

Finite Element Operator Network for Solving Elliptic-type Parametric PDEs

Youngjoon Hong^{*}, Seungchan Ko[†], and Jaeyong Lee[‡]

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 performed various experiments on several benchmark problems and confirmed that our approach has demonstrated excellent performance across various settings and environments, proving its versatility in terms of accuracy, generalization, and computational flexibility. While our method is not meshless, the 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.

Keywords: scientific machine learning, finite element methods, physics-informed operator learning, parametric PDEs, boundary layer

AMS Classification: 65M60, 65N30, 68T20, 68U07

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 [12, 6]. From fluid dynamics, heat transfer, to electromagnetic fields, PDEs provide a mathematical framework that allows us to comprehend and model these complex systems [17, 35]. Their significance extends beyond the realm of physics and finds applications in fields such as finance, economics, and computer graphics [50, 43]. 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 [9, 3]. Various techniques, such as finite difference, finite element, and finite volume methods, are utilized to develop these numerical methods [29, 51, 31]. 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

^{*}Department of Mathematical Sciences, Seoul National University, Seoul, Republic of Korea.
Email: hongyj@kaist.ac.kr

[†]Department of Mathematics, Inha University, Incheon, Republic of Korea.
Email: scko@inha.ac.kr

[‡]Department of AI, Chung-Ang University, Seoul, Republic of Korea.
Email: jaeyong@cau.ac.kr

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 [56, 22]. However, 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 [28]. 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 [39, 23, 55, 41, 47]. Furthermore, this has led to active research on modified models of PINN by leveraging the advantages of it [19, 1, 44]. Despite their advantages, these approaches come with limitations. One prominent drawback of the PINN methods is that they are trained on a single instance of input data such as 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.

In this regard, operator networks, an emerging field in research, address challenges by employing data-driven approaches to comprehend mathematical operators that govern physical systems. This is particularly relevant in solving parametric PDEs, as referenced in multiple studies [37, 32, 53, 7, 34, 38, 42, 16, 30, 44]. In particular, based upon the Universal Approximation Theorem for operators, [37] introduced the Deep Operator Network (DeepONet) architecture. Moreover, the authors in [38] proposed an extension of the DeepONet model, called POD-DeepONet, which replaces a part of the DeepONet architecture with a prefixed basis obtained through proper orthogonal decomposition (POD) on the training data. These methods enable us to make fast predictions of solutions in real-time whenever the given PDE data changes, offering a new framework for solving parametric PDEs. 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. To overcome these limitations, the integration of PINNs and operator learning has led to the development of new methods, such as the Physics-Informed Neural Operator (PINO) [33] and Physics-Informed DeepONet (PIDeepONet) [53]. These methods strive to leverage the strengths of both PINNs and operator learning by incorporating physical equations into the neural operator's loss function. However, they exhibit certain drawbacks, such as relatively low accuracy in handling complex geometries and difficulties in effectively addressing stiff problems like boundary layer issues, which are critical in real-world applications. Moreover, PIDeepONet tends to exhibit high generalization errors when lacking sufficient input samples. Additionally, employing neural networks as the solution space presents challenges in imposing boundary values, potentially leading to less accurate solutions.

To address the aforementioned limitations, this paper proposes a novel operator network for solving diverse parametric PDEs without reliance on training data. More precisely, we introduce the Finite Element Operator Network (FEONet), which utilizes finite element approximation to learn the solution operator. This approach eliminates the need for solution datasets in addressing 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 so that

$$u_h(\mathbf{x}) = \sum \alpha_k \phi_k(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d, \tag{1.1}$$

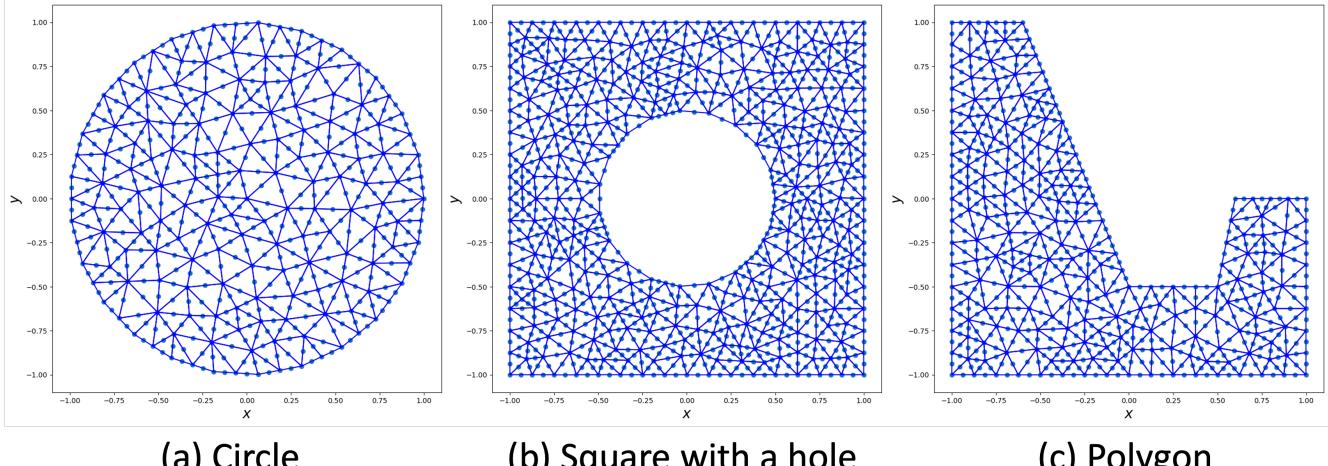


Figure 1: Domain triangulation for three different complex geometries.

where u_h is the FEM solution for the given PDEs. Motivated by (1.1), the FEONet can predict numerical solutions of the PDEs when given the initial conditions, external forcing functions, PDE coefficients, or boundary conditions as inputs. 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, motivated by the classical FEM [11, 8]. This allows the FEONet to accurately approximate the solutions of PDEs, while also ensuring that the boundary conditions are exactly satisfied. This approach involves inferring coefficients, denoted as $\hat{\alpha}_k$, which are subsequently utilized to construct the linear combination $\sum \hat{\alpha}_k \phi_k$ to approximate the solution of PDEs. Inheriting the capability of the FEM handling boundary conditions, the predicted solution will also satisfy the exact boundary condition. Note further that since we reconstruct the solution as a linear combination of the coefficients predicted by neural networks and the basis function, the FEONet scheme generates a solution as a function defined on the whole domain, not only on particular points in the domain. Additionally, since we have adopted the FEM as a baseline framework, we can expect that our proposed method is robust, reliable, and accurate. It is noteworthy that due to the intrinsic structure of the proposed FEONet scheme, we do not need any paired input-output training data and hence the model can be trained in an unsupervised manner.

Another key contribution of this study is the introduction of a novel learning architecture designed to accurately solve convection-dominated singularly perturbed problems, which are characterized by strong boundary layer phenomena. These types of problems present significant challenges for traditional machine learning and numerical methods, primarily due to the sharp transitions within thin layers caused by a small diffusive parameter. In particular, this is challenging in the realm of learning theory, where neural networks typically assume a smooth prior, making them less equipped to handle such abrupt transitions effectively. By adapting theory-guided methods [10], the network effectively captures the behavior of boundary layers, an area that has been underexplored in recent machine-learning approaches.

The structure of this paper is outlined as follows: Section 2 provides a detailed elaboration on the proposed numerical scheme as well as the theoretical convergence results. Subsequently, Section 3 presents the experimental results, demonstrating the computational flexibility of FEONet in both 1D and 2D settings across various PDE problems. Finally, Section 4 summarizes this work.

2 Finite element operator network (FEONet)

In this section, we present our proposed method, the FEONet, by demonstrating how neural network techniques can be 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 $\mathbf{a}(\mathbf{x})$ and $\varepsilon > 0$,

$$-\varepsilon \operatorname{div}(\mathbf{a}(\mathbf{x}) \nabla u) + \mathcal{F}(u) = f \quad \text{in } D. \quad (2.1)$$

Here \mathcal{F} can be either linear or nonlinear, and both cases will be covered in the experiments performed in Section 3. 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.

In addition to the explanation for the proposed method, we will also describe the analytic framework of the FEONet, where we can discuss the convergence analysis of the method.

2.1 Model description

We begin with the following variational formulation of the given PDE (2.1): find $u \in V$ satisfying

$$B[u, v] := \varepsilon \int_D \mathbf{a}(\mathbf{x}) \nabla u \cdot \nabla v \, d\mathbf{x} + \int_D \mathcal{F}(u) v \, d\mathbf{x} = \int_D f v \, d\mathbf{x} =: \ell(v) \quad \text{for all } v \in V, \quad (2.2)$$

for a suitable function space V . In the theory of FEM, the first step is to define a *triangulation* of the given domain $\overline{D} \subset \mathbb{R}^d$, $d = \{1, 2, 3\}$. We shall assume that $\{\mathcal{T}_h\}_{h>0}$ is a collection of conforming shape-regular triangulations of $\overline{D} \subset \mathbb{R}^d$, consisting of d -dimensional simplices K . The finite element parameter $h > 0$ signifies the maximum mesh size of \mathcal{T}_h . 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 nodal basis $\{\phi_j\}$ for V_h , defined by $\phi_j(\mathbf{x}_i) = \delta_{ij}$.

Then the Galerkin approximation is as follows: find $u_h \in V_h$ satisfying

$$B[u_h, v_h] = \ell(v_h) \quad \text{for all } v_h \in V_h. \quad (2.3)$$

If we write the finite element solution

$$u_h = \sum_{k=1}^{N(h)} \alpha_k \phi_k, \quad \alpha_i \in \mathbb{R}, \quad (2.4)$$

the Galerkin approximation (2.3) is transformed into the linear algebraic system

$$A\alpha = F \quad \text{with } A_{ik} := B[\phi_k, \phi_i] \text{ and } F_i := \ell(\phi_i), \quad (2.5)$$

where the matrix $A \in \mathbb{R}^{N(h) \times N(h)}$ is invertible provided that the given PDEs have a suitable structure. Therefore, we can determine the coefficients $\{\alpha_k\}_{k=1}^{N(h)}$ by solving the system of linear equations (2.5), and consequently, produce the approximate solution $u_h \in V_h$ as defined in (2.4).

