tackles the challenge by using data-driven methods to learn mathematical operators governing a physical system for solving parametric PDEs (Lu et al., 2021a; Li et al., 2021a; Wang et al., 2021; Brandstetter et al., 2022; Lienen & Günnemann, 2022; Lu et al., 2022; Fanaskov & Oseledets, 2022; Lee et al., 2023). These methods allow us to make a fast prediction of solution in real-time whenever the given PDE data changes, which offers the completely new framework. However, these methods rely on training data for supervised learning that consists of pre-computed (either analytically or numerically) pairs of PDE parameters (e.g., external force, boundary condition, coefficients, initial condition) and their corresponding solutions. In general, generating a reliable training dataset requires extensive numerical computations, which can be computationally inefficient and time-consuming. Obtaining a sufficiently large dataset is also challenging, especially for systems with complex geometries or nonlinear equations. Additionally, using neural networks as the solution space poses challenges in imposing boundary values and can lead to less accurate solutions. Researchers are actively working on developing methods to enhance the efficiency and accuracy of these data-driven approaches Goswami et al. (2022a).

To address the aforementioned limitations, this paper proposes a novel approach for solving diverse parametric PDEs. Instead of requiring a training dataset of solutions, we introduce a new method to learn the coefficients. Specifically, we present the Finite Element Operator Network (FEONet) that utilizes the finite element approximation to learn the solution operator, eliminating the need for solution datasets to address various parametric PDEs. In the FEM framework, the numerical solution is approximated by the linear combination of nodal coefficients, α_k, and the nodal basis, $\phi_k(\mathbf{x})$. The nodal basis in the FEM consists of piecewise polynomials defined by the finite set of nodes of a mesh, which exhibit both near orthogonality and local support such that

$$u_h(\mathbf{x}) = \sum \alpha_k \phi_k(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d, \quad (1)$$

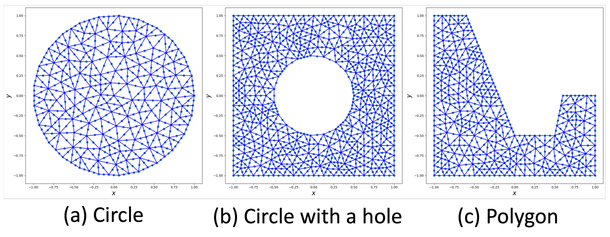


Figure 1: Domain triangulation for three different complex geometries.

where u_h is the FEM solution for the given PDEs. Motivated by (1), the FEONet is able to predict numerical solutions to the PDEs when given the initial conditions, external forcing functions, or PDE coefficients as inputs. As the neural network only predicts the nodal coefficients, this approach eliminates the need for extensive numerical computations to generate a large dataset and also allows for real-time predictions for varying input data. Therefore, the FEONet can learn multiple instances of the solutions of PDEs, making it a versatile approach for solving different types of PDEs in complex domains; see e.g. Figure 1. The loss function of the FEONet is designed based on the residual quantity of the finite element approximation, similar to the FEM Ciarlet (2002); Brenner & Scott (2008). This allows the FEONet to accurately approximate the solutions of PDEs, while also ensuring that the boundary conditions are exactly satisfied. To construct an approximation for the solution of PDEs, the FEONet borrows a framework from the FEM. This approach involves inferring coefficients, denoted as $\widehat{\alpha}_k$, which are then used to construct the linear combination $\sum \widehat{\alpha}_k \phi_k$ to approximate the solution of PDEs. Since each basis function satisfies the exact boundary condition, the predicted solution will also satisfy the exact boundary condition. Additionally, since the high-order FEM is available to generate our framework in (1), the predicted numerical solution is expected to yield relatively small errors compared to other machine learning-based approaches. It is noteworthy that due to the intrinsic structure of the proposed FEONet scheme, we do not need any paired input-output training data (as we adopted the weak residual as a loss) and hence the model can be trained in an unsupervised manner.

Another key contribution of this study is to introduce a novel learning architecture designed to accurately solve convection-dominated singularly perturbed problems involving strong boundary layer phenomena. These types of problems pose significant challenges for traditional numerical methods due to sharp transitions inside thin layers caused by a small diffusive parameter. By adapting theory-guided methods Chekroun et al. (2020), the network effectively captures the behavior of boundary layers, an area that has been underexplored in recent machine-learning approaches.

The main contributions of the paper are summarized as follows:

- The FEONet, as shown in Figure 3, enables the learning of solutions for multiple instances of parametric PDEs without requiring the input-output training dataset. This is the significant challenge that many existing methods face.
- Compared to existing methods similar to our approach, the FEONet offers significantly greater computational flexibility. For instance, it can solve PDEs on domains with complex geometry, as depicted in Figure 4, across diverse settings. Moreover, any form of PDE data (forcing, coefficients, boundary conditions, initial conditions) can be used as input to the operator network. Additionally, our method uniquely enforces exact boundary conditions, both Dirichlet and Neumann. This is a challenge to accomplish with other neural network-based methods, as shown in Table 1.
- Since our method is rooted in numerical analysis, we can apply established techniques from numerical methods. For example, complex scenarios, such as the challenging boundary layer problem significant in engineering contexts, can be effectively addressed using enriched basis functions (Figure 5).
- The FEONet is supported by rigorous theoretical foundations through finite element approximation on P1 and P2 elements, as well as novel convergence analysis for approximation and generalization errors (Theorem 3.1). Moreover, it is anticipated to exhibit low generalization errors.

2 Related works

Physics-informed neural network (PINN). The PINNs, developed by Raissi et al. (2019), utilize a neural network to efficiently solve PDEs with little or no data. The key feature of PINN is its ability to minimize PDE residual loss by enforcing physical constraints, with output fields automatically differentiated with respect to input coordinates. PINNs have been successfully applied in various fields such as material science and biophysics Goswami et al. (2022b); Alber et al. (2019). However, one of the key limitations of the PINN method is that it is trained on a single set of input data, which includes initial conditions, boundary conditions, and external force terms.

Operator learning. Operator learning utilizes a dataset of input-output pairs obtained from an existing solver to train a model to learn the solution operator Bhatnagar et al. (2019); Guo et al. (2016); Khoo et al. (2017); Zhu & Zabaras (2018). Operator learning aims to improve the efficiency and accuracy of solving PDEs by learning the underlying patterns and relationships in the input-output pairs, allowing for real-time predictions for varying input data Hwang et al. (2022). Kovachki et al. (2021) proposed an iterative neural operator structure to learn the solution operator of PDEs. Based on the universal approximation theorem for the operator, Lu et al. (2021a) developed a Deep operator network (DeepONet) architecture. In recent works Brandstetter et al. (2022); Lienen & Günnemann (2022); Pfaff et al. (2021); Li et al. (2020); Boussif et al. (2022), a message-passing neural PDE solver capable of handling diverse problem properties has been developed. However, this approach faces challenges such as the requirement for large amounts of training data, limited generalization ability, and extrapolation to unseen conditions. Researchers are currently addressing these challenges by incorporating physics or PDE constraints and designing more expressive models.

Physics-informed operator learning. The integration of PINNs and operator learning has led to the development of innovative methods Goswami et al. (2022a); Zhu et al. (2019) such as the Physics-informed neural operator (PINO) Li et al. (2021b) and Physics-informed DeepONet (PIDeepONet) Wang et al. (2021). These methods aim to capitalize on the strengths of both PINNs and operator learning by embedding physical equations in the loss function of the neural operator. However, there are certain limitations to these methods, such as the inability to effectively address the boundary layer issue, which plays a crucial role in real-world problems. Additionally, PIDeepONet has been observed to have relatively high generalization errors when no training data is available. Furthermore, PINO has mainly been applied to problems with a square-shaped domain.

3 Method

In this section, we present our proposed method, the FEONet, for solving parametric PDEs. We begin by providing a concise overview of the FEMs, which serve as the foundation of our approach. Next, we introduce the FEONet by demonstrating how neural network techniques can be seamlessly integrated into FEMs to solve the parametric PDEs. Throughout our method description and experiments, we focus on the PDEs of the following form: for uniformly elliptic coefficients $\boldsymbol{a}(\mathbf{x})$ and $\varepsilon > 0$,

$$-\varepsilon \operatorname{div}(\boldsymbol{a}(\mathbf{x})\nabla u) + \mathcal{F}(u) = f \quad \text{in } D. \tag{2}$$

Here $\mathcal{F}$ can be either linear or nonlinear, and both cases will be covered in the experiments performed in Section 4. As will be described in more detail later, we shall propose an operator-learning-type method that can provide real-time solution predictions whenever the input data of the PDE changes.

3.1 Finite element method (FEM)

The FEM is a general class of techniques for the numerical solution of PDEs. FEM is based on the variational formulation of the given PDE (2): find $u \in V$ satisfying

$$B[u,v] := \varepsilon \int_D \boldsymbol{a}(\mathbf{x})\nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x} + \int_D \mathcal{F}(u) v \, \mathrm{d}\mathbf{x} = \int_D f v \, \mathrm{d}\mathbf{x} =: \ell(v) \quad \text{for all } v \in V, \tag{3}$$

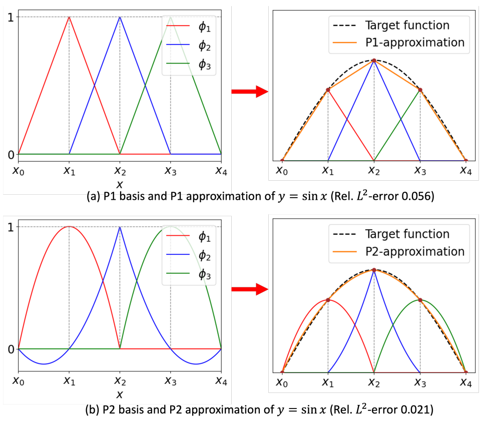
where the solution and test function space V is usually chosen to be an infinite-dimensional function space. In the theory of FEM, the first step is to define a *triangulation* of the given domain $D \subset \mathbb{R}^d$. If $d = 1$ and $D = [a,b]$ for some $a,\, b \in \mathbb{R}$, we set $a = x_0 < x_1 < x_2 < \cdots < x_K = b$ and each interval (1-simplex) $[x_{i-1}, x_i]$ defines an element. For $d = 2$, the triangulation is referred to as a tessellation of D into a finite number of closed triangles (2-simplex) T_i, $i = 1, \cdots, K$, whose interiors are pairwise disjoint, and for $i \neq j$ with $T_i \cap T_j$ is nonempty, $T_i \cap T_j$ is either a common vertex or a common edge of T_i and T_j. In this case, each triangle defines an element and the vertices $\{\mathbf{x}_i\}$ of triangles are called *nodes* (Figure 1). The triangulation in higher dimension $d \geq 3$ can be defined in a similar manner using d-simplex as an element. Letting h_T be the largest edge of a triangle T, we define a finite element parameter $h > 0$ as the longest among the h_T. In addition, let us denote by S_h, the space of all continuous functions v_h defined on D such that the restriction of v_h to an arbitrary triangle is a polynomial. Then we define our finite-dimensional ansatz space as $V_h = S_h \cap V$. We shall denote the set of all vertices in the triangulation by $\{\mathbf{x}_i\}$, and define the so-called *nodal basis* $\{\phi_j\}$ for V_h, defined by $\phi_j(\mathbf{x}_i) = \delta_{ij}$. If we use piecewise linear nodal basis functions, we call it the P1-element method, while the adoption of piecewise quadratic polynomials is called as the P2-element method (Figure 2). Note that the dimension of V_h depends on the triangulation of D, and hence, depends also on the finite element parameter $h > 0$.

The idea of the FEM is to approximate V with a finite-dimensional subspace V_h above, generated by the basis functions $\{\phi_1, \phi_2, \cdots, \phi_{N(h)}\}$, so that the given problem becomes numerically feasible. More precisely, we aim to compute the approximate solution $u_h \in V_h$ based on the, so-called, *Galerkin approximation*

$$B[u_h, v_h] = \ell(v_h) \quad \text{for all } v_h \in V_h. \tag{4}$$

If we write the finite element solution

$$u_h = \sum_{k=1}^{N(h)} \alpha_k \phi_k, \quad \alpha_i \in \mathbb{R}, \tag{5}$$



(a) P1 basis and P1 approximation of $y = \sin x$ (Rel. L^2-error 0.056)

(b) P2 basis and P2 approximation of $y = \sin x$ (Rel. L^2-error 0.021)

Figure 2: Shape of nodal basis functions and approximation of $y = \sin x$ for (a) P1-element and (b) P2-element.

