Journal of Computational Physics 514 (2024) 113217

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Solving parametric elliptic interface problems via interfaced operator network

Sidi Wu [a,b], Aiqing Zhu [b], Yifa Tang [b], Benzhuo Lu [b,*]

[a] School of Mathematical Sciences, Peking University, Beijing 100871, China
[b] LSEC, ICMSEC, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China

ARTICLE INFO

Keywords:
Parametric elliptic interface problems
Interfaced operator network
Operator regression
Mesh-free method

ABSTRACT

Learning operators mapping between infinite-dimensional Banach spaces via neural networks has attracted a considerable amount of attention in recent years. In this paper, we propose an interfaced operator network (IONet) to solve parametric elliptic interface PDEs, where different coefficients, source terms, and boundary conditions are considered as input features. To capture the discontinuities in both the input functions and the output solutions across the interface, IONet divides the entire domain into several separate subdomains according to the interface and uses multiple branch nets and trunk nets. Each branch net extracts latent representations of input functions at a fixed number of sensors on a specific subdomain, and each trunk net is responsible for output solutions on one subdomain. Additionally, tailored physics-informed loss of IONet is proposed to ensure physical consistency, which greatly reduces the training dataset requirement and makes IONet effective without any paired input-output observations inside the computational domain. Extensive numerical studies demonstrate that IONet outperforms existing state-of-the-art deep operator networks in terms of accuracy and versatility.

1. Introduction

Elliptic interface problems have widespread applications across various fields, including fluid mechanics [1,2], materials science [3,4], electromagnetics [5], biomimetics [6,7], and flow in porous media [8]. Accurate modeling and rapid evaluation of these differential equations are critical in both scientific research and engineering applications. Many computational tasks arising in science and engineering often involve repeated evaluation of the outputs of an expensive forward model for many statistically similar inputs. These tasks, known as parametric PDE problems, encompass various areas such as inverse problems, control and optimization, risk assessment, and uncertainty quantification [9,10]. When dealing with parametric PDEs with discontinuous coefficients across certain interfaces, i.e., parametric interface problems, the low global regularity of the solution and the irregular geometry of the interface give rise to additional challenges, particularly for problems with non-smooth interfaces containing geometric singularities such as sharp edges, tips, and cusps.

Consider an open and bounded domain $\Omega \subset \mathbb{R}^d$ with a Lipschitz boundary $\partial\Omega$. The domain Ω is separated into two disjoint subdomains, Ω_1 and Ω_2, by an interface Γ. A sketch of the computational domain considered in 2D is shown in Fig. 1. Then parametric second-order linear elliptic interface problems are of the form:

* Corresponding author.
E-mail address: bzlu@lsec.cc.ac.cn (B. Lu).

https://doi.org/10.1016/j.jcp.2024.113217
Received 28 August 2023; Received in revised form 16 June 2024; Accepted 19 June 2024

Available online 26 June 2024

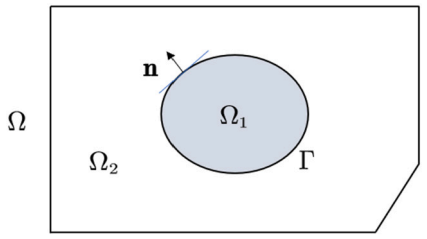


Fig. 1. Domain Ω, its subdomains Ω_1, Ω_2. The interface Γ divides Ω into two disjoint subdomains.

$$-\nabla \cdot (a\nabla u) + bu = f, \quad \text{in } \Omega \setminus \Gamma, \tag{1a}$$

$$[\![u]\!] = g_D, \quad \text{on } \Gamma, \tag{1b}$$

$$[\![a\nabla u \cdot \mathbf{n}]\!] = g_N, \quad \text{on } \Gamma, \tag{1c}$$

$$u = h, \quad \text{on } \partial\Omega, \tag{1d}$$

where $\mathbf{n}$ denotes the outward unit normal vectors of the interface Γ (from Ω_1 to Ω_2), $[\![\cdot]\!]$ denotes the jump across the interface, for a point $\mathbf{x}^\gamma \in \Gamma$,

$$[\![u]\!](\mathbf{x}^\gamma) := \lim_{\substack{\mathbf{x}\in\Omega_2 \\ \mathbf{x}\to\mathbf{x}^\gamma}} u(\mathbf{x}) - \lim_{\substack{\mathbf{x}\in\Omega_1 \\ \mathbf{x}\to\mathbf{x}^\gamma}} u(\mathbf{x}),$$