Now, in this setting let us describe the method we propose in this paper. 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, Section 3.1.4 where extensive experiments were performed addressing various types of input data. For this purpose, the external forcing terms are parametrized by the random parameter ω contained in the (possibly high-dimensional) parameter space Ω . For each realization $f(\mathbf{x}; \omega)$ (and hence for each load vector $F(\omega)$ defined in (2.5)), instead of computing the coefficients from the linear algebraic system (2.5), we approximate the coefficients α by a deep neural network. More precisely, these input features

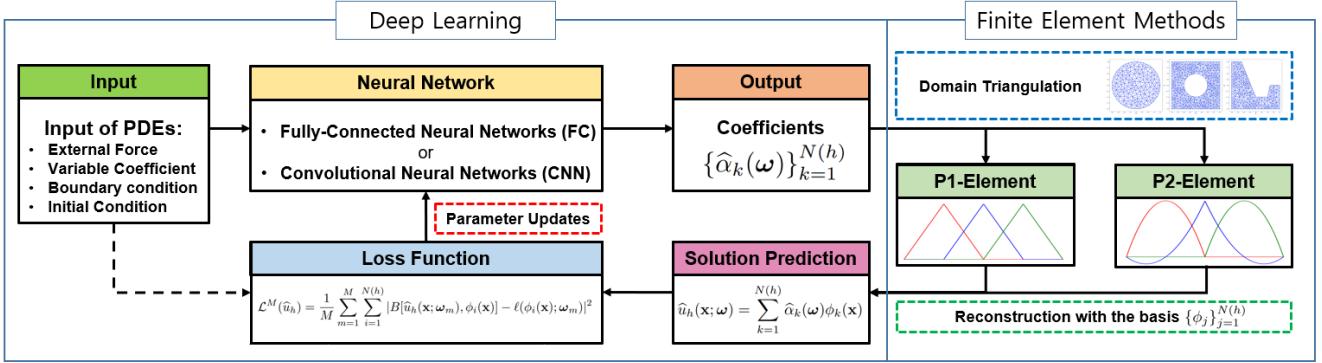


Figure 2: Schematic diagram of the Finite Element Operator Network (FEONet).

representing external forces pass through the deep neural network and the coefficients $\{\hat{\alpha}_k\}$ are generated as an output of the neural network. We then reconstruct the solution

$$\hat{u}_h(\mathbf{x}; \omega) = \sum_{k=1}^{N(h)} \hat{\alpha}_k(\omega) \phi_k(\mathbf{x}). \quad (2.6)$$

For the training, we use a residual of the variational formulation (2.3) and define the loss function by summing over all basis functions: for randomly chosen parameters $\omega_1, \dots, \omega_M \in \Omega$,

$$\mathcal{L}^M(\hat{u}_h) = \frac{1}{M} \sum_{m=1}^M \sum_{i=1}^{N(h)} |B[\hat{u}_h(\mathbf{x}; \omega_m), \phi_i(\mathbf{x})] - \ell(\phi_i(\mathbf{x}); \omega_m)|^2. \quad (2.7)$$

Note that our proposed loss function is based on the variational form of the given problem, and the parameters are updated in the direction that minimizes the corresponding loss function (2.7) defined above. In this paper, we assume that it is possible to find the exact minimizers, allowing us to ignore any errors arising from the optimization process. It is worth noting that loss functions based on the variational form have also been considered in previous work related to PINNs [24, 25]. However, unlike our approach, those methods were designed for single-instance predictions and did not utilize the P1 basis elements that are integral to our operator network. Instead, they employed different types of test functions. While [19] also proposes an operator-learning method and uses the variational form of governing equations as a loss function along with input-output data pairs, there is a difference from our FEONet approach, which is completely free from the need for input-output data pairs. The detailed algorithm for training FEONet is provided in the pseudo-code below.

For each training epoch, once the neural network parameters are updated in the direction of minimizing the loss function (2.7), the external force $f(\mathbf{x}; \omega)$ 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 2.

Remark 2.1. To avoid confusion for readers, here we provide further clarification: in this paper, the term “training data” refers to the labeled data, specifically the pairs of pre-computed PDE data and their corresponding solutions. The term “input sample” is used to denote the randomly extracted points from the parametric domain. In this case, since they don’t require labeling, there is no computational cost associated with the sampling process. In summary, for the training of FEONet, only input samples are required and the model can be trained without the need for pre-computed training data.

Remark 2.2. There are several reasons why we used the variational residual (2.7) as a loss function instead of other types of loss functions, e.g., the physics-informed loss function. Since our method is

Algorithm 1 FEONet Training Procedure

Input: Training samples $f(\mathbf{x}; \boldsymbol{\omega}_m)$ for $m = 1, \dots, M$;

Output: The coefficients $\hat{\alpha}(\boldsymbol{\omega}_m) = [\hat{\alpha}_k(\boldsymbol{\omega}_m)]_{k=1,\dots,N(h)}$ for $m = 1, \dots, M$. Therefore, the predicted solution is $\hat{u}_h(\mathbf{x}; \boldsymbol{\omega}_m) = \sum_{k=1}^{N(h)} \hat{\alpha}_k(\boldsymbol{\omega}_m) \phi_k(\mathbf{x})$;

- 1: Pre-compute A ($A_{ik} = B[\phi_k, \phi_i]$) and $F(\boldsymbol{\omega}_m)$ for $m = 1, \dots, M$ using basis functions $\{\phi_i\}_{i=1}^{N(h)}$ based on domain triangulation $\{\mathcal{T}_h\}$;
- 2: **for** $n = 1, 2, \dots, N_{\text{epochs}}$ **do**
- 3: $\mathcal{L}^M \leftarrow 0$;
- 4: **for** $m = 1, \dots, M$ **do**
- 5: Compute $\hat{\alpha}(\boldsymbol{\omega}_m)$ using FEONet with input $f(\mathbf{x}; \boldsymbol{\omega}_m)$;
- 6: $\mathcal{L}^M \leftarrow \mathcal{L}^M + |A\hat{\alpha}(\boldsymbol{\omega}_m) - F(\boldsymbol{\omega}_m)|^2$;
- 7: **end for**
- 8: Update FEONet parameters by minimizing \mathcal{L}^M ;
- 9: **end for**

inherited from the classical FEM, in order to use the well-known techniques from numerical analysis, it is more “natural” to consider the loss function we chose. As a consequence, we are able to successfully obtain the theoretical convergence result from the viewpoint of the FEM, where the analysis mainly depends on the form of the loss function. To be more specific, in appendix A.1 and our subsequent paper [20], we prove the best approximation property of the FEONet, providing the theoretical evidence that we chose the right loss function. In summary, we have adopted the variational residual as a loss function from the theoretical perspective. On the other hand, there are also some advantages of using this weak formulation as a loss function. For example, we can reduce the computation error for the derivative like the variational physics-informed neural networks [24, 25].

2.2 Convergence analysis of FEONet

In this section, we address the convergence result of the FEONet which provides theoretical justification for the proposed method. As outlined in the previous subsection, the proposed method is based on the FEM, and the finite element approximation u_h in (2.4) can be considered as an intermediate solution between the true solution u of (2.2) and the approximate solution \hat{u}_h predicted by the FEONet.

For simplicity, we will focus on self-adjoint equations with homogeneous Dirichlet boundary conditions:

$$\begin{aligned} -\operatorname{div}(\mathbf{a}(\mathbf{x}) \nabla u) + c(\mathbf{x}) u &= f(\mathbf{x}) \quad \text{in } D, \\ u &= 0 \quad \text{on } \partial D, \end{aligned} \tag{2.8}$$

where the coefficient $\mathbf{a}(\mathbf{x})$ is uniformly elliptic and $c(\mathbf{x}) \geq 0$. In the present paper, we focus on the case when the finite element parameter $h > 0$ is fixed and deal with the convergence of the FEONet prediction to the finite element solution u_h . The analysis for general equations that cannot be covered by the form (2.8) and the theoretical analysis addressing the role of $h > 0$ in the error estimate are of independent interest and will be investigated in the forthcoming paper.