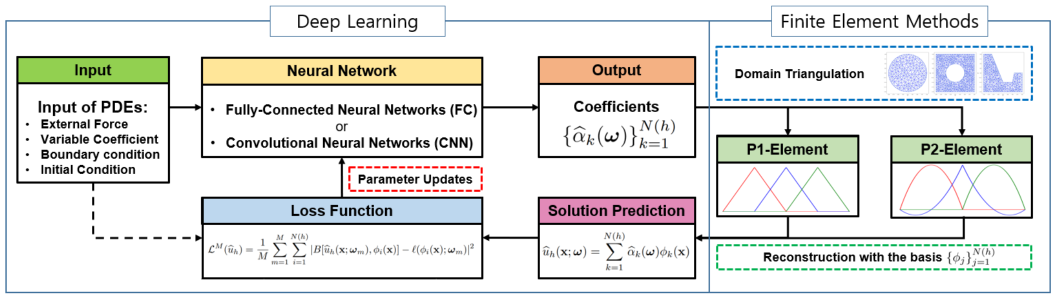

Deep Learning
Finite Element Methods
Input
Input of PDEs:
External Force
Variable Coefficient
Boundary condition
Initial Condition
Neural Network
Fully-Connected Neural Networks (FC)
or
Convolutional Neural Networks (CNN)
Output
Coefficients
$\{\widehat{\alpha}_k(\boldsymbol{\omega})\}_{k=1}^{N(h)}$
Domain Triangulation
P1-Element
P2-Element
Parameter Updates
Loss Function
$\mathcal{L}^M(\widehat{u}_h) = \frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{N(h)} |B[\widehat{u}_h(\mathbf{x};\boldsymbol{\omega}_m), \phi_i(\mathbf{x})] - \ell(\phi_i(\mathbf{x});\boldsymbol{\omega}_m)|^2$
Solution Prediction
$\widehat{u}_h(\mathbf{x};\boldsymbol{\omega}) = \sum_{k=1}^{N(h)} \widehat{\alpha}_k(\boldsymbol{\omega})\phi_k(\mathbf{x})$
Reconstruction with the basis $\{\phi_j\}_{j=1}^{N(h)}$

Figure 3: Schematic diagram of the Finite Element Operator Network (FEONet).

the Galerkin approximation (4) is transformed into the linear algebraic system

$$A\alpha = F \quad \text{with } A_{ik} := B[\phi_k, \phi_i] \text{ and } F_i := \ell(\phi_i), \tag{6}$$

where the matrix $A \in \mathbb{R}^{N(h)\times N(h)}$ is invertible provided that the given PDEs have some suitable structure. Therefore, we can determine the coefficients $\{\alpha_k\}_{k=1}^{N(h)}$ by solving the system of linear equations (6), and consequently, produce the approximate solution $u_h \in V_h$ as defined in (5).

Once we compute the approximate solution u_h of u, we can measure how close $u_h \in V_h$ is to $u \in V$. If the given PDE has some good structure, the weak solution u is sufficiently smooth, and the triangulation is reasonable, we can derive the following error estimates: for some constants C_1, $C_2 > 0$ independent of $h > 0$, we have

$$\begin{aligned} &\textbf{(P1-element method)} \qquad \|u - u_h\|_{L^2(D)} \le C_1 h^2, \\ &\textbf{(P2-element method)} \qquad \|u - u_h\|_{L^2(D)} \le C_2 h^3. \end{aligned} \tag{7}$$

For a detailed discussion of the error analysis, see, for example, Theorem 5.4.8 of Brenner & Scott (2008). From the estimates (7), we can confirm that the approximate solution u_h computed by the FEM converges to the true solution u as $h \to 0$ and the convergence rate for the P1 method is 2, while the P2-method achieves the higher convergence rate 3.

3.2 Finite Element Operator Network (FEONet)

The FEONet is based on the FEM described in the previous section. In our scheme, an input for the neural network is the data for the given PDEs. In this paper, we choose external force as a prototype input feature. Note, however, that the same scheme can also be developed with other types of data, e.g., boundary conditions, diffusion coefficients or initial conditions for time-dependent problems. See, for example, Appendix D.4 where extensive experiments were performed addressing the cases of another type of input data. For this purpose, the external forcing terms are parametrized by the random parameter $\boldsymbol{\omega}$ contained in the (possibly high-dimensional) parameter space Ω. For each realization $f(\mathbf{x};\boldsymbol{\omega})$ (and hence for each load vector $F(\boldsymbol{\omega})$ defined in (6)), instead of computing the coefficients from the linear algebraic system (6), we approximate the coefficients α by a deep neural network. More precisely, these input features representing external forces pass through the deep neural network and the coefficients $\{\widehat{\alpha}_k\}$ are generated as an output of the neural network. We then reconstruct the solution

$$\widehat{u}_h(\mathbf{x};\boldsymbol{\omega}) = \sum_{k=1}^{N(h)} \widehat{\alpha}_k(\boldsymbol{\omega})\phi_k(\mathbf{x}). \tag{8}$$

For the training, we use a residual of the variational formulation (4) and define the loss function by summing over all basis functions: for randomly chosen parameters $\boldsymbol{\omega}_1, \cdots, \boldsymbol{\omega}_M \in \Omega$,

$$\mathcal{L}^M(\widehat{u}_h) = \frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{N(h)} |B[\widehat{u}_h(\mathbf{x};\boldsymbol{\omega}_m), \phi_i(\mathbf{x})] - \ell(\phi_i(\mathbf{x});\boldsymbol{\omega}_m)|^2. \tag{9}$$

For each training epoch, once the neural network parameters are updated in the direction of minimizing the loss function (9), the external force $f(\mathbf{x};\boldsymbol{\omega})$ passes through this updated neural network to generate more refined coefficients, and this procedure is repeated until the sufficiently small loss is achieved. A schematic diagram of the FEONet algorithm is depicted in Figure 3.

3.3 Convergence of FEONet

We address a convergence result of the FEONet which provides theoretical justification for the proposed method. As outlined in the previous section, the proposed method is based on the FEM and the finite element approximation u_h in (5) can be considered as an intermediate solution between the true solution u of (3) and the approximate solution $\widehat{u}_h$ predicted by the FEONet. Consequently, in order for error analysis, we simply split the error $u - \widehat{u}_h$ into two parts; namely,

$$u - \widehat{u}_h = (u - u_h) + (u_h - \widehat{u}_h) =: \text{(I)} + \text{(II)}. \tag{10}$$

The first error (I) can be controlled via the classical theory of FEM, e.g. through the error estimate (7). The second error (II) is a core part which needs a novel theoretical investigation. The parameters of $\widehat{u}_h =: \widehat{u}_{h,n,M}$ concerning the convergence is the parameter representing neural network architecture and the number of sampling points $\boldsymbol{\omega} \in \Omega$ which we denote by $n \in \mathbb{N}$ and $M \in \mathbb{N}$ respectively. Here, larger $n \in \mathbb{N}$ implies more approximation power of neural networks. We will prove the following theorem concerning the second error (II), which guarantees that our method is reliable and the theorem forms the theoretical foundation of the proposed numerical scheme. The precise mathematical statement and proof can be found in Theorem B.7.

Theorem 3.1. *If we let u_h be the finite element approximation of the true solution u and $\widehat{u}_{h,n,M}$ be the approximate solution computed by the FEONet, then we have*

$$\mathbb{E}_{\boldsymbol{\omega}}\left[\|u_h - \widehat{u}_{h,n,M}\|^2_{L^2(D)}\right] \to 0 \quad \text{as } n,\, M \to \infty. \tag{11}$$

It is noteworthy that $\widehat{u}_h$ shares some interesting properties with u_h. As it will be made clear in the next section, we may conclude from experiments that $\|\text{(I)}\| \gg \|\text{(II)}\|$, and the total error $\|u - \widehat{u}_h\|$ is sufficiently close to the finite element error $\|\text{(I)}\|$. Therefore, as we can see from Figure 6, for large enough $n,\, M \in \mathbb{N}$, $\|u - \widehat{u}_h\|$ has the convergence rates close to 2 and 3 (for P1 and P2 respectively) with respect to $h > 0$, which agrees well with the theoretical result (7) for $\|\text{(I)}\|$.

4 Numerical experiments

In this section, we demonstrate the experimental results of the FEONet across various settings for PDE problems. We randomly generate 3000 input samples for external forces and train the FEONet in an unsupervised manner without the usage of pre-computed input-output (f, u) pairs. We then evaluate the performance of our model with another randomly generated test set consisting of 3000 pairs of f and the corresponding solutions u. For this test data, we employed the FEM with a sufficiently fine mesh discretization to ensure that the numerical solutions can be considered as true solutions. The forcing term f is randomly generated. While there are other operator-learning-based methods, such as PINO (Li et al., 2021b), that can be trained without input-output data pairs, they often face limitations when dealing with domains of complex shapes. A detailed comparison with the results for PINO can be found in Appendix C.3. Consequently, our primary comparison will be against DeepONet (Lu et al., 2021a), taking into account varying numbers of training data, and PIDeepONet (Wang et al., 2021). The triangulation of domains and the generation of the nodal basis for the FEM are performed using FEniCS (Alnaes et al., 2015; Logg et al., 2012). A detailed description of the model's hyperparameters and experimental setup can be found in Appendix D.

4.1 Comparison in various PDE problems

To evaluate the computational flexibility of the FEONet, we conducted a sequence of experiments covering diverse domains, boundary conditions, and equations.

Various domains. We consider the 2D convection-diffusion equation as

$$\begin{aligned} -\varepsilon \Delta u + \mathbf{v} \cdot \nabla u &= f(x,y), \quad && (x,y) \in D, \\ u(x,y) &= 0, && (x,y) \in \partial D. \end{aligned} \tag{12}$$

For our numerical experiments, we fixed the values of $\varepsilon = 0.1$ and $\mathbf{v} = (-1, 0)$ across various domains D, which included a circle, a square with a hole, and a polygon (see Figure 1). The second to fourth columns of Table 1 display the mean relative L^2 errors of the test set for FEONet (w/o labeled

Table 1: Mean Rel. L^2 test errors ($\times\mathbf{10^{-2}}$) with standard deviations ($\times\mathbf{10^{-2}}$) for the 3000 test set on diverse PDE problems. Five training trials are performed independently. Domain I (circle), Domain II (square with a hole), Domain III (polygon) with Eq. (12), BC I (Dirichlet), BC II (Neumann) with Eq. (13), Eq I (second-order linear equation (14)) and Eq II (Burgers' equation (15)).

Model (#Train data)	Domain I	Domain II	Domain III	BC I	BC II	Eq I	Eq II
DON (supervised, w/30)	27.15$\pm$1.16	51.21$\pm$3.58	53.92$\pm$4.59	21.75$\pm$1.19	22.75$\pm$1.05	24.38$\pm$1.37	10.26$\pm$0.14
DON (supervised, w/300)	2.10$\pm$0.75	5.62$\pm$0.37	6.22$\pm$0.96	0.68$\pm$0.11	0.96$\pm$0.06	0.76$\pm$0.10	**0.20**$\pm$0.09
DON (supervised, w/3000)	**0.69**$\pm$0.17	4.75$\pm$0.75	6.20$\pm$1.00	0.53$\pm$0.36	0.33$\pm$0.09	0.33$\pm$0.27	0.24$\pm$0.13
PIDeepONet (w/o labeled data)	9.80$\pm$9.41	101.03$\pm$167.46	32.89$\pm$6.34	1.51$\pm$0.46	1.43$\pm$0.45	19.41$\pm$11.30	2.66$\pm$0.71
Ours (w/o labeled data)	1.24$\pm$0.00	**1.76**$\pm$0.03	**0.51**$\pm$0.00	**0.13**$\pm$0.01	**0.32**$\pm$0.03	**0.13**$\pm$0.01	0.54$\pm$0.07

data), PIDeepONet (w/o labeled data), and DeepONet (with 30/300/3000 input-output data pairs) across various domains. When the DeepONet is trained with either 30 or 300 data pairs, FEONet consistently surpasses them in terms of numerical errors. Notably, even when DeepONet is trained with 3000 data pairs, FEONet achieves errors that are either comparable to or lower than those of DeepONet. It's noteworthy that in more intricate domains like Domain II and Domain III, both the DeepONet (even with 3000 data pairs) and PIDeepONet find it challenging to produce accurate predictions. Figure 4 demonstrates the solution profile, emphasizing that FEONet is more precise in predicting the solution than both DeepONet and PIDeepONet, particularly in complex domains.

Various boundary conditions. We also tested the performance of each model, when the given PDEs were equipped with either Dirichlet or Neumann boundary conditions:

$$\begin{aligned} -\varepsilon u_{xx} + b u_x &= f(x), \qquad x \in D, \\ u(x) = 0 \quad \text{or} \quad u_x(x) &= 0, \quad\ x \in \partial D, \end{aligned} \tag{13}$$

where $\varepsilon = 0.1$ and $b = -1$. For the Neumann boundary condition, we add u to the left-hand side of (13) to guarantee the uniqueness of the solution up to constant. The FEONet has an additional significant advantage to make the predicted solutions satisfy the exact boundary condition, by selecting the appropriate coefficients for the basis functions. Using the FEONet without any input-output training data, we are able to obtain similar or slightly higher accuracy compared to the DeepONet with 3000 training data, for both homogeneous Dirichlet or Neumann boundary conditions as shown in Table 1.

Various equations. We performed some additional experiments applying the FEONet to the various equations: (i) general second-order linear PDE with variable coefficients defined as

$$\begin{aligned} -\varepsilon u_{xx} + b(x) u_x + c(x) u &= f(x), x \in D, \\ u(x) &= 0, \quad x \in \partial D, \end{aligned} \tag{14}$$

where $\varepsilon = 0.1$, $b(x) = x^2 + 1$ and $c(x) = x$; and (ii) the Burgers' equation defined as

$$\begin{aligned} -u_{xx} + u u_x &= f(x), \quad x \in D, \\ u(x) &= 0, \qquad x \in \partial D. \end{aligned} \tag{15}$$

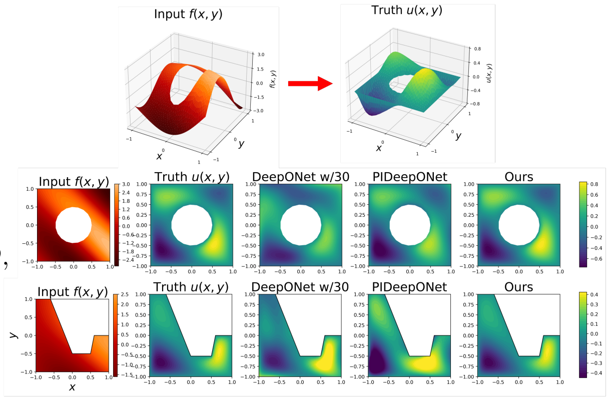


Figure 4: Solution profiles for complex geometries.