$$[\![a\nabla u \cdot \mathbf{n}]\!](\mathbf{x}^\gamma) := \lim_{\substack{\mathbf{x}\in\Omega_2 \\ \mathbf{x}\to\mathbf{x}^\gamma}} a(\mathbf{x})\nabla u(\mathbf{x}) \cdot \mathbf{n} - \lim_{\substack{\mathbf{x}\in\Omega_1 \\ \mathbf{x}\to\mathbf{x}^\gamma}} a(\mathbf{x})\nabla u(\mathbf{x}) \cdot \mathbf{n}.$$

Here, $g_D(\mathbf{x}) : \Gamma \to \mathbb{R}$ and $g_N(\mathbf{x}) : \Gamma \to \mathbb{R}$ are the interface conditions, and $h(\mathbf{x}) : \partial\Omega \to \mathbb{R}$ is the boundary condition; the coefficient $a(\mathbf{x}) : \Omega \to \mathbb{R}$ is continuous and positive in each of the subdomains but discontinuous across the interface; the coefficient $b(\mathbf{x}) : \Omega \to \mathbb{R}$ and source $f(\mathbf{x}) : \Omega \to \mathbb{R}$ are continuous in each of the subdomains but may be discontinuous across the interface. Additionally, we will also consider a nonlinear example, i.e., replacing bu here with $b(u)$. The latent solution $u(\mathbf{x}) : \Omega \to \mathbb{R}$ to this problem typically has higher regularity in each subdomain, but lower global regularity across the whole domain, even with discontinuities at the interface. Solving these parametric elliptic interface problems requires learning the solution operator that maps variable PDE parameters such as the coefficient $a(\mathbf{x})$ and the source term $f(\mathbf{x})$ directly to the corresponding solution u. This paper introduces a novel operator network for approximating operators involving discontinuities in both input and output functions. We then demonstrate its effectiveness in approximating the solution operator of parametric elliptic interface problems.

Classical numerical methods for solving elliptic interface problems can be roughly divided into two categories: interface-fitted methods and interface-unfitted methods. The first type of approach is suitable for solving PDE problems defined in complex domains. The methods in this category include classical finite element method (FEM) [11–13], boundary element method (BEM) [14], weak Galerkin method [15], and so on. To maintain optimum convergence behavior, these methods require the mesh surface to be aligned with the interface. This alignment ensures that interface conditions are correctly applied, enhancing the accuracy of numerical solutions. However, generating interface-fitted meshes for irregular domains or interfaces could result in significant computational costs [16], especially when high accuracy is required. To alleviate the burden of mesh generation, many works employ interface-unfitted meshes (e.g., a uniform Cartesian mesh) to discretize the computational domain and enforce interface conditions by modifying finite difference stencils or finite element bases near the interface. For instance, the immersed boundary method (IBM) [17], the immersed interface method (IIM) [18], the immersed finite element method [19], the ghost fluid method (GFM) [20] and its improvement (xGFM) [21], the cartesian grid finite volume approach (FVM) [22,23], the matched interface and boundary method [24], the extended finite element methods (XFEM) [25,26], and references therein. In general, the numerical solution of these methods becomes more accurate with mesh refinement, but also more time consuming.

Besides the mesh-based methods, there are also numerous efforts focusing on mesh-free numerical methods for interface problems, such as direct meshless local Petrov-Galerkin method [27], the global RBF-QR collection method [28], the local RBF meshless methods [29], and the meshless method based on pascal polynomials and multiple-scale approach [30]. Alternatively, there is a growing interest in utilizing neural network-based methods to solve elliptic interface problems. For instance, [31] employed a shallow neural network to remove the inhomogeneous boundary conditions and developed a deep Ritz-type approach to solve the interface problem with continuous solutions. An important development in this direction is the combination of deep learning and domain decomposition methods due to the observation that the solution to the interface problem is typically piece-wise continuous. The solution to Eq. (1) can be approximated by minimizing a loss function derived from either the least squares principle [32,33] or the variational principle [34,35]. Moreover, adaptively setting appropriate penalty weights among different terms in the loss function could improve accuracy [33,36,37]. And specially designed neural network structures, such as incorporating multi-scale features [38,39] and augmenting extra feature input [40–42], are also able to further enhance the performance of neural models.