Analytic framework 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

$$f(\mathbf{x}; \boldsymbol{\omega}) \in C(\Omega; L^1(D)) \quad \text{and} \quad \sup_{\boldsymbol{\omega} \in \Omega} \int_D |f(\mathbf{x}; \boldsymbol{\omega})| d\mathbf{x} < C \tag{2.9}$$

for some constant $C > 0$. For each $\omega \in \Omega$, the external force $f(\mathbf{x}; \omega)$ is determined, and the corresponding weak solution is denoted by $u(\mathbf{x}; \omega)$ which satisfies the following weak formulation:

$$B[u, v] := \int_D [\mathbf{a}(\mathbf{x}) \nabla u \cdot \nabla v + c(\mathbf{x}) uv] d\mathbf{x} = \int_D f(\mathbf{x}) v d\mathbf{x} =: \ell(v) \quad \forall v \in H_0^1(D). \quad (2.10)$$

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

$$B[u_h, v_h] = \ell(v_h) \quad \forall v_h \in V_h. \quad (2.11)$$

We shall write

$$u_h(\mathbf{x}, \omega) = \sum_{k=1}^{N(h)} \alpha_k^*(\omega) \phi_k(\mathbf{x}), \quad (2.12)$$

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

$$A\alpha^* = F, \quad (2.13)$$

with

$$A_{ik} = B[\phi_k, \phi_i] \quad \text{and} \quad F_i = \ell(\phi_i). \quad (2.14)$$

In other words, u_h in (2.12) is the finite element approximation of the solution to the equation (2.8), as described in Section 2.1. It is noteworthy that instead of the equations (2.13), α^* can also be characterized in the following way:

$$\alpha^* = \arg \min_{\alpha \in C(\Omega, \mathbb{R}^{N(h)})} \mathcal{L}(\alpha), \quad (2.15)$$

where \mathcal{L} is the population loss

$$\mathcal{L}(\alpha) = \mathbb{E}_{\omega \sim \mathbb{P}_\Omega} \left[\sum_{i=1}^{N(h)} |B[\hat{u}(\omega), \phi_i] - \ell(\phi_i; (\omega))|^2 \right] = \|A\alpha(\omega) - F(\omega)\|_{L^2(\Omega)}^2. \quad (2.16)$$

Next, we shall define a class of neural networks, that plays the role of an approximator for coefficients. For each $L \in \mathbb{N}$, we write an L -layer neural network by a function $f^L(x) : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$, defined by

$$f^1(x) = W^1 x + b^1 \quad \text{and} \quad f^\ell(x) = W^\ell \sigma(f^{\ell-1}(x)) + b^\ell \quad \text{for } 2 \leq \ell \leq L, \quad (2.17)$$

where $W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $b^\ell \in \mathbb{R}^{n_\ell}$ are the weight and the bias respectively for the ℓ -th layer, and $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is a nonlinear activation function. We denote the architecture by the vector $\vec{n} = (n_0, \dots, n_L)$, the class of neural network parameters by $\theta := \theta_{\vec{n}} = \{(W^1, b^1), \dots, (W^L, b^L)\}$, and its realization by $\mathcal{R}[\theta](x)$. For a given neural network architecture \vec{n} , the family of all possible parameters is denoted by

$$\Theta_{\vec{n}} = \left\{ \{(W^\ell, b^\ell)\}_{\ell=1}^L : W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, b^\ell \in \mathbb{R}^{n_\ell} \right\}. \quad (2.18)$$

For two neural networks θ_i , $i = 1, 2$, with architectures $\vec{n}_i = (n_0^{(i)}, \dots, n_{L_i}^{(i)})$, we write $\vec{n}_1 \subset \vec{n}_2$ if for any $\theta_1 \in \Theta_{\vec{n}_1}$, there exists $\theta_2 \in \Theta_{\vec{n}_2}$ satisfying $\mathcal{R}[\theta_1](x) = \mathcal{R}[\theta_2](x)$ for all $x \in \mathbb{R}^{n_0}$.

Now let us assume that there exists a sequence of neural network architectures $\{\vec{n}_n\}_{n \geq 1}$ satisfying $\vec{n}_n \subset \vec{n}_{n+1}$ for all $n \in \mathbb{N}$. We define the associated collection of neural networks by

$$\mathcal{N}_n^* = \{\mathcal{R}[\theta] : \theta \in \Theta_{\vec{n}_n}\}. \quad (2.19)$$

It is straightforward to verify that $\mathcal{N}_n^* \subset \mathcal{N}_{n+1}^*$ for any $n \in \mathbb{N}$. We start with the following theorem. This is known as the *universal approximation theorem*, and known to hold in various scenarios [13, 21, 45].

Theorem 2.3. Let K be a compact set in \mathbb{R}^d and assume that $g \in C(K, \mathbb{R}^D)$. Then there holds

$$\lim_{n \rightarrow \infty} \inf_{\tilde{g} \in \mathcal{N}_n^*} \|\tilde{g} - g\|_{C(K)} = 0. \quad (2.20)$$

In the present paper, the target function to be approximated by neural networks is $\alpha^*(\boldsymbol{\omega})$. Due to the assumption (2.9), we can verify that $\alpha^*(\boldsymbol{\omega})$ is continuous and bounded. This motivates us to deal with the family of bounded neural networks as an ansatz space. For this, let us write $\sup_{\boldsymbol{\omega} \in \Omega} |\alpha^*(\boldsymbol{\omega})| < C_\alpha$, and assume that σ^* is a bounded activation function with continuous inverse (e.g., sigmoid or tanh activation functions) with $\sigma_- := \inf_{x \in \mathbb{R}} \sigma^*(x)$ and $\sigma_+ := \sup_{x \in \mathbb{R}} \sigma^*(x)$. We then define a function $h : [\sigma_-, \sigma_+] \rightarrow [-C_\alpha, C_\alpha]$ satisfying $h(\sigma_-) = -C_\alpha$, $h(\sigma_+) = C_\alpha$. By the assumptions, h also has a continuous inverse. With the above notations, we consider the following family of neural networks:

$$\mathcal{N}_n = \{h \circ \sigma^*(g_n) : g_n \in \mathcal{N}_n^*\}, \quad (2.21)$$

where we can see that

$$\|g\|_{C(\Omega)} \leq C_\alpha, \quad \forall g \in \bigcup_{n \in \mathbb{N}} \mathcal{N}_n. \quad (2.22)$$

In the analysis, we will use the following version of the universal approximation theorem for bounded neural networks, which is quoted from [26].

Theorem 2.4. Let Ω be a compact set and $\alpha^* : \Omega \rightarrow \mathbb{R}^{N-1}$ is a continuous function defined in (2.13). Then we have

$$\lim_{n \rightarrow \infty} \inf_{\tilde{g} \in \mathcal{N}_n} \|\tilde{g} - \alpha^*\|_{C(\Omega)} = 0. \quad (2.23)$$

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

$$\hat{\alpha}(n) = \arg \min_{\alpha \in \mathcal{N}_n} \mathcal{L}(\alpha), \quad (2.24)$$

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

$$u_{h,n}(\mathbf{x}; \boldsymbol{\omega}) = \sum_{k=1}^{N(h)} \hat{\alpha}(n)_k(\boldsymbol{\omega}) \phi_k(\mathbf{x}). \quad (2.25)$$

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:

$$\hat{\alpha}(n, M) = \arg \min_{\alpha \in \mathcal{N}_n} \mathcal{L}^M(\alpha), \quad (2.26)$$

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

$$\mathcal{L}^M(\alpha) = \frac{|\Omega|}{M} \sum_{m=1}^M \sum_{i=1}^{N(h)} |B[\hat{u}(\boldsymbol{\omega}_m), \phi_i] - \ell(\phi_i; (\boldsymbol{\omega}_m))|^2 = \frac{|\Omega|}{M} \sum_{m=1}^M |A\alpha(\boldsymbol{\omega}_m) - F(\boldsymbol{\omega}_m)|^2, \quad (2.27)$$

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

$$u_{h,n,M}(\mathbf{x}; \boldsymbol{\omega}) = \sum_{k=1}^{N(h)} \hat{\alpha}(n, M)_k(\boldsymbol{\omega}) \phi_k(\mathbf{x}), \quad (2.28)$$

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

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

$$\|u - u_{h,n,M}\|_{L^2(\Omega; L^2(D))} \rightarrow 0 \quad \text{as } h \rightarrow 0 \text{ and } n, M \rightarrow \infty. \quad (2.29)$$

The main error can be split into three parts:

$$u - u_{h,n,M} = (u - u_h) + (u_h - u_{h,n}) + (u_{h,n} - u_{h,n,M}). \quad (2.30)$$

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 classical theory of the FEM (e.g. [8, 11]) if $h > 0$ is sufficiently small, then we can expect to obtain a sufficiently small error. Therefore, in this paper, we shall assume that we have chosen suitable $h > 0$ which guarantees a sufficiently small error for the finite element approximation, and only focus on the convergence of $u_h - u_{h,n,M}$ with a fixed $h > 0$.

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, i.e., we will show that

$$u_h - u_{h,n,M} \rightarrow 0 \quad \text{as } n, M \rightarrow \infty.$$

Approximation error We first note from (2.16) and (2.27), that A defined in (2.12), (2.13) 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 [26].

Lemma 2.5. *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 $\mathbf{x} \in \mathbb{R}^{N(h)}$,*

$$\rho_{\min} |\mathbf{x}| \leq |A\mathbf{x}| \leq \rho_{\max} |\mathbf{x}|. \quad (2.31)$$

Since the equation under consideration (2.8) is self-adjoint, the corresponding bilinear form $B[\cdot, \cdot]$ defined in (2.10) is symmetric which automatically guarantees that the matrix A in our case is symmetric. Furthermore, due to the facts that the coefficient $\mathbf{a}(\cdot)$ 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 2.5 to the matrix A of our interest.

We are now ready to present our first convergence result regarding the approximation error. The proof is based on Lemma 2.5 and can be found in Appendix A.1.

Theorem 2.6. *Assume that (2.9) holds. Then we have that*

$$\|\alpha^* - \hat{\alpha}(n)\|_{L^2(\Omega)} \rightarrow 0 \quad \text{as } n \rightarrow \infty. \quad (2.32)$$

Generalization error In order to handle the generalization error, we first introduce the concept so-called *Rademacher complexity* [5].

Definition 2.7. 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 $i = 1, \dots, M$.

Note that the Rademacher complexity is the expectation of the maximum correlation between the random noise $(\varepsilon_1, \dots, \varepsilon_M)$ and the vector $(f(X_1), \dots, f(X_M))$, 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 [18, 52, 48].

Now we define the function class of interest

$$\mathcal{G}_n := \{|A\alpha - F|^2 : \alpha \in \mathcal{N}_n\}, \quad (2.33)$$

where the matrix A and the vector F are defined in (2.14). In the following theorem, note that we assume 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 [48, 14, 40]. For the proof of the theorem below, see Appendix A.2.