Since the Burgers' equation is nonlinear, an iterative method is required for classical numerical schemes including the FEM, which usually causes some computational costs. Furthermore, it is more difficult than linear equations to obtain training data for nonlinear equations. One important advantage of the FEONet is its ability to effectively learn the nonlinear structure without any training data. Once the training process is completed, the model can provide accurate real-time solution predictions without relying on iterative schemes. See Appendix D.6 for a detailed explanation of the loss function for the nonlinear Burgers' equation. The 7th and 8th columns of Table 1 highlight the effectiveness of the FEONet, predicting the solutions with reasonably low errors for various equations. Although the 8th column of Table 1 shows that the DeepONet with 3000 training data has a lower error compared to our model, it demonstrates the strength of the FEONet that achieves a similar level of accuracy even for the nonlinear PDE, that can be trained in an unsupervised manner.

Table 2: Comparisons of various deep-learning-based models for solving the singular perturbation problem. Each model is trained five times independently.

Model	Ours	PINN	DeepONet w/30	PIDeepONet
Requirement for labeled data	**No**	**No**	Yes	**No**
Multiple instance	**Yes**	No	**Yes**	**Yes**
Mean Rel. L^2 test error$\pm$std	**0.0132**$\pm$0.0091	1.3827$\pm$0.7580	0.2303$\pm$0.0074	0.5713$\pm$0.0007

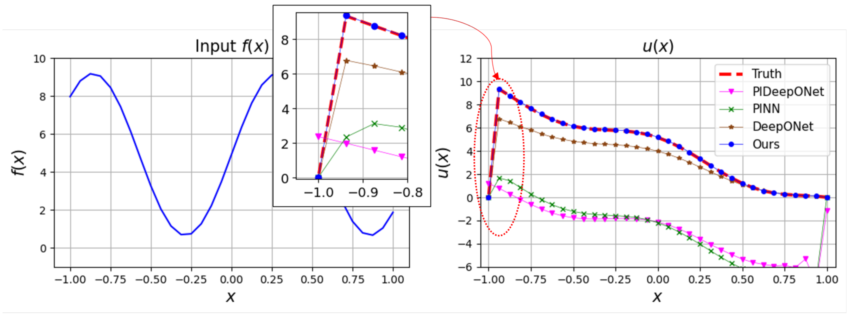


Figure 5: Input function f and the corresponding approximate solution u obtained by various deep-learning-based methods for the singular perturbation problem with $\varepsilon = 10^{-5}$.

Various input functions. FEONet can accommodate various input functions, including diffusion coefficients (Figure 10), boundary conditions (Figure 11), and initial conditions for time-dependent problems (Figure 12). This demonstrates FEONet's flexibility in adapting to a broad spectrum of input function types, underscoring its versatility. For more details, see Appendix D.4.

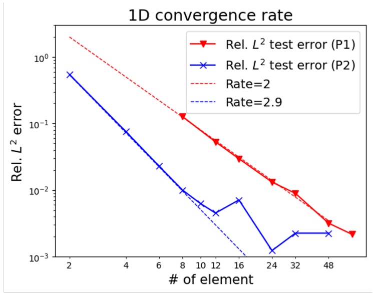
4.2 Singular perturbation problem

One of the notable advantages of the FEONet, which predicts coefficients through well-defined basis functions, is its applicability to problems involving singularly perturbed PDEs. This is accomplished by incorporating additional basis functions guided by theoretical considerations. For this we consider

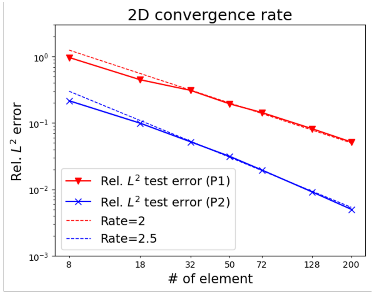
$$\begin{aligned} -\varepsilon u_{xx} + b u_x &= f(x), \quad x \in D, \\ u(x) &= 0, \qquad x \in \partial D, \end{aligned} \tag{16}$$

where $\varepsilon \ll 1$. For the implementation, we assign the values of $b = -1$ and $\varepsilon = 10^{-5}$. For instance, in the right panel of Figure 5, the red dotted line (ground-truth) shows the solution profile which produces a sharp transition near $x = -1$. To capture the boundary layer, we utilize an additional basis function known as the corrector function in mathematical analysis, defined as

$$\phi_{\text{cor}}(x) := e^{-(1+x)/\varepsilon} - (1 - (1 - e^{-2/\varepsilon})(x + 1)/2), \tag{17}$$



(a) 1D convergence plot (log-log scale)



(b) 2D convergence plot (log-log scale)

Figure 6: Rel. L^2 error of the FEONet with respect to the number of elements.

see Appendix D.7 for more detailed information regarding the derivation of the additional basis function. By integrating the boundary layer element into the finite element space, we establish the proposed enriched Galerkin space for utilization in the FEONet. As the corrector basis is incorporated alongside the conventional nodal basis functions in the FEM, we also predict the additional coefficient originating from the corrector basis. Note that the computational overhead remains minimal as the enriched basis exclusively encompasses boundary elements. Table 2 summarizes the characteristics of various operator learning models and the corresponding errors when solving the singularly perturbed PDE described in (16). The performance of the FEONet in accurately solving boundary layer

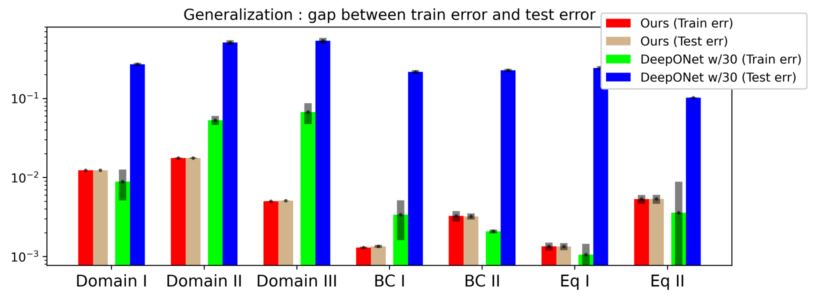


Figure 7: Rel. L^2 train and test error comparison between the FEONet and the DeepONet with 30 training data. Results are averaged over five independent training trials, and black bars indicate the standard deviation.

problems surpasses that of other models, even without labeled data, across multiple instances. The incorporation of a very small ε in the residual loss of the PDE makes operator learning challenging or unstable when using PINN and PIDeepONet. In fact, neural networks inherently possess a smooth prior, which can lead to difficulties in handling boundary layer problems. In contrast, FEONet utilizes theory-guided basis functions, enabling the predicted solution to accurately capture the sharp transition near the boundary layer. The PIDeepONet faces difficulties in effectively learning both the residual loss and boundary conditions, while PINNs struggle to predict the solution (refer to Figure 14). Figure 5 exhibits the results of operator learning for the singular perturbation problem. As depicted in the zoomed-in graph of Figure 5, the FEONet stands out as the only model capable of accurately capturing sharp transitions, whereas other models exhibit noticeable errors.

4.3 Further experiments

Convergence rate. One of the intriguing aspects is that the theoretical results on the convergence error rate of the FEONet (as presented in (7)) are also observable in the experimental results. Figure 6 shows the relationship between the test error and the number of elements, utilizing both P1 and P2 basis functions. In 1D problems, the convergence rate is approximately 2 for P1 basis functions and 2.9 for P2 basis functions. Meanwhile, in 2D problems, the convergence rate is approximately 2 for P1 basis functions and 2.5 for P2 basis functions. These remarkable trends confirm the theoretical convergence rates observed in the experimental results.

Generalization errors. Figure 7 displays a comparison of the training and test errors between FEONet (w/o labeled data) and DeepONet (with 30 training data pairs) across different settings as performed in Section 4 (Table 1). The disparity between the training error and the test error serves as a reliable indicator of how effectively the models generalize the underlying phenomenon. As depicted in Figure 7, the generalization error of FEONet is notably smaller in comparison to that of DeepONet when trained with 30 data pairs.

5 Conclusion

In this paper, we introduce the FEONet, a novel deep learning framework that approximates nonlinear operators in infinite-dimensional Banach spaces using finite element approximations. Our operator learning method for solving PDEs is both simple and remarkably effective, resulting in significant improvements in predictive accuracy, domain flexibility, and data efficiency compared to state-of-the-art techniques. Furthermore, we demonstrate that the FEONet is capable of learning the solution operator of parametric PDEs, even in the absence of paired input-output training data, and accurately predicting solutions that exhibit singular behavior in thin boundary layers. Although the external force was considered as the input function in the experiment, the FEONet can be easily extended to operator learning for other input functions, such as boundary conditions, variable coefficients, or initial conditions of time-dependent PDEs (see Appendix D.4). We are confident that the FEONet can provide a new and promising approach to simulate and model intricate, nonlinear, and multiscale physical systems, with a wide range of potential applications in science and engineering.

6 Reproducibility Statement

All the data utilized in this paper was generated using Fenics as explained in Appendix D.3. Experiment details can be found in both the main text and the Appendix D. For reproducibility, we submit the code for the most fundamental settings as supplemental material.

References

Mark Alber, Adrian Buganza Tepole, William R Cannon, Suvranu De, Salvador Dura-Bernal, Krishna Garikipati, George Karniadakis, William W Lytton, Paris Perdikaris, Linda Petzold, et al. Integrating machine learning and multiscale modeling—perspectives, challenges, and opportunities in the biological, biomedical, and behavioral sciences. *NPJ digital medicine*, 2(1):115, 2019.

M. S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3, 2015. doi: 10.11588/ans.2015.100.20553.

Kendall Atkinson. *An introduction to numerical analysis*. John wiley & sons, 1991.

Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

Peter L. Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: risk bounds and structural results. *J. Mach. Learn. Res.*, 3(Spec. Issue Comput. Learn. Theory):463–482, 2002. ISSN 1532-4435. doi: 10.1162/153244303321897690. URL `https://doi.org/10.1162/153244303321897690`.

Edward A Bender. *An introduction to mathematical modeling*. Courier Corporation, 2000.

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

Oussama Boussif, Yoshua Bengio, Loubna Benabbou, and Dan Assouline. MAgnet: Mesh Agnostic Neural PDE Solver. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=bx2roi8hca8`.

Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message Passing Neural PDE Solvers. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=vSix3HPYKSU`.

Susanne C Brenner and Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer Science & Business Media, New York, 2008.

Richard L Burden, J Douglas Faires, and Annette M Burden. *Numerical analysis*. Cengage learning, 2015.

Mickaël D. Chekroun, Youngjoon Hong, and Roger Temam. Enriched numerical scheme for singularly perturbed barotropic quasi-geostrophic equations. *Journal of Computational Physics*, 416:109493, 2020.

Philippe G. Ciarlet. *The Finite Element Method for Elliptic Problems*. Society for Industrial and Applied Mathematics, 2002. doi: 10.1137/1.9780898719208. URL `https://epubs.siam.org/doi/abs/10.1137/1.9780898719208`.

Richard Courant and David Hilbert. *Methods of Mathematical Physics*. Interscience, New York, 2nd edition, 1953. ISBN 978-0-471-50447-4. Translated from German: Methoden der mathematischen Physik I, 2nd ed, 1931 [15].

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, 2(4):303–314, 1989. ISSN 0932-4194. doi: 10.1007/BF02551274. URL `https://doi.org/10.1007/BF02551274`.

Vladimir Fanaskov and Ivan Oseledets. Spectral Neural Operators. *arXiv preprint arXiv:2205.10573*, 2022.

Neil A Gershenfeld and Neil Gershenfeld. *The nature of mathematical modeling*. Cambridge university press, 1999.

Giorgio Gnecco and Marcello Sanguineti. Approximation error bounds via Rademacher's complexity. *Appl. Math. Sci. (Ruse)*, 2(1-4):153–176, 2008. ISSN 1312-885X.

Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. Physics-informed deep neural operators networks. *arXiv preprint arXiv:2207.05748*, 2022a.

Somdatta Goswami, Minglang Yin, Yue Yu, and George Em Karniadakis. A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 391:114587, 2022b. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.114587. URL `https://www.sciencedirect.com/science/article/pii/S004578252200010X`.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 481–490, 2016.

Youngjoon Hong and Chang-Yeol Jung. Enriched spectral method for stiff convection-dominated equations. *Journal of Scientific Computing*, 74(3):1325–1346, 2018.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL `https://www.sciencedirect.com/science/article/pii/089360809190009T`.

Thomas J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, 2000.

Rakhoon Hwang, Jae Yong Lee, Jin Young Shin, and Hyung Ju Hwang. Solving pde-constrained control problems using operator learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):4504–4512, Jun. 2022. doi: 10.1609/aaai.v36i4.20373. URL `https://ojs.aaai.org/index.php/AAAI/article/view/20373`.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric PDE problems with artificial neural networks. *arXiv preprint arXiv:1707.03351*, 2017.