Theorem 2.8. Suppose that (2.9) holds, and we further assume that for any $n \in \mathbb{N}$, $\lim_{M \rightarrow \infty} R_M(\mathcal{G}_n) = 0$. Then we have with probability 1 that

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

Convergence of FEO Net From Theorem 2.6 and Theorem 2.8 combined with the triangular inequality, we have probability 1 over i.i.d. samples that

$$\lim_{n \rightarrow \infty} \lim_{M \rightarrow \infty} \|\alpha^* - \hat{\alpha}(n, M)\|_{L^2(\Omega)} = 0. \quad (2.34)$$

Based on the convergence of the approximate coefficients (2.34), let us now state the convergence theorem for the predicted solution, whose proof is presented in Appendix A.3.

Theorem 2.9. Suppose (2.9) holds, and assume that for all $n \in \mathbb{N}$, $R_M(\tilde{\mathcal{G}}_n) \rightarrow 0$ as $M \rightarrow \infty$, with $\tilde{\mathcal{G}}_n := \{|A\alpha - F|^2 : \alpha \in \mathcal{N}_n\}$. Then for given $h > 0$, we have with probability 1 over i.i.d. samples that

$$\lim_{n \rightarrow \infty} \lim_{M \rightarrow \infty} \|u_h - u_{h,n,M}\|_{L^2(\Omega; H^s(D))} = 0. \quad (2.35)$$

3 Numerical experiments

In this section, we demonstrate the experimental results of the FEO Net across various settings for PDE problems. We randomly generate 3000 input samples for external forces and train the FEO Net 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 (See Appendix B.1). While there are other operator learning-based methods, such as PINO [33], that can be trained without input-output data pairs, they often face limitations when dealing with domains of complex shapes.

Table 1: Mean Rel. L^2 test errors ($\times 10^{-2}$) with standard deviations ($\times 10^{-2}$) 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. (3.1), BC I (Dirichlet), BC II (Neumann) with Eq. (3.2), Eq I (second-order linear equation (3.3)) and Eq II (Burgers' equation (3.4)).

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

Table 2: Mean Rel. L^2 test errors ($\times 10^{-2}$) for the 1000 test set on diverse PDE problems. Coefficient (Variable coefficient as an input) with Eq. (3.3), Boundary (Boundary condition as an input) with Eq. (3.5), Initial (Time-dependent problem) with Eq. (3.6) and Vector-valued (Stokes problem) with Eq. (3.7).

Model (#Train data)	Coefficient	Boundary	Initial	Vector-valued
POD-DeepONet (supervised, w/10)	25.276	47.500	7446.077	-
POD-DeepONet (supervised, w/100)	10.536	28.471	230.288	-
POD-DeepONet (supervised, w/1000)	0.371	6.095	10.628	-
Ours (w/o labeled data)	0.612	0.430	0.218	5.141

Our primary comparison will be against DeepONet [37] and POD-DeepONet [38], with varying numbers of training data, as well as against PIDeepONet [53]. The FEONet is designed to learn the operator without requiring pre-computed training data, which significantly reduces computational overhead. The generation of random input samples, which are used for unsupervised learning, imposes negligible computational costs, offering flexibility in choosing the number of input function samples for training. In contrast, DeepONet (and POD-DeepONet) relies on supervised learning, requiring input-output data pairs. This necessitates pre-computing the output data for each input function sample, resulting in substantial computational demands. Consequently, it is challenging to directly compare the training times of FEONet with those of DeepONet (or POD-DeepONet). Note that training FEONet to solve the 2D Poisson equation in a domain with a circular hole, achieving a 1.3% relative error, took approximately 28 minutes and 36 seconds on an NVIDIA RTX A5000 24GB GPU. The triangulation of domains and the generation of the nodal basis for the FEM are performed using FEniCS [2, 36]. A detailed description of the model used in the experiments, as well as the comparison of computational time between FEM and operator learning models, can be found in Appendix B.2 and B.3.

In Section 3.1, we will solve PDE problems in various environments and settings to verify the computational flexibility of our method and confirm the flexibility of the proposed operator learning method by employing various types of input functions. In Section 3.2, we will conduct some further experiments on the parabolic problem and the Stokes problem. In Section 3.3, we will demonstrate that the FEONet framework can make an accurate solution prediction for the singular perturbation problems. Finally, in Section 3.4, additional experiments will be conducted to confirm the robustness and reliability of our method across various performance metrics. Throughout the experiments, we will cover both one-dimensional and two-dimensional cases, and we have appropriately organized them according to the objectives of each experiment.

3.1 Performance evaluation of FEONet in various scenarios

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

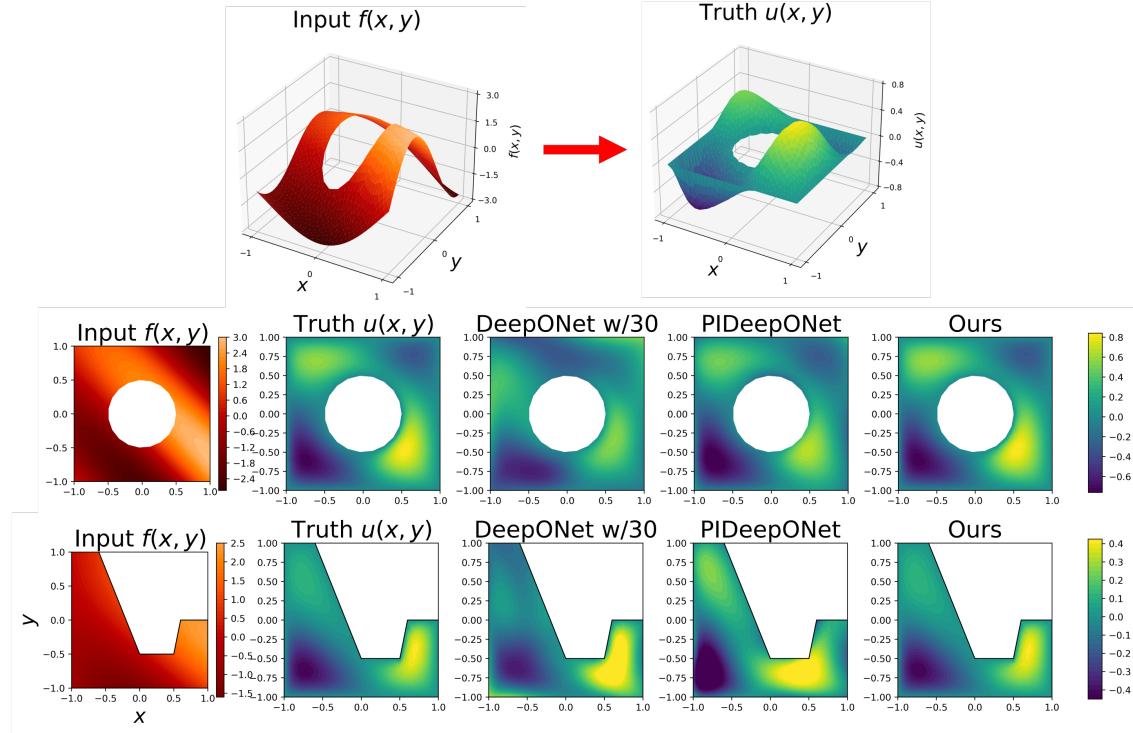


Figure 3: Solution profiles for complex geometries.

3.1.1 Simulations on different domains

We consider the 2D convection-diffusion equation as

$$\begin{aligned} -\varepsilon \Delta u + \mathbf{v} \cdot \nabla u &= f(x, y), \quad (x, y) \in D, \\ u(x, y) &= 0, \quad (x, y) \in \partial D. \end{aligned} \tag{3.1}$$

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 data), PIDDeepONet (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 PIDDeepONet find it challenging to produce accurate predictions. Figure 3 demonstrates the solution profile, emphasizing that FEONet is more precise in predicting the solution than both DeepONet and PIDDeepONet, particularly in complex domains.

3.1.2 Simulations on Dirichlet and Neumann boundary conditions

We tested the performance of each model when the given PDEs were equipped with either Dirichlet or Neumann boundary conditions. We consider the convection-diffusion equation as

$$\begin{aligned} -\varepsilon u_{xx} + bu_x &= f(x), \quad x \in D, \\ u(x) = 0 \quad \text{or} \quad u_x(x) = 0, & \quad x \in \partial D, \end{aligned} \tag{3.2}$$

where $\varepsilon = 0.1$ and $b = -1$. For the Neumann boundary condition, we add u to the left-hand side of (3.2) to address the uniqueness of the solution. 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 fifth and sixth columns in Table 1.

3.1.3 Simulations on linear and nonlinear PDEs

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

$$\begin{aligned} -\varepsilon u_{xx} + b(x)u_x + c(x)u &= f(x), \quad x \in D, \\ u(x) = 0, & \quad x \in \partial D, \end{aligned} \tag{3.3}$$

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

$$\begin{aligned} -u_{xx} + uu_x &= f(x), \quad x \in D, \\ u(x) = 0, & \quad x \in \partial D. \end{aligned} \tag{3.4}$$

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. The seventh and eighth columns of Table 1 highlight the effectiveness of the FEONet, predicting the solutions with reasonably low errors for various equations. Although the eighth 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.

3.1.4 Simulations on various types of input functions

In this section, we will demonstrate that our method can accommodate various types of input functions. Specifically, we shall utilize, as an input for our model, variable coefficients and boundary conditions. Throughout the series of experiments, we will confirm that our approach can make a fast and accurate solution prediction for such various scenarios. Note, however, that the proposed method cannot take the geometry of the domain as an input of neural networks, which is of independent interest (See, for example, the references [54, 49] where this issue is addressed).

Variable coefficient as an input 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 (3.3). Figure 4 displays the solution profile obtained when variable coefficients are considered as input. The second column in Table 2 shows that when the input is a coefficient variable, POD-DeepONet, using all 1,000 data pairs, produces a slightly smaller error than FEONet, but the errors are quite similar. This demonstrates the strength of FEONet in achieving comparable accuracy while being trained in an unsupervised manner.