Patrick Kidger and Terry Lyons. Universal Approximation with Deep Narrow Networks. In Jacob Abernethy and Shivani Agarwal (eds.), *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pp. 2306–2327. PMLR, 09–12 Jul 2020. URL `https://proceedings.mlr.press/v125/kidger20a.html`.

Seungchan Ko, Seok-Bae Yun, and Youngjoon Hong. Convergence analysis of unsupervised Legendre-Galerkin neural networks for linear second-order elliptic PDEs, 2022.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

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

Mats G Larson and Fredrik Bengzon. *The finite element method: theory, implementation, and applications*, volume 10. Springer Science & Business Media, 2013.

Jae Yong Lee, SungWoong CHO, and Hyung Ju Hwang. HyperDeepONet: learning operator with complex target function space using the limited resources via hypernetwork. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=OAw6V3ZAhSd`.

Randall J. LeVeque. *Finite-Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6755–6766. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/4b21cf96d4cf612f239a6c322b10c8fe-Paper.pdf`.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*, 2021a. URL `https://openreview.net/forum?id=c8P9NQVtmnO`.

Zongyi Li, Hongkai Zheng, Nikola B. Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *CoRR*, abs/2111.03794, 2021b. URL `https://arxiv.org/abs/2111.03794`.

Marten Lienen and Stephan Günnemann. Learning the Dynamics of Physical Systems from Sparse Observations with Finite Element Networks. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=HFmAukZ-k-2`.

Chia-Chiao Lin and Lee A Segel. *Mathematics applied to deterministic problems in the natural sciences*. SIAM, 1988.

A. Logg, K.-A. Mardal, G. N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012. doi: 10.1007/978-3-642-23099-8.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021a.

Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021b.

Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Xuhui Meng, Zhen Li, Dongkun Zhang, and George Em Karniadakis. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Computer Methods in Applied Mechanics and Engineering*, 370:113250, 2020.

Michael E Mortenson. *Mathematics for computer graphics applications*. Industrial Press Inc., 1999.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=roNqYL0_XP`.

Allan Pinkus. Approximation theory of the MLP model in neural networks. In *Acta numerica, 1999*, volume 8 of *Acta Numer.*, pp. 143–195. Cambridge Univ. Press, Cambridge, 1999. doi: 10.1017/S0962492900002919. URL `https://doi.org/10.1017/S0962492900002919`.

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

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms.* Cambridge University Press, 2014. ISBN 978-1-10-705713-5.

Carl P Simon, Lawrence Blume, et al. *Mathematics for economists*, volume 7. Norton New York, 1994.

John C Strikwerda. *Finite difference schemes and partial differential equations*. SIAM, 2004.

Martin J. Wainwright. *High-dimensional statistics*, volume 48 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 2019. ISBN 978-1-108-49802-9. doi: 10.1017/9781108627771. URL `https://doi.org/10.1017/9781108627771`. A non-asymptotic viewpoint.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, 7(40):eabi8605, 2021.

Liu Yang, Xuhui Meng, and George Em Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

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

Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

Olek C Zienkiewicz and Robert L Taylor. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 2000.

A Notations

The list of notations used throughout the paper is provided in Table 3.

Table 3: Notations

Notation	Meaning
u	ground-truth solution of the given PDE (target function)
u_h	finite element solution of the given PDE
$\widehat{u}_h = \widehat{u}_{h,n,M}$	solution predicted by the FEONet
h	discretization parameter for the FEM
n	number of learnable parameters in the neural network
M	number of input samples
α	true coefficients for the FEM basis
$\widehat{\alpha}$	predicted coefficient by the FEONet
ϕ_{cor}	corrector basis for the singular perturbation problem
$B[\cdot,\cdot]$	bilinear form of the variational form of the given PDE
$\ell(\cdot)$	linear functional of the variational form of the given PDE
δ_{ij}	Kronecker delta
$\mathbb{N}$	set of natural numbers
$\mathbb{R}$	set of real numbers
D	physical domain
Ω	parametric domain
$\mathbf{x}$	(vector-valued) physical variable
$\boldsymbol{\omega}$	(vector-valued) parametric random variable
div	divergence operator
$\boldsymbol{a}(\mathbf{x})$, $c(\mathbf{x})$	coefficients of PDEs
$f(\mathbf{x}, \boldsymbol{\omega})$	external force on $\mathbf{x} \in D$ parametrized by $\boldsymbol{\omega} \in \Omega$
$\mathcal{L}$	population loss of the FEONet
$\mathcal{L}^M$	empirical loss of the FEONet
$\mathcal{R}_M(\mathcal{G})$	Rademacher complexity of the function class $\mathcal{G}$

B Convergence analysis of FEONet

In this section, we shall provide the convergence analysis of the FEONet to establish the theoretical justification of the proposed numerical scheme. In order to convey the idea of the proof clearly, we will focus solely on self-adjoint equations with homogeneous Dirichlet boundary conditions:

$$\begin{aligned} -\text{div}\,(\boldsymbol{a}(\mathbf{x})\nabla u) + c(\mathbf{x})u &= f(\mathbf{x}) \quad \text{in } D, \\ u &= 0 \qquad\;\; \text{on } \partial D, \end{aligned} \tag{18}$$

where the coefficient $\boldsymbol{a}(\mathbf{x})$ is uniformly elliptic and $c(\mathbf{x}) \geq 0$. Note, however, that our analysis can be extended to more general cases in a straightforward manner.

B.1 Mathematical formulation

As described before, the input of neural networks is an external forcing term f, which is parametrized by the random parameter $\boldsymbol{\omega}$ in the probability space $(\Omega, \mathcal{G}, \mathbb{P})$. In this section, we will regard $f(\mathbf{x}; \boldsymbol{\omega})$ as a bivariate function defined on $D \times \Omega$, and assume that

$$f(\mathbf{x}; \boldsymbol{\omega}) \in C(\Omega; L^1(D)) := \left\{ f : \Omega \to L^1(D) : \sup_{\boldsymbol{\omega} \in \Omega} \int_D |f(\mathbf{x}; \boldsymbol{\omega})| \,\mathrm{d}\mathbf{x} < \infty \right\}. \tag{19}$$

For each $\boldsymbol{\omega} \in \Omega$, the external force $f(\mathbf{x}; \boldsymbol{\omega})$ is determined, and the corresponding weak solution is denoted by $u(\mathbf{x}; \boldsymbol{\omega})$ which satisfies the following weak formulation:

$$B[u, v] := \int_D [\boldsymbol{a}(\mathbf{x})\nabla u \cdot \nabla v + c(\mathbf{x})uv] \,\mathrm{d}\mathbf{x} = \int_D f(\mathbf{x})v \,\mathrm{d}\mathbf{x} =: \ell(v) \quad \forall v \in H_0^1(D). \tag{20}$$

For given $h > 0$, let $V_h \subset H_0^1(D)$ be a conforming finite element space generated by the basis functions $\{\phi_k\}_{k=1}^{N(h)}$ and $u_h \in V_h$ be a finite element approximation of u satisfying the Galerkin approximation

$$B[u_h, v_h] = \ell(v_h) \quad \forall v_h \in V_h. \tag{21}$$

We shall write

$$u_h(\mathbf{x}, \boldsymbol{\omega}) = \sum_{k=1}^{N(h)} \alpha_k^*(\boldsymbol{\omega})\phi_k(\mathbf{x}), \tag{22}$$

where α^* is the solution of the linear algebraic equations

$$A\alpha^* = F, \tag{23}$$

with

$$A_{ik} = B[\phi_k, \phi_i] \quad \text{and} \quad F_i = \ell(\phi_i). \tag{24}$$

In other words, u_h in (22) is the finite element approximation of the solution to the equation (18), as described in Section 3.1. It is noteworthy that instead of the equations (23), α^* can also be characterized in the following way:

$$\alpha^* = \operatorname*{arg\,min}_{\alpha \in C(\Omega, \mathbb{R}^{N(h)})} \mathcal{L}(\alpha), \tag{25}$$

where $\mathcal{L}$ is the population loss

$$\mathcal{L}(\alpha) = \mathbb{E}_{\boldsymbol{\omega} \sim \mathbb{P}_\Omega}\left[\sum_{i=1}^{N(h)} |B[\widehat{u}(\boldsymbol{\omega}), \phi_i] - \ell(\phi_i; (\boldsymbol{\omega}))|^2\right] = \|A\alpha(\boldsymbol{\omega}) - F(\boldsymbol{\omega})\|_{L^2(\Omega)}^2. \tag{26}$$

Next, let us denote the class of feed-forward neural networks by $\mathcal{N}_n$. Here, the subscript n represents the architecture of neural networks and we assume that $\mathcal{N}_{n_2}$ has more expressive power than $\mathcal{N}_{n_1}$ provided that $n_1 \leq n_2$. For example, n can be the number of layers with bounded width or the number of neurons when the number of layers is fixed. The class of neural networks is known to be a suitable ansatz space for the nonlinear approximation, as supported by the universal approximation theorem (see, for example, Cybenko (1989); Hornik (1991); Pinkus (1999); Kidger & Lyons (2020)). For the later analysis, in this section, we shall assume that for all the neural networks under consideration, the last layer is activated by a bounded activation function (e.g. sigmoid, tanh, etc.), so that the resulting neural networks are uniformly bounded. Then by a simple scaling argument, we can show that the universal approximation theorem still holds for this new class of neural networks: see, for example, Theorem 2.2 in Ko et al. (2022).

Now as a neural network approximation of α^*, we seek for $\widehat{\alpha}(n) : \Omega \to \mathbb{R}^{N(h)}$ solving the residual minimization problem

$$\widehat{\alpha}(n) = \operatorname*{arg\,min}_{\alpha \in \mathcal{N}_n} \mathcal{L}(\alpha), \tag{27}$$

where the loss function is minimized over $\mathcal{N}_n$, and we shall write the corresponding solution by

$$u_{h,n}(\mathbf{x}; \boldsymbol{\omega}) = \sum_{k=1}^{N(h)} \widehat{\alpha}(n)_k(\boldsymbol{\omega})\phi_k(\mathbf{x}). \tag{28}$$

Note that for the neural network under consideration $\alpha \in \mathcal{N}_n$, the input vector is $\boldsymbol{\omega} \in \Omega$ which determines the external force $f(\mathbf{x}; \boldsymbol{\omega})$ and the output is the coefficient vector in $\mathbb{R}^{N(h)}$.

As a final step, let us define the solution to the following discrete residual minimization problem:

$$\widehat{\alpha}(n, M) = \operatorname*{arg\,min}_{\alpha \in \mathcal{N}_n} \mathcal{L}^M(\alpha), \tag{29}$$

where $\mathcal{L}^M$ is the empirical loss defined by the Monte–Carlo integration of $\mathcal{L}(\alpha)$:

$$\mathcal{L}^M(\alpha) = \frac{|\Omega|}{M}\sum_{m=1}^{M}\sum_{i=1}^{N(h)} |B[\widehat{u}(\boldsymbol{\omega}_m), \phi_i] - \ell(\phi_i; (\boldsymbol{\omega}_m))|^2 = \frac{|\Omega|}{M}\sum_{m=1}^{M} |A\alpha(\boldsymbol{\omega}_m) - F(\boldsymbol{\omega}_m)|^2, \tag{30}$$

with $\{\boldsymbol{\omega}_n\}_{m=1}^{M}$ is an i.i.d. sequence of random variables distributed according to $\mathbb{P}_\Omega$. Then we write the corresponding solution as

$$u_{h,n,M}(\mathbf{x};\boldsymbol{\omega}) = \sum_{k=1}^{N(h)} \widehat{\alpha}(n,M)_k(\boldsymbol{\omega})\phi_k(\mathbf{x}), \tag{31}$$

which is the numerical solution actually computed by the scheme proposed in the present paper.

In this paper, we assume that it is possible to find the exact minimizers for the problems (27) and (29), and the error that occurred by the optimization can be ignored.

In order to provide appropriate theoretical backgrounds for the proposed method, it would be reasonable to show that our solution is sufficiently close to the approximate solution computed by the proposed scheme for various external forces, as the index $n \in \mathbb{N}$ for a neural-network architecture and the number of input samples $M \in \mathbb{N}$ goes to infinity. Mathematically, it can be formulated as

$$\|u - u_{h,n,M}\|_{L^2(\Omega;L^2(D))} \to 0 \quad \text{as } h \to 0 \text{ and } n, M \to \infty. \tag{32}$$

The main error can be split into three parts:

$$u - u_{h,n,M} = (u - u_h) + (u_h - u_{h,n}) + (u_{h,n} - u_{h,n,M}). \tag{33}$$

The first error is the one caused by the finite element approximation, which is assumed to be negligible for a proper choice of h. In fact, according to the estimate (7), we can reduce the error as much as we want if we choose proper $h > 0$. Henceforth, we shall assume that we have chosen suitable $h > 0$ which guarantees a sufficiently small error for the finite element approximation. The second error is referred to as the approximation error, as it appears when we approximate the target function with a class of neural networks. The third error is often called the generalization error, which measures how well our approximation predicts solutions for unseen data. We will focus on proving that our approximate solution converges to the finite element solution (which is sufficiently close to the true solution) as a neural network architecture grows and the number of input samples goes to infinity.

B.2 Approximation error

We first note from (26) and (30), that A defined in (22), (23) mostly determines the structures of the loss functions and it would be advantageous for us to analyze the loss function if we know more about the matrix. Note that the matrix A is determined by the structure of the given differential equations, the choice of basis functions and the boundary conditions. Therefore, the characterization of A which is useful for the analysis of the loss function and can cover a wide range of PDE settings simultaneously is important. The next lemma addresses this issue, which is quoted from Ko et al. (2022).

Lemma B.1. *Suppose that the matrix $A \in \mathbb{R}^{N(h)\times N(h)}$ is symmetric and invertible, and let $\rho_{\min} = \min_i\{|\lambda_i|\}$, $\rho_{\max} = \max_i\{|\lambda_i|\}$ where $\{\lambda_i\}$ is the family of eigenvalues of the matrix A. Then we have for all $\mathbf{x} \in \mathbb{R}^{N(h)}$,*

$$\rho_{\min}|\mathbf{x}| \le |A\mathbf{x}| \le \rho_{\max}|\mathbf{x}|. \tag{34}$$

Since the equation under consideration (18) is self-adjoint, the corresponding bilinear form $B[\cdot,\cdot]$ defined in (20) is symmetric which automatically guarantees that the matrix A in our case is symmetric. Furthermore, due to the facts that the coefficient $\boldsymbol{a}(\cdot)$ is uniformly elliptic and $c(\cdot)$ is non-negative, the bilinear form $B[\cdot,\cdot]$ is coercive, which implies that A is positive-definite. Therefore, we can apply Lemma B.1 to the matrix A of our interest.

We are now ready to prove the approximation error for neural networks converges to zero, which is encapsulated in the following theorem.

Theorem B.2. *Assume that* (19) *holds. Then we have that*

$$\|\alpha^* - \widehat{\alpha}(n)\|_{L^2(\Omega)} \to 0 \quad \text{as } n \to \infty. \tag{35}$$

Proof. Since A is symmetric and invertible, by Proposition B.1, we obtain

$$\begin{aligned}\|\alpha^* - \widehat{\alpha}(n)\|_2^2 &\lesssim \|A\alpha^* - A\widehat{\alpha}(n)\|_2^2 \lesssim \left(\|A\alpha^* - F\|_2^2 + \|A\widehat{\alpha}(n) - F\|_2^2\right) \\ &= \mathcal{L}(\widehat{\alpha}(n)) \lesssim \inf_{\alpha\in\mathcal{N}_n} \mathcal{L}(\alpha) = \inf_{\alpha\in\mathcal{N}_n} \|A\alpha - F\|_2^2 \\ &\lesssim \inf_{\alpha\in\mathcal{N}_n} \left(\|A\alpha - A\alpha^*\|_2^2 + \|A\alpha^* - F\|_2^2\right) \\ &\lesssim \inf_{\alpha\in\mathcal{N}_n} \|\alpha - \alpha^*\|_2^2.\end{aligned}$$

Note that the term $\rho_{\max}/\rho_{\min}$ may depend on $h > 0$. But as we mentioned before, we assumed that $h > 0$ is fixed with the sufficiently small finite element error. Then by the universal approximation property , $\inf_{\alpha\in\mathcal{N}_n} \|\alpha - \alpha^*\|_2^2 \to 0$ as $n \to \infty$, which completes the proof. □

B.3 GENERALIZATION ERROR

We first introduce the concept so-called *Rademacher complexity* Bartlett & Mendelson (2002).

Definition B.3. *For a collection $\{X_i\}_{i=1}^M$ of i.i.d. random variables, we define the Rademacher complexity of the function class $\mathcal{G}$ by*

$$R_M(\mathcal{G}) = \mathbb{E}_{\{X_i,\varepsilon_i\}_{i=1}^M}\left[\sup_{f\in\mathcal{G}}\left|\frac{1}{M}\sum_{i=1}^M \varepsilon_i f(X_i)\right|\right],$$

where ε_i's denote i.i.d. Bernoulli random variables, in other words, $\mathbb{P}(\varepsilon_i = 1) = \mathbb{P}(\varepsilon_i = -1) = \frac{1}{2}$ for all $i = 1, \cdots, M$.

Note that the Rademacher complexity is the expectation of the maximum correlation between the random noise $(\varepsilon_1, \cdots, \varepsilon_M)$ and the vector $(f(X_1), \cdots, f(X_M))$, where the supremum is taken over the class of functions $\mathcal{G}$. By intuition, the Rademacher complexity of $\mathcal{G}$ is often used to measure how the functions from $\mathcal{G}$ can fit random noise. For a more detailed discussion on the Rademacher complexity, see Gnecco & Sanguineti (2008); Wainwright (2019); Shalev-Shwartz & Ben-David (2014).

Next, we shall find the connection between the generalization error and the Rademacher complexity for the uniformly bounded function class $\mathcal{G}$. The following theorem, we assume that the function class is b-uniformly bounded, meaning that $\|f\|_\infty \le b$ for arbitrary $f \in \mathcal{G}$.

Theorem B.4. [Theorem 4.10 in Wainwright (2019)] *Assume that the function class $\mathcal{G}$ is b-uniformly bounded and let $M \in \mathbb{N}$. Then for arbitrary small $\delta > 0$, we have*

$$\sup_{f\in\mathcal{G}}\left|\frac{1}{M}\sum_{i=1}^M f(X_i) - \mathbb{E}[f(X)]\right| \le 2R_M(\mathcal{G}) + \delta,$$

with probability at least $1 - \exp(-\frac{M\delta^2}{2b^2})$.

Now we define the function class of interest

$$\mathcal{G}_n := \{|A\alpha - F|^2 : \alpha \in \mathcal{N}_n\}, \tag{36}$$

where the matrix A and the vector F are defined in (24). Then by Lemma B.1, we have that

$$\|A\alpha - F\|_{L^\infty(\Omega)} \le \|A\alpha\|_{L^\infty(\Omega)} + \|F\|_{L^\infty(\Omega)} \le \rho_{\max}\|\alpha\|_{L^\infty(\Omega)} + \|f\|_{C(\Omega;L^1(D))}.$$

Since the class of neural networks under consideration is uniformly bounded and (19) holds, we see that for any $n \in \mathbb{N}$, $\mathcal{G}_n$ is $\tilde{b}$-uniformly bounded for some $\tilde{b} > 0$. The following lemma is the direct consequence of Theorem B.4 within our setting.

Lemma B.5. *Let $\{\boldsymbol{\omega}_m\}_{m=1}^M$ be i.i.d. random samples selected from the distribution $\mathbb{P}_\Omega$. Then for arbitrary small $\delta > 0$, we obtain with probability at least $1 - 2\exp(-\frac{M\delta^2}{32\tilde{b}^2})$ that*

$$\sup_{\alpha\in\mathcal{N}_n}\left|\mathcal{L}^M(\alpha) - \mathcal{L}(\alpha)\right| \le 2R_n(\mathcal{G}_n) + \frac{\delta}{2}. \tag{37}$$

From Lemma B.5, now we shall establish the following convergence result for the generalization error. Note that we assume here that for any $n \in \mathbb{N}$, the Rademacher complexity of $\mathcal{G}_n$ converges to zero, which is known to be true in many cases.

Theorem B.6. *Suppose that* (19) *holds, and we further assume that for any* $n \in \mathbb{N}$, $\lim_{M\to\infty} R_M(\mathcal{G}_n) = 0$. *Then we have with probability* 1 *that*

$$\lim_{n\to\infty}\lim_{M\to\infty} \|\widehat{\alpha}(n,M) - \widehat{\alpha}(n)\|_{L^2(\Omega)} = 0.$$

Proof. By Proposition B.1 and the definition (27), we obtain

$$\begin{aligned}\|\widehat{\alpha}(n) - \widehat{\alpha}(n,M)\|_2^2 &\lesssim \|A\widehat{\alpha}(n) - A\widehat{\alpha}(n,M)\|_2^2 \\ &\lesssim \left(\|A\widehat{\alpha}(n) - F\|_2^2 + \|A\widehat{\alpha}(n,M) - F\|_2^2\right) \\ &= \left(\mathcal{L}(\widehat{\alpha}(n)) + \mathcal{L}(\widehat{\alpha}(n,M))\right) \\ &\lesssim \mathcal{L}(\widehat{\alpha}(n,M)).\end{aligned} \tag{38}$$

We next apply Lemma B.5 for $\delta = 2M^{-\frac{1}{2}+\varepsilon}$ with $0 < \varepsilon < \frac{1}{2}$. Then with probability at least $1 - 2\exp(-\frac{M^{2\varepsilon}}{8\widetilde{b}^2})$, we have from the minimality of $\widehat{\alpha}(n,M)$ that,

$$\mathcal{L}(\widehat{\alpha}(n,M)) \le \mathcal{L}^M(\widehat{\alpha}(n,M)) + 2R_M(\mathcal{G}_n) + M^{-\frac{1}{2}+\varepsilon} \le \mathcal{L}^M(\widehat{\alpha}(n)) + 2R_M(\mathcal{G}_n) + M^{-\frac{1}{2}+\varepsilon}.$$

Using Lemma B.5 again gives us that

$$\mathcal{L}(\widehat{\alpha}(n,M)) \le \mathcal{L}(\widehat{\alpha}(n)) + 4R_M(\mathcal{G}_n) + 2M^{-\frac{1}{2}+\varepsilon}.$$

Letting $M \to \infty$ on (38), we obtain that

$$\lim_{M\to\infty} \|\widehat{\alpha}(n,M) - \widehat{\alpha}(n)\|_2^2 \lesssim \mathcal{L}(\widehat{\alpha}(n)).$$

As we did before, we conclude that

$$\begin{aligned}\lim_{n\to\infty}\lim_{M\to\infty} \|\widehat{\alpha}(n,M) - \widehat{\alpha}(n)\|_2^2 &\lesssim \lim_{n\to\infty} \mathcal{L}(\widehat{\alpha}(n)) \\ &\lesssim \lim_{n\to\infty} \inf_{\alpha\in\mathcal{N}_n} \|\alpha - \alpha^*\|_2^2 = 0,\end{aligned}$$

which completes the proof. ☐

B.4 Proof of main theorem

From Theorem B.2 and Theorem B.6 combined with the triangular inequality, we have

$$\lim_{n\to\infty}\lim_{M\to\infty} \|\alpha^* - \widehat{\alpha}(n,M)\|_{L^2(\Omega)} = 0. \tag{39}$$

Let us now prove the main theorem on the convergence.

Theorem B.7 (Main theorem)**.** *Suppose* (19) *holds, and assume that for all* $n \in \mathbb{N}$, $R_M(\widetilde{\mathcal{G}}_n) \to 0$ *as* $M \to \infty$, *with* $\widetilde{\mathcal{G}}_n := \{|A\alpha - F|^2 : \alpha \in \mathcal{N}_n\}$. *Then for given* $h > 0$, *we have with probability* 1 *that*

$$\lim_{n\to\infty}\lim_{M\to\infty} \|u_h - u_{h,n,M}\|_{L^2(\Omega;L^2(D))} = 0. \tag{40}$$

Proof. From Theorem B.2, Theorem B.6, we note that for a fixed $h > 0$,

$$\begin{aligned}\|u_h - u_{h,n,M}\|_{L^2(\Omega;L^2(D))}^2 &= \int_\Omega\int_D \left|\sum_{i=1}^{N(h)} (\alpha_i^* - \widehat{\alpha}(n,M)_i)\phi_i\right|^2 \mathrm{d}\mathbf{x}\,\mathrm{d}\boldsymbol{\omega} \\ &= \int_\Omega\int_D \left|\sum_{i,j=1}^{N(h)} (\alpha_i^* - \widehat{\alpha}(n,M)_i)(\alpha_j^* - \widehat{\alpha}(n,M)_j)\phi_i\phi_j\right|^2 \mathrm{d}\mathbf{x}\,\mathrm{d}\boldsymbol{\omega} \\ &\le \int_\Omega\int_D \left(\sum_{i,j=1}^{N(h)} |\alpha_i^* - \widehat{\alpha}(n,M)_i|^2|\phi_i|^2 + \sum_{i,j=1}^{N(h)} |\alpha_j^* - \widehat{\alpha}(n,M)_j|^2|\phi_j|^2\right) \mathrm{d}\mathbf{x}\,\mathrm{d}\boldsymbol{\omega} \\ &\le \int_\Omega\int_D 2N(h)\sum_{k=1}^{N(h)} |\alpha_k^* - \widehat{\alpha}(n,M)_k|^2|\phi_k|^2 \,\mathrm{d}\mathbf{x}\,\mathrm{d}\boldsymbol{\omega} \\ &\le 2|D|N(h)\|\alpha^* - \widehat{\alpha}(n,M)\|_{L^2(\Omega)}^2 \to 0,\end{aligned}$$

where we have used the fact that $|\phi_j| \le 1$ for all $1 \le j \le N(h)$ together with Young's inequality. ☐

C Comparison of FEONet with other models and methods

C.1 Comparing the FEONet with other physics-informed operator learning models (PIDeeepONet and PINO)

The proposed FEONet model is a data-free approach that does not require labeled training data, and it has a number of advantages in learning the solution operator as confirmed in Section 4. Unlike methods designed for a single instance, such as PINN Raissi et al. (2019), this novel numerical scheme enables the handling of multiple instances of parametric PDEs. On the other hand, well-established models for operator learning, such as FNO Li et al. (2021a) and DeepONet Lu et al. (2021a), depend on labeled training data for their training process. To address the limitation of these operator-learning methods, recent advances in physics-informed methods, such as Physics-informed DeepONet (PIDeepONet) Wang et al. (2021) and Physics-informed neural operator (PINO) Li et al. (2021b) have emerged. These methods utilize the PDE residual loss, enabling the training of the models without labeled data. However, these methods are not suitable for complex 2D domains or singular perturbation problems as demonstrated in the experiments conducted in Section 4.2.

vs PIDeepONet. The FEONet effectively utilizes the weak form of the given PDE by employing it as a loss function over the triangulation of the physical domain. In contrast, PIDeepONet requires a complex grid sampling process in a complicated 2D domain, resulting in computational costs associated with calculating the residual loss using automatic differentiation. Another challenging aspect is determining the weighting of the loss function related to boundary conditions, which is used to satisfy the boundary conditions in the physics-informed loss. Moreover, as shown in Table 2 and Figure 5, when a small diffusion coefficient $\varepsilon > 0$ is used in a PDE residual loss term, the models struggle to accurately learn the corresponding solutions of the PDEs. This raises a significant concern regarding reliability, as illustrated in Figure 14, which displays the predictions from various training trials. In this regard, we also conducted a comparative analysis of the supervised DeepONet approach for PDE problems under various conditions, considering different amounts of available data.

vs PINO. The comparison with the FNO and PINO models was excluded in Section 4 because they are not directly suitable for problems on complex geometry domains with non-periodic boundary conditions, such as those depicted in Figure 1. To elucidate this, we provide additional experiments using the PINO as described in Li et al. (2021b). The authors proposed extending the FNO with Fourier continuation and zero padding in the input for handling non-periodic boundary conditions. For this comparison, we trained PINO with Dirichlet boundary conditions on a square domain containing a hole, and employed circular masking. Figure 8 displays a solution profile obtained by predicting a test sample using models trained with both PINO and FEONet. As evident in the third plot, PINO struggles to adhere to boundary conditions in complex domains, especially those with a central hole. This challenge underpins our decision to primarily compare with PIDeepONet among the physics-informed models in Section 4.