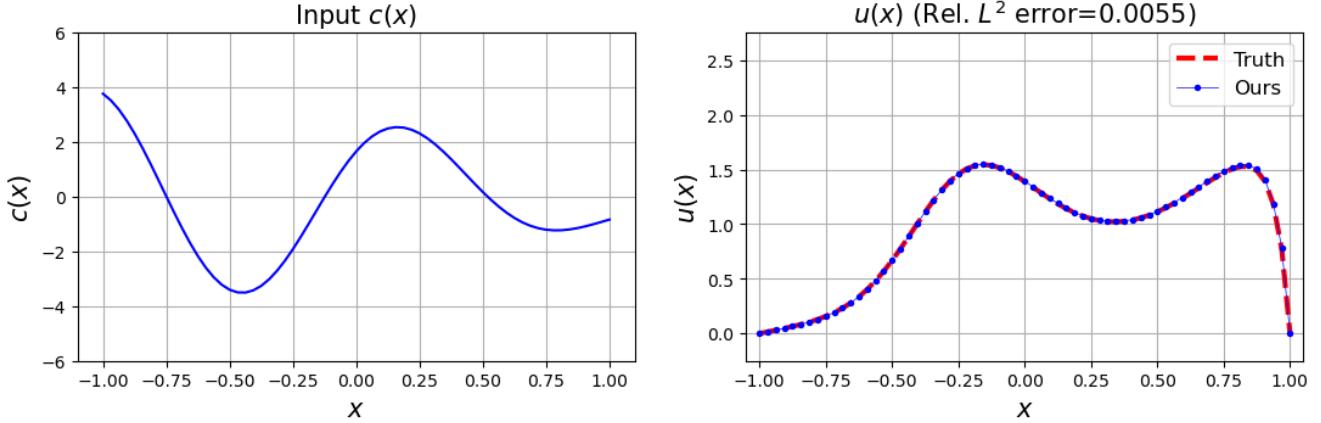


Figure 4: Solution profiles predicted by the trained FEONet for PDE (3.3) obtained from variable coefficients $c(x)$ as an input function.

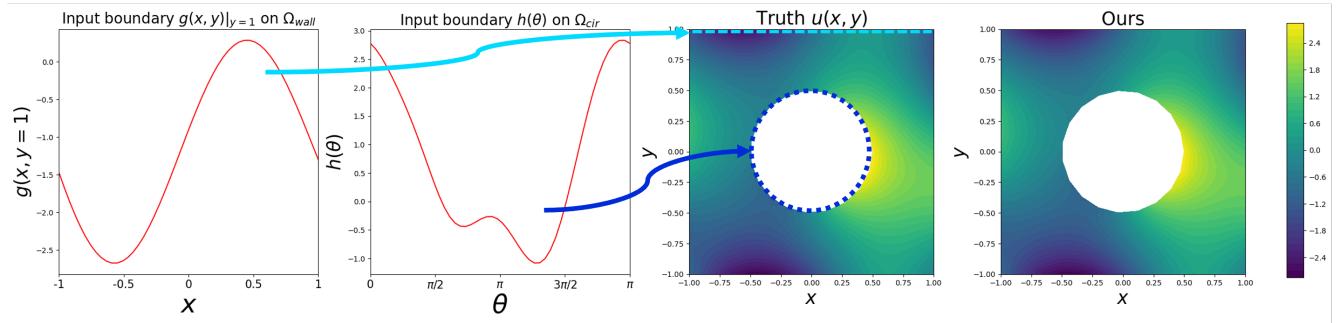


Figure 5: Solution profiles predicted by the trained FEONet for PDE (3.5) obtained from the boundary condition input. Note that the boundary condition $g(x, y)$ for the region with a circle hole inside Ω_{cir} is represented as $h(\theta)$ in polar coordinates. The average of relative L^2 errors across the entire test set when comparing the exact solution to the predictions is 0.0043.

Boundary condition as an input We address the case when the input is a boundary condition. We consider the 2D Poisson equation as

$$\begin{aligned} -\Delta u(x, y) &= f(x, y), \quad (x, y) \in D, \\ u(x, y) &= g(x, y), \quad (x, y) \in \partial D. \end{aligned} \tag{3.5}$$

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 5 demonstrates that even in the case of an operator that takes the boundary condition $g(x, y)$ as input, the FEONet can predict the solution with high accuracy, as shown in the third column of Table 2, compared to POD-DeepONet. This demonstrates the versatility and adaptability of the FEONet approach for accommodating different types of input functions.

3.2 Further extensions of FEONet

In this section, we will extend FEONet to handle the parabolic problem and the Stokes problem.

3.2.1 Time-dependent problem

The FEONet can be extended to handle the parabolic problem with initial conditions as input. We consider the time-dependent problem with varying initial conditions. We consider the 1D convection-diffusion equation

$$\begin{aligned} u_t - \nu u_{xx} + bu_x &= 0, & t \in [0, 1], x \in D = [-1, 1], \\ u(0, x) &= u_0(x), & x \in D, \\ u(t, x) &= 0, & x \in \partial D, \end{aligned} \quad (3.6)$$

where $\nu = 0.1$ and $b = -1$. We aim 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 [27] to sequentially learn and predict solutions over the time domain $t = [0, 1]$, divided into 10 intervals. For training FEONet, we utilized only the input function $u_0(x)$ and equation (3.6) without any additional data. See Appendix B.4 for more details. Figure 6 illustrates FEONet's effective prediction of true solutions using an initial condition from the test dataset. The fourth column in Table 2 shows that the FEONet—trained without any data pairs—yields a smaller error than POD-DeepONet.

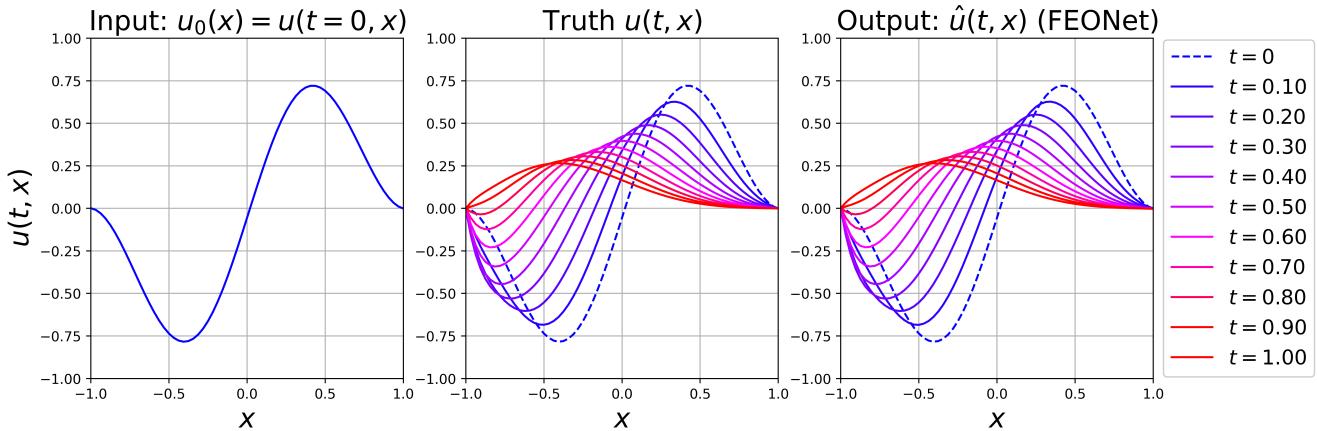


Figure 6: Solution profiles predicted by the trained FEONet for PDE (3.6) obtained from the initial condition $u(t = 0, x)$ as an input function. The average of relative L^2 error across the entire test set, when comparing the exact solution profile with the predicted solution, is 0.0022.

3.2.2 Stokes problem

We consider a more intricate yet practical fluid dynamics issue: the two-dimensional steady-state Stokes equations, which can be described as follows:

$$\begin{aligned} -\nabla \cdot (\nabla \mathbf{u} - pI) &= \mathbf{f}, & (x, y) \in D = [0, 1]^2, \\ \nabla \cdot \mathbf{u} &= 0, & (x, y) \in D, \\ \mathbf{u} &= \mathbf{g}, & (x, y) \in \Gamma_D = \{(x, y) \in D | y = 0\}, \\ (\nabla \mathbf{u} - pI) \cdot \mathbf{n} &= (0, 0), & (x, y) \in \Gamma_N = \{(x, y) \in D | x = 0, 1 \text{ or } y = 1\} \end{aligned} \quad (3.7)$$

where $\mathbf{g}(x, y) = (3 + 1.7 \sin(2\pi x), 0)$. We aim to learn the operator $\mathcal{G} : \mathbf{f} = (f_1, f_2) \mapsto [\mathbf{u}, p]$ using the FEONet. For this, we employ the P2 Lagrange element for velocity and the P1 Lagrange element for pressure, which is known as the Taylor-Hood element [15]. Figure 7 demonstrates the results of the operator learning for the 2D Stokes equation using FEONet. This illustrates that FEONet is applicable to the learning of operators for the system of PDEs with vectorial inputs and outputs. In this problem, which