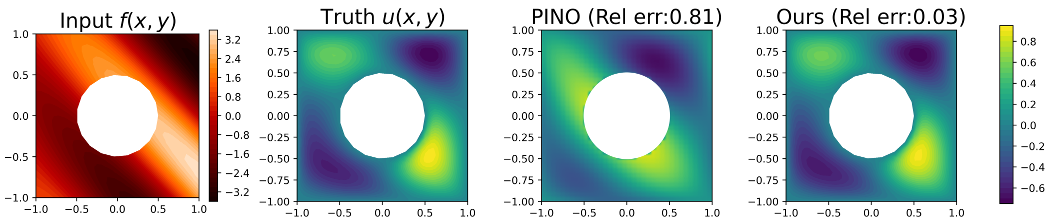


Figure 8: The profile of solution prediction for the square with a hole domain using PINO and FEONet.

C.2 Comparing the FEONet with the classical FEM

The FEONet is an operator-learning model designed to learn the relationship between PDE parameters and their corresponding solutions. Once trained, the FEONet can make a fast solution inference for varying PDE parameters. This capability is the primary reason why operator networks are gaining attention in scientific machine learning (Kovachki et al., 2021; Goswami et al., 2022a; Lu et al., 2022).

On the other hand, if we try to solve PDEs for various parameters using the traditional FEM, we need to solve the PDE anew every time the parameters change. Therefore, if we are dealing with a problem where real-time prediction is crucial (for example, when creating a solution database for a specific engineering problem), this incurs significant computational costs. Moreover, when solving nonlinear problems, we need to apply the iterative methods for each PDE parameter, which causes a computation overhead. As mentioned earlier, the FEONet is versatile in addressing this issue, and this is the most significant difference between the traditional FEM and the FEONet.

To illustrate this point quantitatively, let us consider a scenario where we need to solve 10,000 parametric PDEs with varied parameters - a situation often encountered in computational material database generation. In such cases, operator learning methods like the FEONet is confirmed to be substantially faster than the FEM. Thus, from a fast-inference perspective, models like the FEONet have a clear advantage. Table 4 compares the inference time taken by calculating and comparing based on the number of input samples.

Table 4: Comparison of computational times for FEONet and FEM with different numbers of input samples.

# input samples	1	10	100	1000	10000
FEM	0.022 (s)	0.254 (s)	1.308 (s)	11.530 (s)	116.035 (s)
FEONet (Ours)	0.007 (s)	0.021 (s)	0.150 (s)	3.028 (s)	29.231 (s)

While the solutions computed by the FEONet might not as accurate as those obtained through the classical FEM, it's essential to place this in the broader context of operator learning methods. No existing operator learning method currently outperforms traditional numerical techniques in terms of accuracy. The primary objective of operator learning methods (Lu et al., 2021a; Wang et al., 2021; Li et al., 2021a), including the FEONet, isn't necessarily to achieve the highest accuracy but to offer swift predictions. This is especially vital in tfields such as computational materials, where engineers aim to establish extensive databases — sometimes comprising over half a million datasets. For applications like the inverse design of materials, the focus is more on achieving adequate quality rapidly rather than pursuing perfect accuracy. Thus, when we assess the FEONet, we primarily compare it with other operator learning methods, not with classical numerical methods, as they are different knives serving distinct purposes.

C.3 Comparing the FEONet with other deep learning-based models

Additional comparisons of various deep learning-based approaches for solving PDEs are demonstrated in Table 5.

Table 5: Comparison of various machine learning approaches for solving PDEs.

	Unsupervised	PDE instances	Prediction location	Applicable for complex domains and singularly perturbed problem
PINN (Raissi et al., 2019), DRM (Yu et al., 2018)	No	Single instance	Entire domain	△
DON (Lu et al., 2021a)	No	Multiple instances	Entire domain	△
FNO (Li et al., 2021a)	No	Multiple instances	Grid points	×
PIDeepONet (Wang et al., 2021)	Yes	Multiple instances	Entire domain	△
PINO (Li et al., 2021b)	Yes	Multiple instances	Grid points	×
Graph-based neural PDE solver (Brandstetter et al., 2022; Lienen & Günnemann, 2022).	No	Multiple instances	Grid points	-
Ours	Yes	Multiple instances	Entire domain	○

D Experiment details

D.1 Hyperparameters

For the problems under consideration, we used the neural network, which consists of 6 convolutional layers with swish activation followed by a fully connected layer flattening the output. For 1D problems, we used Conv1D, while Conv2D was used for 2D problems. The FEONet was trained with the LBFGS optimizer along with the following hyperparameters.

- Maximal number of iterations per optimization hestep : 10,
- Termination tolerance on first-order optimality : 10^{-15},
- Termination tolerance on function value/parameter changes : 10^{-15},
- Update history size : 10.

For both the DeepONet and the PIDeepONet, a fully connected neural network with a depth of 3 and a width of 100 was used for the trunk and branch networks. The two models differ only in the loss function, where the training data and the residual of the PDE were used. The PINN also used a fully connected neural network with a depth of 3 and a width of 100. All these models used the tanh activation function and a learning rate of 10^{-4} with the commonly used ADAM optimizer. To ensure the sufficient convergence of the training results for the boundary layer problem, we conducted five experiments with the PINN for 5×10^5 iterations and five experiments with the PIDeepONet for 10^4 iterations. Note that computing the residual loss of PDE for every input function f in the PIDeepONet requires significant computation time using the automatic differentiation. We used the Intel Xeon Gold 6226R processor and NVIDIA RTX A6000 48GB GPU.

Table 6: Parameters for each problem

	Input function	PDE dimension	P1 or P2	# Element	Iteration (FEONet)	Iteration (DeepONet)
Domain I Domain II Domain III	$m_j \in [1, 2]$ $n_j \in [0, \pi]$	2D	P2	392 551 334	5×10^4	5×10^5
BC I	$m_j \in [3, 5]$	1D	P2	24	1.5×10^5	5×10^5
BC II	$n_j \in [0, 2\pi]$			32	2×10^5	5×10^5
Eq I	$m_j \in [3, 5]$	1D	P2	32	1.5×10^5	5×10^5
Eq II	$n_j \in [0, 2\pi]$		P1	128	3.5×10^5	5×10^5
Singular	$m_j \in [3, 5]$ $n_j \in [0, 2\pi]$	1D	P1	32	5×10^4	5×10^5

D.2 Trade-off of FEONet

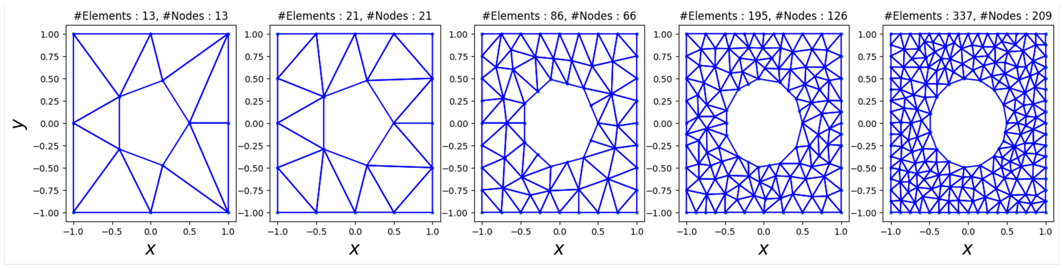
The order of the polynomial basis and the number of elements employed in FEONet for each domain, boundary condition, and equation are presented in the third and fourth columns of Table 6. Utilizing a higher-order basis and a greater number of elements for domain triangulation in FEONet can enhance prediction accuracy, but it comes at the cost of increased memory usage and longer training time. Therefore, determining the appropriate basis and element number for FEONet involves a trade-off between prediction accuracy and computational cost. Further details can be found in Table 6.

D.3 FEniCS for ground-truth solution, domain triangulation and nodal basis

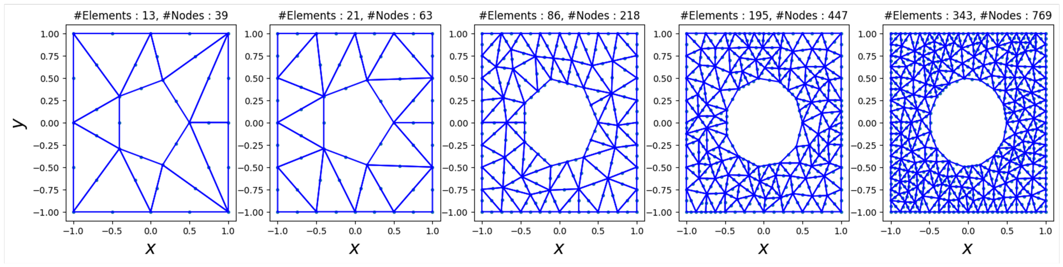
The FEniCS (Alnaes et al., 2015; Logg et al., 2012), which stands for Finite Element Computational Software, is a widely used open-source software suite that is specifically designed for solving PDEs using the FEM. It offers a robust and user-friendly interface that enables solving a broad range of PDEs in various settings. Using the FEniCS, we obtained solutions with very small errors for each PDE problem on sufficiently fine meshes, which were used as the ground-truth solutions for each problem. We were able to easily obtain nodes and nodal basis by triangulating the 2D domain as shown in Figure 1 and Figure 9. Additionally, in our experiments, we were able to compute the loss function (9) by using the FEniCS both for the P1 and the P2 elements.

D.4 Types and random generation of input functions for FEONet

This paper specifically has focused on the learning of the solution operator of PDEs from external forcing terms. Notably, the FEONet can be readily extended to handle various input functions, such


#Elements : 13, #Nodes : 13
#Elements : 21, #Nodes : 21
#Elements : 86, #Nodes : 66
#Elements : 195, #Nodes : 126
#Elements : 337, #Nodes : 209
x
y

(a) Triangulation and nodes for P1-element method


#Elements : 13, #Nodes : 39
#Elements : 21, #Nodes : 63
#Elements : 86, #Nodes : 218
#Elements : 195, #Nodes : 447
#Elements : 343, #Nodes : 769
x
y

(b) Triangulation and nodes for P2-element method

Figure 9: FEM triangulation of Domain II (square with a hole) with varying numbers of elements.

as boundary conditions, coefficients, or initial conditions, in the learning process. We shall confirm this based on some additional experiments with various input functions.

As a first example, the FEONet is capable of learning the operator mapping $\mathcal{G} : c(x) \mapsto u(x)$ from the variable coefficient $c(x)$ to the corresponding solution $u(x)$ of a PDE (14). Figure 10 displays the solution profile obtained when variable coefficients are considered as input. Next, we address the case when the input is a boundary condition. We consider the 2D Poisson equation as

$$\begin{aligned} -\Delta u(x, y) &= f(x, y), \quad (x, y) \in D, \\ u(x, y) &= g(x, y), \quad (x, y) \in \partial D. \end{aligned} \tag{41}$$

where $f(x, y) = 2$ and the domain D is a square with a hole (see (b) in Figure 1). The FEONet is utilized to learn the operator $\mathcal{G} : g(x, y) \mapsto u(x, y)$. Figure 11 demonstrates that even in the case of an operator that takes the boundary condition $g(x, y)$ as input, we can predict the solution with high accuracy. Finally, we consider the time-dependent problem with varying initial conditions. We consider the 1D convection-diffusion equation

$$\begin{aligned} u_t - \nu u_{xx} + b u_x &= 0, \qquad t \in [0, 1], x \in D = [-1, 1], \\ u(0, x) &= u_0(x), \quad x \in D, \\ u(t, x) &= 0, \qquad x \in \partial D, \end{aligned} \tag{42}$$

where $\nu = 0.1$ and $b = -1$. Our aim is to learn the operator $\mathcal{G} : u_0(x) \mapsto u(t, x)$. We use the implicit Euler method to discretize the time with $\Delta t = 0.01$. We employed the marching-in-time scheme proposed by Krishnapriyan et al. (2021) to sequentially learn and predict solutions over the time domain $t = [0, 1]$, divided into 10 intervals. Notably, for training FEONet, we utilized only the input function $u_0(x)$ and equation (42) without any additional data. Figure 12 demonstrates the successful prediction of the true solutions using the FEONet for arbitrary two initial conditions from the test data set. These experiments demonstrate the versatility and adaptability of the FEONet approach for accommodating different types of input functions.

In order to train the network, we generate random input functions (e.g. external forcing functions, variable coefficient functions, boundary conditions, and initial conditions). Inspired by Bar-Sinai et al. (2019), we created a random signal $f(\mathbf{x}; \boldsymbol{\omega})$ as a linear combination of sine functions and cosine functions. More precisely, we use

$$f(x) = m_0 \sin(n_0 x) + m_1 \cos(n_1 x) \tag{43}$$

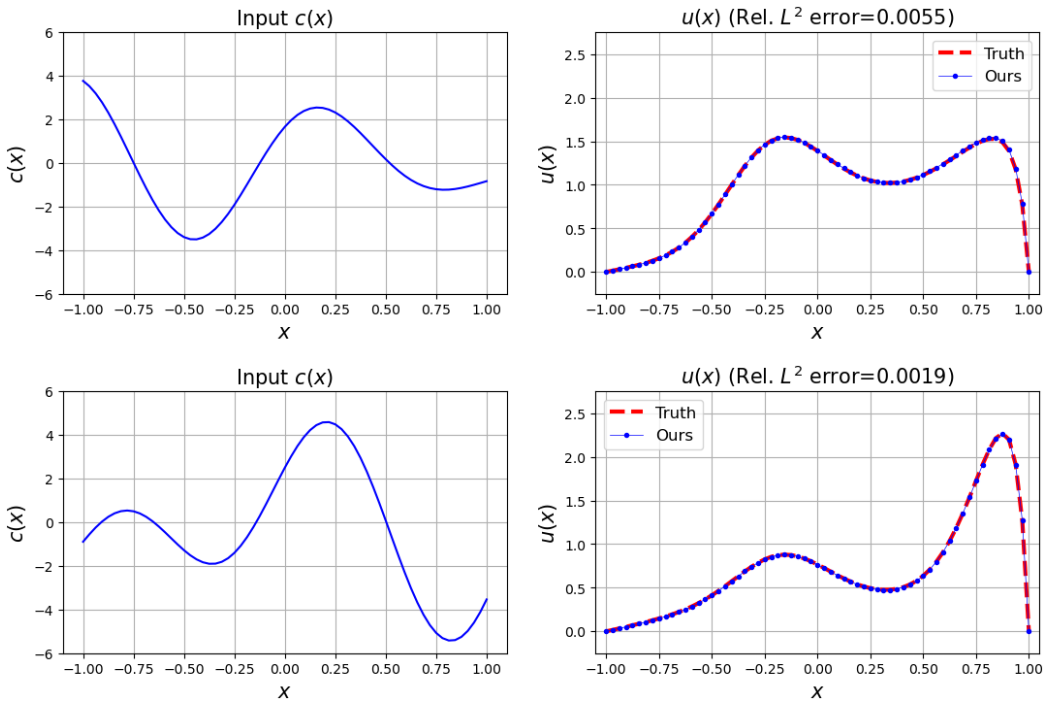


Figure 10: Solution profiles predicted by the trained FEONet for PDE (14) obtained from variable coefficients $c(x)$ as two distinct input functions.

for a 1D case and

$$f(x, y) = m_0 \sin(n_0 x + n_1 y) + m_1 \cos(n_2 x + n_3 y) \tag{44}$$

for a 2D case where m_i for $i = 1, 2$ and n_j for $j = 0, 1, 2, 3$ are drawn independently from the uniform distributions whose ranges are shown in the second column of Table 6. In our experiments for DeepONet with input-output data pairs, we prepare a total of 30/300/3000 pairs of (f, u) by applying the FEM on sufficiently fine meshes. It is worth noting that even when considering different random input functions, such as those generated by Gaussian random fields, we consistently observe similar results. This robustness indicates the reliability and stability of the FEONet approach across various input scenarios.

D.5 Description of various domains with complex geometries

Figure 1 illustrates three complex domains considered in this paper. Firstly, Domain I is chosen as a domain with a smooth boundary, represented by a circle centered at $(0, 0)$ with radius 1, to accommodate non-rectangular shapes. Secondly, Domain II is designed to have a challenging geometry with holes, excluding a circle of radius 0.5 centered at $(0, 0)$ from the square domain $[-1, 1] \times [-1, 1]$. Lastly, Domain III is created to have an irregular polygon with sharp turns, connecting the vertices $(-1.0, -1.0)$, $(1.0, -1.0)$, $(1.0, 0.0)$, $(0.6, 0.0)$, $(0.5, -0.5)$, $(-0.0, -0.5)$, $(-0.6, 1.0)$, $(-1.0, 1.0)$, and $(-1.0, -1.0)$. Although this paper focuses on these three domains in the experiments, the proposed FEONet has the potential to solve PDEs using FEM triangulation for more complex domains such as airfoils where the classical FEM works.

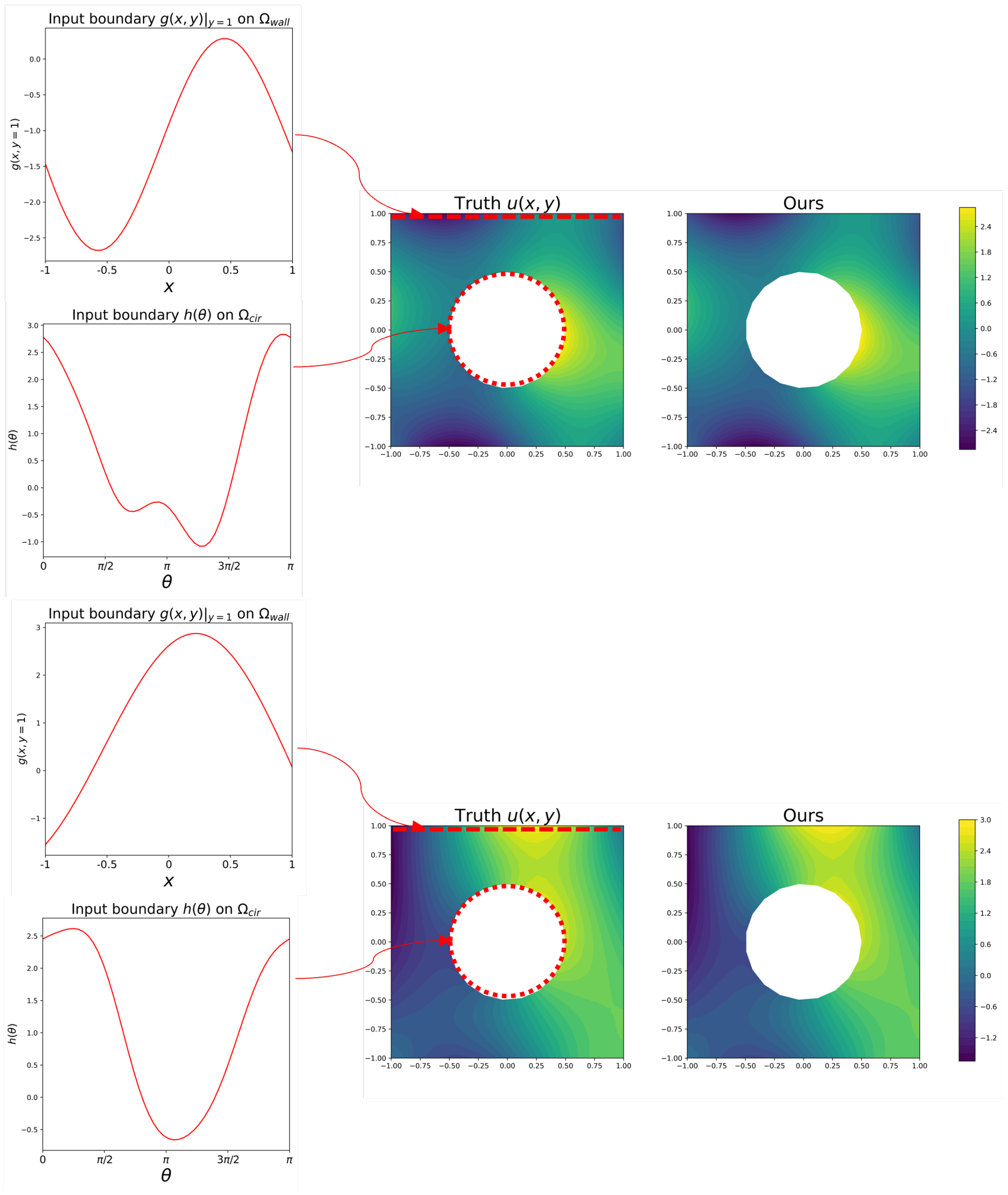


Figure 11: Solution profiles predicted by the trained FEONet for PDE (41) obtained from the boundary condition input. Note that the boundary condition $g(x, y)$ for the region with a circle hole inside Ω_{cir} is represented as $h(\theta)$ in polar coordinates.

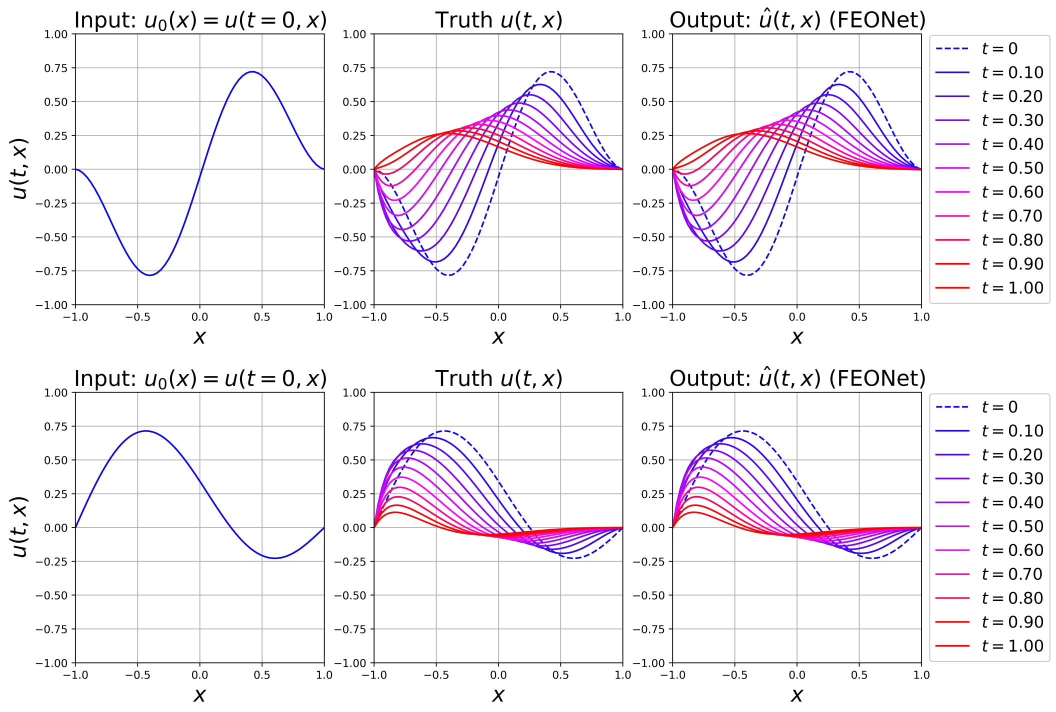


Figure 12: Solution profiles predicted by the trained FEONet for PDE (42) obtained from the initial condition $u(t=0,x)$ as two distinct input functions.

D.6 Burgers' equation

For the variational loss function for the Burgers' equation described in (15), we identified the uu_x term as $(\frac{1}{2}u^2)_x$. Therefore, we induce the $B[\widehat{u}_h(x;\boldsymbol{\omega}_m), \phi_i(x)]$ in the loss function (9) as

$$\begin{aligned}
B[\widehat{u}_h(x;\boldsymbol{\omega}_m), \phi_i(x)] &= \varepsilon \int_D (\widehat{u}_h)_x(\phi_i)_x \,\mathrm{d}x + \int_D \widehat{u}_h(\widehat{u}_h)_x\phi_i \,\mathrm{d}x \\
&= \varepsilon \int_D (\widehat{u}_h)_x(\phi_i)_x \,\mathrm{d}x - \int_D \frac{1}{2}\widehat{u}_h^2(\phi_i)_x \,\mathrm{d}x \\
&= \underbrace{\varepsilon \sum_{k=1}^{N(h)} \widehat{\alpha}_k \int_D (\phi_k)_x(\phi_i)_x \,\mathrm{d}x}_{\text{(I)}} - \underbrace{\int_D \frac{1}{2}\left(\sum_{k=1}^{N(h)} \widehat{\alpha}_k\phi_k\right)^2 (\phi_i)_x \,\mathrm{d}x}_{\text{(II)}}.
\end{aligned}$$

The main difference from other PDE problems is the second term (II) on the right-hand side. Since the term (II) cannot be expressed linearly with respect to $\widehat{\alpha}_k$, an iterative method should be used for the linearization when using the classical FEM. However, the loss function can be directly computed for the FEONet, and once the training is completed, it can make a real-time solution prediction without any need for iteration methods whenever the forcing function is given. In other words, the FEONet is able to learn the coefficients $\{\widehat{\alpha}_k\}_{k=1}^{N(h)}$ effectively even for the nonlinear Burgers' equation.