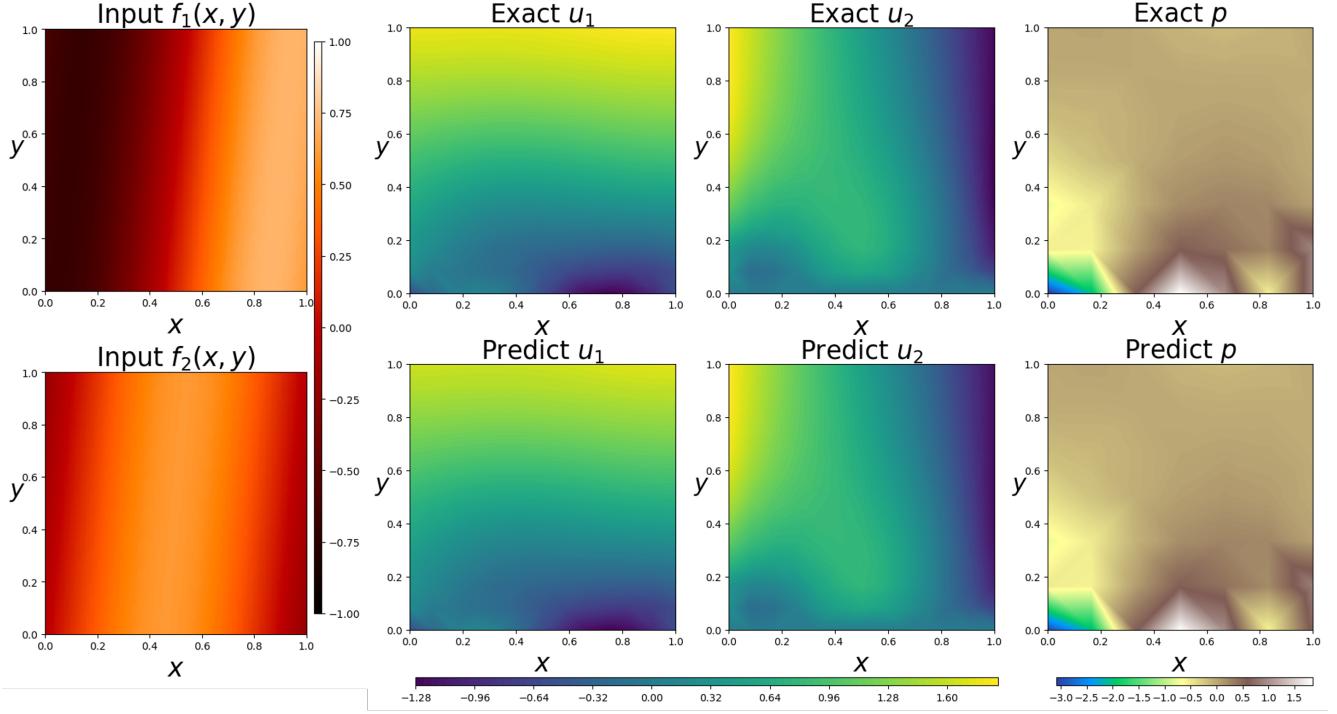


Figure 7: Solution profiles predicted by the trained FEONet for 2D Stokes equation (3.7). The average of relative L^2 errors across the entire test set when comparing the exact solution to the predictions for u_1 , u_2 , and p are 0.0499, 0.1092, and 0.0685, respectively.

requires predicting three outputs— $u_1(x, y)$, $u_2(x, y)$, and $p(x, y)$ —the experiment for POD-DeepONet was omitted as shown in the fifth column of Table 2 because constructing a natural extension of the basis using POD is challenging. This remains an area for future work involving the application of POD-DeepONet.

3.3 Singular perturbation problem

Since the proposed method is based on domain meshing, we may face some potential challenges associated with the scenarios involving singular behavior including interior or boundary shocks. In particular, as our method is not mesh-free, refined meshes are required to handle certain problems, such as shock formation. This is a recognized limitation of our approach. However, while our method may face difficulties in solving singular problems, it benefits from the ability to leverage various existing numerical analysis techniques to address these challenges. To the best of our knowledge, such problems are very challenging for other machine learning-based approaches, including mesh-free methods. More precisely, 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. The boundary layer problem is a well-established challenge in scientific computing. This issue arises from the presence of thin regions near boundaries where the solution exhibits steep gradients. Such regions significantly impact the overall behavior of the system. Typically, in cases where the target function demonstrates abrupt transitions, such as when the diffusion coefficient is small in our model problem, neural network algorithms frequently encounter difficulties in converging to optimal solutions. This is largely due to the phenomenon known as spectral bias [46]. Neural networks are generally predicated on a smooth prior assumption, but spectral bias leads to a failure to accurately capture abrupt transitions or singular behaviors of the target solution function. To address this challenge, we propose a novel semi-analytic machine learning approach, guided by singular perturbation analysis, for effectively capturing the behavior of thin boundary layers. This is

accomplished by incorporating additional basis functions guided by theoretical considerations. For this we consider

$$\begin{aligned} -\varepsilon u_{xx} + bu_x &= f(x), \quad x \in D, \\ u(x) &= 0, \quad x \in \partial D, \end{aligned} \tag{3.8}$$

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 8, 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

$$\phi_{\text{cor}}(x) := e^{-(1+x)/\varepsilon} - (1 - (1 - e^{-2/\varepsilon})(x + 1)/2). \tag{3.9}$$

Table 3: 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
Meshless	No	Yes	Yes	Yes

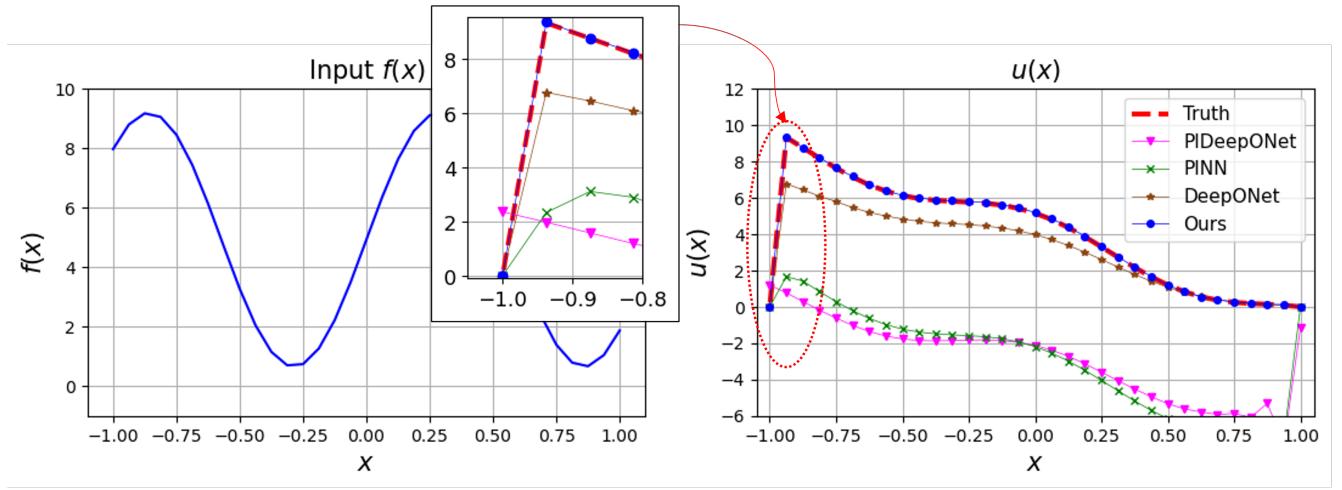


Figure 8: Input function f and the corresponding approximate solution u obtained by various deep-learning-based methods for the singular perturbation problem with $\varepsilon = 10^{-5}$.

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 cost remains minimal as the enriched basis exclusively encompasses boundary elements. Table 3 summarizes the characteristics of various operator learning models and the corresponding errors when solving the singularly perturbed PDE described in (3.8). The performance of the FEONet in accurately solving boundary layer 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. Figure 8 exhibits the results of operator learning for the singular perturbation problem. As depicted in the zoomed-in graph of Figure 8, the FEONet stands out as the only model capable of accurately capturing sharp transitions, whereas other models exhibit noticeable errors.

3.4 Generalization capacity and convergence rate

Generalization capacity 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 3, 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.

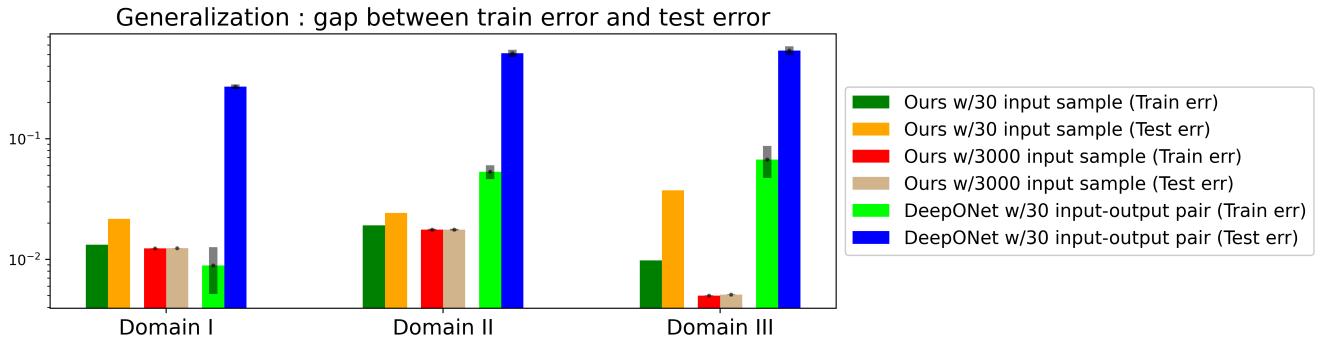


Figure 9: Rel. L^2 train and test error comparison between the FEONet with 30/3000 input samples (w/o labeled data) and the DeepONet (supervised) with 30 train-test data pair.

Figure 9 presents the results comparing the generalization capability of FEONet when trained with only 30 random input samples (green and yellow bars), which should be distinguished from the 30 input-output data pairs used for supervised learning. More specifically, this figure illustrates the generalization capabilities of FEONet in comparison to DeepONet across the three distinct domain settings discussed in Section 3.1: Domain I (circle), Domain II (square with a hole), and Domain III (polygon). Each bar represents the models’ performance when solving Equation (3.1) within these different geometrical domains. Figure 9 compares the training and test errors of FEONet, trained through an unsupervised approach with 30 and 3,000 input samples, against DeepONet, trained through supervised learning with 30 input-output data pairs. The difference between the training and test errors serves as a reliable indicator of the models’ generalization effectiveness. Notably, FEONet demonstrates superior generalization compared to DeepONet, even without requiring input-output paired data. Remarkably, with only 30 input samples for training, FEONet achieves significantly better generalization and exhibits lower test errors than DeepONet. The experimental details can be found in Appendix B.6.