D.7 Singular perturbation problem

Solving convection-dominated singularly perturbed problems in numerical analysis is a challenging task as the small diffusive parameter $\varepsilon > 0$ generates a sharp transition inside thin layers, requiring special treatment. The transition of the boundary layer for (16) with respect to different viscosity parameters $\varepsilon > 0$ is provided in Figure 13. ML approaches suffer from the boundary layer problem since neural networks have a smooth prior (see Figure 5). When solving a single singularly perturbed

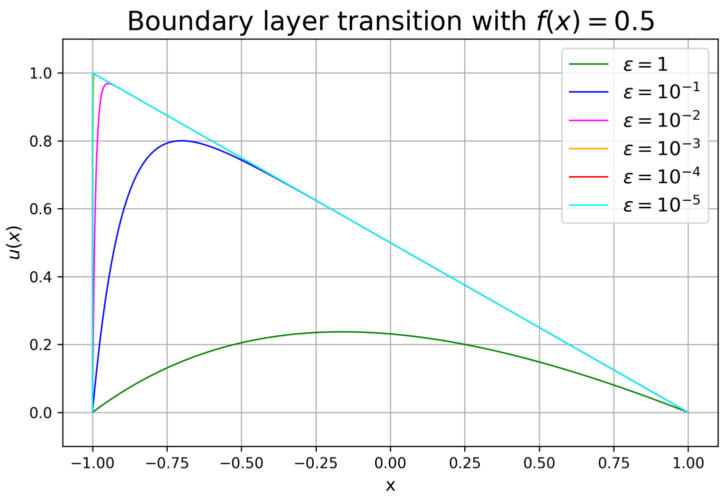


Figure 13: Boundary layer transition for (16) with respect to different viscosity parameters $\varepsilon > 0$ is displayed. As ε decreases, the thickness of the boundary layer gets smaller.

PDE using the PINN, the training using residual loss becomes unstable due to a very small diffusion coefficient, resulting in poor learning. Figure 14 shows different solutions predicted by five different PINN training for a single PDE in 5×10^5 iterations.

Our FEONet overcomes this limitation by using an additional basis ϕ_{cor}, as defined in (17). We obtain the corrector ϕ_{cor} by deriving the asymptotic equation with the formal limit equation assuming $\varepsilon = 0$, as described in Hong & Jung (2018). Figure 15 shows the shape of the boundary layer corrector for $\varepsilon = 10^{-5}$.

As a paradigm example to derive ϕ_{cor}, we consider the following singularly perturbed convection-diffusion equation:

$$\begin{cases} -\varepsilon u^{\varepsilon}_{xx} - u^{\varepsilon}_{x} = f(x), & 0 < x < 1, \\ u^{\varepsilon} = 0, & x = 0, 1. \end{cases} \tag{45}$$

Our main objective is to construct the corrector basis for the singular perturbation problem. To find the corrector, we formally replace ε by 0 in $(45)_1$, whose limit problem of (45) at $\varepsilon = 0$ is

$$\begin{cases} -u^{0}_{x} = f(x), & 0 < x < 1, \\ u^{0} = 0, & x = 1. \end{cases} \tag{46}$$

We impose the inflow boundary condition at $x = 1$ for u^0, leading to the formal limit of u^0 in the following form:

$$u^0(x) = \int_0^1 f \,\mathrm{d}x. \tag{47}$$

Performing matching asymptotics for equation (45), we observe the presence of a boundary layer of size ϵ near the outflow boundary at $x = 0$. Consequently, we determine the asymptotic equation, considering small ϵ, for the corrector φ that approximates the difference $u^{\epsilon} - u^0$. This equation can be written as follows:

$$\begin{cases} -\varepsilon \varphi_{xx} - \varphi_{x} = 0, & 0 < x < 1, \\ \varphi = -u^{0}, & x = 0. \end{cases} \tag{48}$$

It is well-known that the corrector φ is given in the form

$$\varphi(x) = -u^0(0)\, e^{-x/\varepsilon} + \text{e.s.t.}, \tag{49}$$

where the e.s.t. stands for an exponentially small term with respect to the small perturbation parameter $\varepsilon > 0$. To obtain the implementable corrector basis, we modify equation (49) by incorporating the boundary condition.

Performing the conventional energy estimates on the difference $u^{\varepsilon} - (u^0 + \varphi)$, we notice that

$$\begin{aligned} &\|u^{\varepsilon} - (u^0 + \varphi)\|_{L^2((0,1))} \leq \kappa\varepsilon, \\ &\|u^{\varepsilon} - u^0\|_{L^2((0,1))} \leq \kappa\varepsilon^{\frac{1}{2}}, \end{aligned} \tag{50}$$

for a constant $\kappa > 0$ independent of ε. The convergence results indicate two important observations. The convergence results highlight two significant observations. Firstly, the diffusive solution u^{ϵ} converges to the limit solution u^0 as ϵ approaches 0, with the convergence rate proportional to ϵ. Secondly, based on these convergence results, we can deduce that the corrector φ effectively captures the singular behavior of u^{ϵ} at small diffusivity ϵ. This implies that the diffusive solution u^{ϵ} can be decomposed into the sum of a rapidly decaying component represented by φ and a slowly varying component represented by u^0.

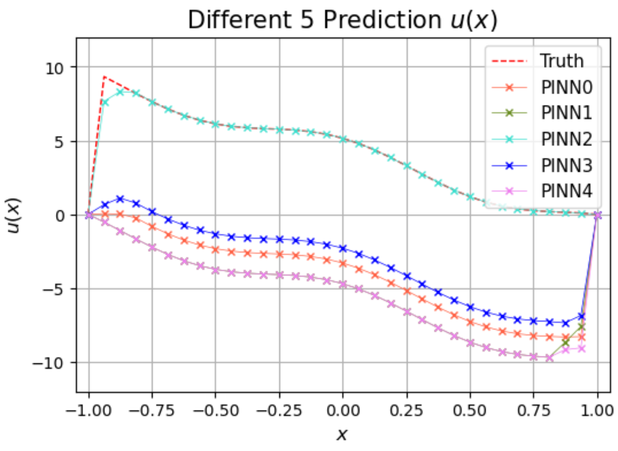


Figure 14: The solution profiles of five different experiments solving the singular perturbation problem using the PINN.

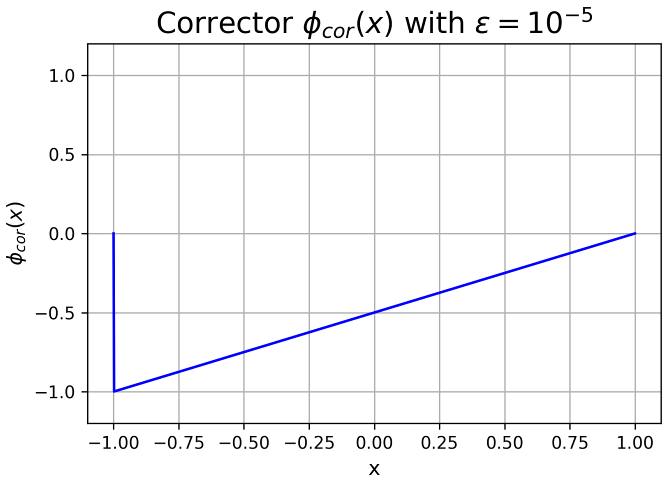


Figure 15: The profile of the boundary layer corrector which is used as the additional basis function for the singular perturbation problem.

D.8 Plots on convergence rates

Let us make some further comments on the experimental details for Figure 6. For 1D case, we consider the convection-diffusion equation

$$\begin{aligned} -\varepsilon u_{xx} + b u_x &= f(x), \quad x \in D = [-1, 1], \\ u(x) &= 0 \quad , \quad x \in \partial D, \end{aligned} \tag{51}$$

where $\varepsilon = 0.1$ and $b = -1$. For the 2D case, we consider the following equation

$$\begin{aligned} -\varepsilon \Delta u + \mathbf{v} \cdot \nabla u &= f(x, y), \quad (x, y) \in D = [-1, 1]^2, \\ u(x, y) &= 0, \qquad\quad (x, y) \in \partial D, \end{aligned} \tag{52}$$

where $\varepsilon = 0.1$ and $\mathbf{v} = (-1, 0)$. The FEONet was used to approximate the solution of these two equations using P1 and P2 nodal basis functions, and the experiments were conducted with varying domain triangulation to have different numbers of elements to observe the convergence rate. As shown in Figure 6, the observed convergence rate of the FEONet shows the convergence rates close to 2 and 3 for P1 and P2 approximation respectively, which are the theoretical results for the classical FEM. Since the precision scale of the neural network we used here is about 10^{-2}, we cannot observe the exact trends in rates below this level of error. We expect to see the same trend even at lower errors if we use larger scale and more advanced models than the CNN structure.

D.9 Additional plot on generalization error

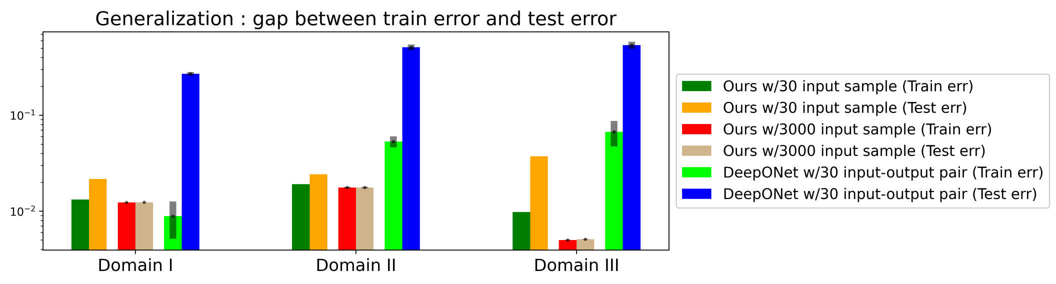


Figure 16: Rel. L^2 train and test error comparison between the FEONet with 30/3000 input samples (w/o labeled data) and the DeepONet (supervised) with 30 train-test data pair.

The FEONet is a model capable of learning the operator without the need for paired input-output training data. What we need for the training is the generation of random input samples, which only imposes negligible computational cost. Therefore, the number of input function samples for the FEONet training can be chosen arbitrarily. In the experiments performed in Section 4, we generated 3000 random input function samples to train the FEONet. On the other hand, the DeepONet requires paired input-output training data for supervised learning which causes significant computational costs to prepare. In this paper, we have focused our experiments on considering the DeepONet trained with 30, 300, and 3000 input-output data pairs. This allowed us to highlight the FEONet's ability to achieve a certain level of accuracy without relying on data pairs. In Figure 7, We compared the generalization capability of both models. We used 3000 input samples to train the FEONet and 30 input-output data pairs to train the DeepONet. Even with the 30 data pairs, there occurred a lot of computational costs to gain the data. Although it is predictable that increasing the number of input-output data pairs for the DeepONet would narrow the gap between train and test errors, preparing a large amount of data causes significant computational costs.

In addition to the experiments for Figure 7, in order to reduce the possibility of reader's confusion, we also present the results comparing generalization capability in Figure 16, when the FEONet is trained only with 30 random input samples (green and yellow bar) (which should be distinguished from 30 input-out data pair for the supervised learning). Figure 16 depicts a comparison of train and test errors between the FEONet, trained through an unsupervised approach with 30 and 3000 input samples, and the DeepONet, trained through supervised learning with 30 input-output data pairs. Notably, the FEONet shows superior generalization compared to the DeepONet, even without requiring input-output paired data. Remarkably, utilizing only 30 input samples for training, the FEONet achieves significantly better generalization and exhibits lower test errors than that of the DeepONet.

E LIMITATIONS

Despite the promising results presented in this study, there are still several technical questions that remain unresolved and necessitate further investigation. One such question is determining the optimal features embedding or network architecture of the FEONet for a given parametric governing law. Understanding the most effective approach for feature representation and network design is crucial for achieving optimal performance and generalization in the FEONet framework. Addressing this question would contribute to advancing the understanding and applicability of the FEONet methodology. Once this issue is resolved, our approach holds the potential for addressing challenging problems, including the Navier-Stokes equations, which represent one of the most fascinating nonlinear problems in the numerical solution of partial differential equations. By applying the FEONet to such complex scenarios as a future direction, we can make significant advancements in the field and uncover new insights into the behavior and dynamics of these systems.

The issue of scalability is a prominent concern not just in scientific machine learning but also in traditional scientific computing. In our case, for large-scale and intricate problems, there exists the potential for condition number challenges, given that our loss function is grounded in finite element matrices. To mitigate this concern, it is possible to utilize established preconditioning techniques. These aim to impose a reduced condition number on the finite element matrices, much in the way they are applied in conventional numerical methods. It's pertinent to highlight that the act of multiplying the loss function by a matrix (or a preconditioner) equates to the addition of an extra layer in neural networks. Therefore, the proposed framework remains compatible with the preconditioned loss function. Interestingly, this might be viewed as another strength of our numerical analysis-based approach.