Rate of convergence One of the intriguing aspects is that the theoretical results on the convergence error rate of the FEONet are also observable in the experimental results. Figure 10 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. The experimental details can be found in Appendix B.5.

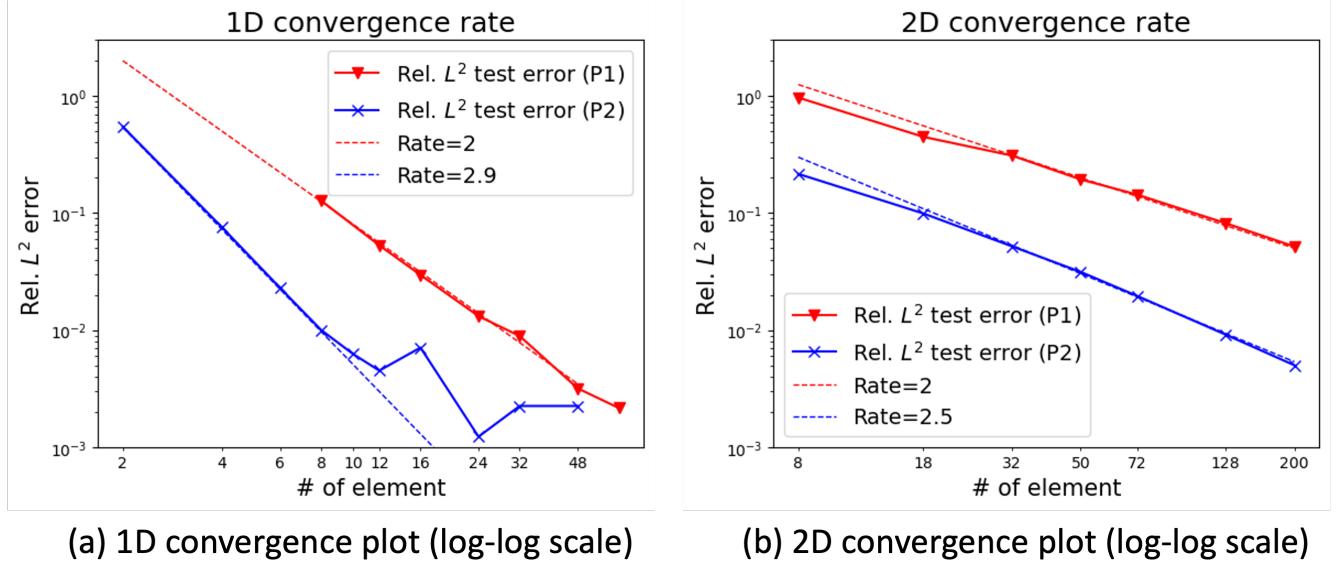


Figure 10: Rel. L^2 error of the FEONet with respect to the number of elements.

Remark 3.1. As we can see from the left in Figure 10, as we increased the number of elements, i.e., on a high-fidelity mesh, the error began to increase again after a certain point, which means that there is an issue regarding the high-dimensional coefficient output of the neural network. To explain this phenomenon, we theoretically investigated the FEONet in our subsequent paper [20] and successfully identified the cause of such phenomenon. More precisely, we discovered that the condition number of the finite element matrix is closely related to the high-fidelity issue of FEONet, which makes total error increase when the degree of freedom is large. Furthermore, this theoretical finding provides a critical clue for resolving the computational issue arising from high-fidelity meshes. By using a preconditioner to define a loss function, we can significantly reduce the total error and the FEONet can effectively make a solution prediction even for the case of a large number of elements. See [20] for details.

4 Conclusions

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 existing 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. The FEONet is capable of learning the operator by incorporating not only external forces but also boundary conditions, variable coefficients, and the initial conditions of time-dependent PDEs. On the other hand, our method also has some drawbacks that need to be overcome. For example, since the proposed method is not mesh-free and based on domain meshing, we may face challenges in solving high-dimensional PDEs or handling singular problems, such as shock formation (see,

e.g. [23]). Note, however, that since our method is based on the classical FEM, we can directly apply numerical analysis techniques to solve these problems, which will be addressed in future work. If these aspects are effectively supplemented, we are confident that the FEO Net 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.

A Proof of theoretical results

A.1 Approximation error

We shall prove Theorem 2.4 addressing the approximation error for the coefficients.

Proof of Theorem 2.6. Since A is symmetric and invertible, by Proposition 2.5, we obtain

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

Note that the constants appearing in the inequalities above 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, $\inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_2^2 \rightarrow 0$ as $n \rightarrow \infty$, which completes the proof. \square

A.2 Generalization error

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

Theorem A.1. [Theorem 4.10 in [52]] *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*

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

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

Next, note from Lemma 2.5 that

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

Since the class of neural networks under consideration is uniformly bounded and (2.9) holds, we see that for any $n \in \mathbb{N}$, \mathcal{G}_n is \tilde{b} -uniformly bounded for some $\tilde{b} > 0$. Then the following lemma is the direct consequence of Theorem A.1 within our setting.

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

$$\sup_{\alpha \in \mathcal{N}_n} |\mathcal{L}^M(\alpha) - \mathcal{L}(\alpha)| \leq 2R_n(\mathcal{G}_n) + \frac{\delta}{2}. \quad (\text{A.1})$$

Now from Lemma A.2, we shall prove Theorem 2.6 concerning the generalization error for the coefficients.

Proof of Theorem 2.8. By Proposition 2.5 and the definition (2.24), we obtain

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

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

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

Using Lemma A.2 again gives us that

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

Letting $M \rightarrow \infty$ on (A.2), we obtain that

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

As we did before, we conclude that

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

which completes the proof. \square

A.3 Convergence of FEONet

Based on the results proved above, now we can address the convergence of the predicted solution by the FEONet to the finite element solution.

Proof of Theorem 2.9. From Theorem 2.6, Theorem 2.8, we note that for a fixed $h > 0$,

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

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

B Experimental details

B.1 Types and random generation of input functions for FEONet

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 [4], we created a random signal $f(\mathbf{x}; \boldsymbol{\omega})$ as a linear combination of sine functions and cosine functions. More precisely, we use

$$f(x) = m_0 \sin(n_0 x) + m_1 \cos(n_1 x) \quad (\text{B.1})$$

for a 1D case and

$$f(x, y) = m_0 \sin(n_0 x + n_1 y) + m_1 \cos(n_2 x + n_3 y) \quad (\text{B.2})$$

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. 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.

B.2 Details on hyperparameters

For the problems under consideration in the paper, we used neural networks, which consist 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 step: 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 (or POD-DeepONet) and the PIDDeepONet, 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 PIDDeepONet for 10^4 iterations. Note that computing the residual loss of PDE for every input function f in the PIDDeepONet requires significant computation time using the automatic differentiation.

B.3 Comparison of computational time

Once the operator-learning ML models (DeepONet, POD-DeepONet, and FEONet) are trained, they can quickly infer solutions for varying PDE parameters. This capability is a key reason why operator networks are becoming increasingly popular in scientific machine learning. In contrast, solving PDEs with traditional FEM for various parameters requires re-solving the PDE each time the parameters change, incurring computational costs. This challenge is further exacerbated when dealing with nonlinear problems, where iterative methods must be applied for each set of PDE parameters, adding to the computation overhead. As mentioned earlier, FEONet is particularly well-suited to address this issue, highlighting a critical

difference between traditional FEM and FEONet. To quantify this advantage, let us consider a scenario where 10,000 parametric PDEs with varying parameters need to be solved—a common requirement in computational material database generation. In such cases, operator learning methods like FEONet are significantly faster than traditional FEM. Table 4 illustrates the inference time comparison based on the number of input samples, underscoring the efficiency of models like FEONet in fast-inference applications.

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

# input samples	1	10	100	1000	10000
FEM	0.012 (s)	0.141 (s)	1.039 (s)	9.979 (s)	99.826 (s)
DeepONet	0.147 (s)	0.123 (s)	0.251 (s)	0.338 (s)	0.746 (s)
POD-DeepONet	0.170 (s)	0.190 (s)	0.191 (s)	0.398 (s)	0.765 (s)
FEONet (Ours)	0.106 (s)	0.221 (s)	0.315 (s)	0.374 (s)	1.529 (s)

B.4 Extension of FEONet to time-dependent PDEs

We use the FEONet to learn an operator that maps $u_0(x) = u(t = 0, x)$ to $u(\Delta t, x)$ (more precisely, the coefficients of $u(\Delta t, x)$) in Equation (3.6). The other model then predicts the solution at the next time step solution $u(2\Delta t, x)$ as an input $u(\Delta t, x)$, and continues iteratively. To facilitate this, we employed the implicit Euler method to discretize time with $\Delta t = 0.01$, and specifically trained the model using the following loss function:

$$\begin{aligned} \mathcal{L}^M(\hat{u}_h) = \frac{1}{M} \sum_{m=1}^M \sum_{i=1}^N & \left| \int_D \frac{\hat{u}_h(l\Delta t, x) - \hat{u}_h((l-1)\Delta t, x)}{\Delta t} \phi_i dx \right. \\ & \left. + \nu \int_D \hat{u}_h(l\Delta t, x)_x (\phi_i)_x dx - b \int_D \hat{u}_h(l\Delta t, x) (\phi_i)_x dx \right| \quad (B.3) \end{aligned}$$

for $l = 1, 2, \dots, 100$ where $\hat{u}_h = \hat{u}_h(l\Delta t, x; \omega_m)$. This approach could involve training 100 FEONet models over the interval $[0, 1]$ by dividing it into 100 segments. However, for efficiency, we divided the interval into 10 segments and modified a model with $u(t = 0, x)$ as input and $[u(t = k\Delta t, x)]_{k=1}^{10}$ as output, and then used the output $u(t = 10\Delta t, x)$ as input for the next model. Therefore, this approach involved using 10 separate FEONet models.

B.5 Plots on convergence rates

Let us make some further comments on the experimental details for Figure 10. For 1D case, we consider the convection-diffusion equation

$$\begin{aligned} -\varepsilon u_{xx} + bu_x &= f(x), \quad x \in D = [-1, 1], \\ u(x) &= 0 \quad , \quad x \in \partial D, \end{aligned} \quad (B.4)$$

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

$$\begin{aligned} -\varepsilon \Delta u + \mathbf{v} \cdot \nabla u &= f(x, y), \quad (x, y) \in D = [-1, 1]^2, \\ u(x, y) &= 0, \quad (x, y) \in \partial D, \end{aligned} \quad (B.5)$$

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 10, the observed convergence rate of the FEONet shows 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.

B.6 Additional plot on generalization capacity

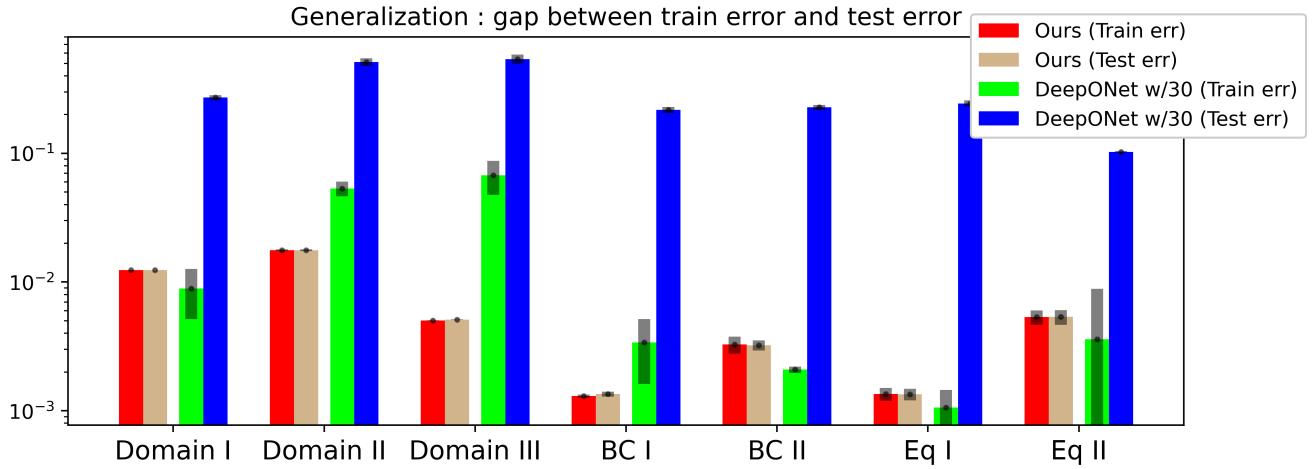


Figure 11: Rel. L^2 train and test error comparison between the FEONet and the DeepONet with 30 training data. Results are averaged over five independent training trials, and black bars indicate the standard deviation.

Figure 11 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 3 (Table 1). 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 are 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. As depicted in Figure 11, the generalization error of FEONet is notably smaller in comparison to that of DeepONet when trained with 30 data pairs.

Acknowledgments

Y. Hong was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2021R1A2C1093579) and by the Korea government(MSIT) (RS-2023-00219980). Jae Yong Lee was supported by Institute for Information & Communications Technology Planning & Evaluation (IITP) through the Korea government (MSIT) under Grant No. 2021-0-01341 (Artificial Intelligence Graduate School Program (Chung-Ang University)). S. Ko was supported by National Research Foundation of Korea Grant funded by the Korean Government (RS-2023-00212227). The authors thank the anonymous referees for their helpful comments that improved the quality of the manuscript.

References

- [1] M. Alber, A. Buganza Tepole, W. R. Cannon, S. De, S. Dura-Bernal, K. Garikipati, G. Karniadakis, W. W. Lytton, P. Perdikaris, L. 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.
- [2] 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.
- [3] K. Atkinson. *An introduction to numerical analysis*. John wiley & sons, 1991.
- [4] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.
- [5] P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: risk bounds and structural results. *J. Mach. Learn. Res.*, 3(Spec. Issue Comput. Learn. Theory):463–482, 2002.
- [6] E. A. Bender. *An introduction to mathematical modeling*. Courier Corporation, 2000.
- [7] J. Brandstetter, D. E. Worrall, and M. Welling. Message Passing Neural PDE Solvers. In *International Conference on Learning Representations*, 2022.
- [8] S. C. Brenner and R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer Science & Business Media, New York, 2008.
- [9] R. L. Burden, J. D. Faires, and A. M. Burden. *Numerical analysis*. Cengage learning, 2015.
- [10] M. D. Chekroun, Y. Hong, and R. Temam. Enriched numerical scheme for singularly perturbed barotropic quasi-geostrophic equations. *Journal of Computational Physics*, 416:109493, 2020.
- [11] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. Society for Industrial and Applied Mathematics, 2002.
- [12] R. Courant and D. Hilbert. *Methods of Mathematical Physics*. Interscience, New York, 2nd edition, 1953. Translated from German: Methoden der mathematischen Physik I, 2nd ed, 1931 [15].
- [13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, 2(4):303–314, 1989.
- [14] W. E, C. Ma, and L. Wu. The Barron space and the flow-induced function spaces for neural network models. *Constr. Approx.*, 55(1):369–406, 2022.
- [15] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Oxford university press, 2014.
- [16] V. Fanaskov and I. Oseledets. Spectral Neural Operators. *arXiv preprint arXiv:2205.10573*, 2022.
- [17] N. A. Gershenfeld and N. Gershenfeld. *The nature of mathematical modeling*. Cambridge university press, 1999.
- [18] G. Gnecco and M. Sanguineti. Approximation error bounds via Rademacher’s complexity. *Appl. Math. Sci. (Ruse)*, 2(1-4):153–176, 2008.
- [19] S. Goswami, M. Yin, Y. Yu, and G. E. Karniadakis. A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 391:114587, 2022.

- [20] Y. Hong, S. Ko, and J. Lee. Error analysis for finite element operator learning methods for solving parametric second-order elliptic pdes, 2024.
- [21] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.
- [22] T. J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, 2000.
- [23] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.
- [24] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.
- [25] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, 2021.
- [26] S. Ko, S.-B. Yun, and Y. Hong. Convergence analysis of unsupervised Legendre-Galerkin neural networks for linear second-order elliptic PDEs, 2022.
- [27] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.
- [28] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.
- [29] M. G. Larson and F. Bengzon. *The finite element method: theory, implementation, and applications*, volume 10. Springer Science & Business Media, 2013.
- [30] J. Y. Lee, S. CHO, and H. J. Hwang. HyperDeepONet: learning operator with complex target function space using the limited resources via hypernetwork. In *The Eleventh International Conference on Learning Representations*, 2023.
- [31] R. J. LeVeque. *Finite-Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.
- [32] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*, 2021.
- [33] Z. Li, H. Zheng, N. B. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar. Physics-informed neural operator for learning partial differential equations. *CoRR*, abs/2111.03794, 2021.
- [34] M. Lienen and S. Günnemann. Learning the Dynamics of Physical Systems from Sparse Observations with Finite Element Networks. In *International Conference on Learning Representations*, 2022.
- [35] C.-C. Lin and L. A. Segel. *Mathematics applied to deterministic problems in the natural sciences*. SIAM, 1988.
- [36] A. Logg, K. Mardal, G. N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

- [37] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.
- [38] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. 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.
- [39] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.
- [40] Y. Lu, J. Lu, and M. Wang. A priori generalization analysis of the deep ritz method for solving high dimensional elliptic partial differential equations. In M. Belkin and S. Kpotufe, editors, *Proceedings of Thirty Fourth Conference on Learning Theory*, volume 134 of *Proceedings of Machine Learning Research*, pages 3196–3241. PMLR, 15–19 Aug 2021.
- [41] X. Meng, Z. Li, D. Zhang, and G. E. Karniadakis. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Computer Methods in Applied Mechanics and Engineering*, 370:113250, 2020.
- [42] S. T. Miller, N. V. Roberts, S. D. Bond, and E. C. Cyr. Neural-network based collision operators for the boltzmann equation. *Journal of Computational Physics*, 470:111541, 2022.
- [43] M. E. Mortenson. *Mathematics for computer graphics applications*. Industrial Press Inc., 1999.
- [44] R. G. Patel, N. A. Trask, M. A. Wood, and E. C. Cyr. A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373:113500, 2021.
- [45] A. Pinkus. Approximation theory of the MLP model in neural networks. In *Acta numerica*, 1999, volume 8 of *Acta Numer.*, pages 143–195. Cambridge Univ. Press, Cambridge, 1999.
- [46] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.
- [47] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.
- [48] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.
- [49] K. Shukla, V. Oommen, A. Peyvan, M. Penwarden, N. Plewacki, L. Bravo, A. Ghoshal, R. M. Kirby, and G. E. Karniadakis. Deep neural operators as accurate surrogates for shape optimization. *Engineering Applications of Artificial Intelligence*, 129:107615, 2024.
- [50] C. P. Simon, L. Blume, et al. *Mathematics for economists*, volume 7. Norton New York, 1994.
- [51] J. C. Strikwerda. *Finite difference schemes and partial differential equations*. SIAM, 2004.
- [52] M. J. Wainwright. *High-dimensional statistics*, volume 48 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 2019. A non-asymptotic viewpoint.

- [53] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, 7(40):eabi8605, 2021.
- [54] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science Advances*, 7(40):eabi8605, 2021.
- [55] L. Yang, X. Meng, and G. E. Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.
- [56] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 2000.