Although these numerical methods have been shown to be effective to some extent, they are only employed to solve a given instance of the elliptic interface problem (1), where the coefficient functions $a(\mathbf{x})$ and $b(\mathbf{x})$, the forcing term $f(\mathbf{x})$, the interface conditions $g_D(\mathbf{x})$ and $g_N(\mathbf{x})$, and the boundary condition $h(\mathbf{x})$ are given in advance. In other words, these methods treat a PDE

with different parameters as different tasks, each of which needs to be solved end-to-end, which is computationally expensive and time-consuming. To address these challenges, one approach is to employ a reduced-order model that leverages a set of high-fidelity solution snapshots to construct rapid emulators [43,44]. However, the validity of this method relies on the assumption that the solution set is contained approximately in a low-dimensional manifold, which could potentially lead to compromised accuracy and generalization performance [45,46].

Recently, as an emerging paradigm in scientific machine learning, several operator neural networks, such as PDE-Nets [47], deep operator network (DeepONet) [48] and Fourier neural operator (FNO) [49], have been developed to directly learn the solution mapping between two infinite-dimensional function spaces. Although prediction accuracy may be limited, the ability of neural networks to learn from data makes them particularly well suited to this task. Such methods have great potential for developing fast forward and inverse solvers for PDE problems, and have shown good performance in building surrogate models for many types of PDEs, including the Burgers' equation [48,50], Navier-Stokes equations [49,51], Darcy flow [51], diffusion-reaction PDE [52], and so on.

Despite the aforementioned success, these operator learning methods typically have difficulty effectively capturing the discontinuities of input and output functions due to the following two reasons. Theoretically, their approximation theory usually assumes that the input and output functions are continuous [48]. Practically, to handle input functions numerically, we typically need to discretize the input functions and evaluate them at a set of locations. This approach may overlook the discontinuity of the true input functions, as the input functions were expected to be continuous. In addition, the output functions are represented by a network which is a continuous approximator. However, for interface problems, the global regularity of the coefficients and the solutions is usually very low, even discontinuous [53]. These limit the ability of operator networks to accurately represent and learn the complex behavior associated with interface problems.

To address these limitations, in this paper we propose a novel mesh-free method for approximating the solution operator of parametric elliptic interface problems. Different from existing operator networks, we divide the whole domain into several separate subdomains according to the interface, and leverage multiple branch nets and trunk nets. Specifically, each branch net encodes the input function at a fixed number of sensors in each subdomain, and each trunk net is responsible for output solutions in each subdomain. Such an architecture allows the model to accommodate irregularities in input functions and solutions. In addition, tailored physics-informed loss is proposed to ensure physical consistency, which greatly reduces the requirement for training datasets and makes the network effective without any paired input-output observations in the interior of the computational domain. The proposed method circumvents mesh generation and numerical discretization at the interface(s), thus easily handling problems in irregular domains. And the model can be trained only once for fast simulation with different input functions. Herein, we name this neural operator as the Interfaced Operator Network (IONet). Numerical results show that IONet exhibits better accuracy, as well as generalization properties for various input parameters, compared with state-of-the-art neural models.

The rest of this paper is organized as follows: In Section 2, we review the basic idea of the operator network and the DeepONet method. In Section 3, we introduce the proposed interfaced neural network in detail. Then, in Section 4, we investigate the performance of the proposed methods in several typical numerical examples. Finally, we conclude the paper and discuss some future directions in Section 5.

2. Learning operators with neural networks

In this section, we briefly introduce the DeepONet model architecture [48] and its two extensions, the Multi-input operator network (MIONet) [50] and the Physics-informed DeepONet (PI-DeepONet) [52], for learning nonlinear operators between infinite function spaces.

Let $\mathcal{V}$ and $\mathcal{U}$ be two Banach spaces, and let $\mathcal{G}$ be an operator that maps between these two infinite-dimensional function spaces, i.e., $\mathcal{G}:\mathcal{V}\to\mathcal{U}$. We assume that for each $v(\mathbf{y}):\mathbf{y}\to\mathbb{R}$ in $\mathcal{V}$, there exists a unique corresponding output function u in $\mathcal{U}$ that can be represented as $\mathcal{G}(v)(\mathbf{x}):\mathbf{x}\to\mathbb{R}$. Analogously, in the context of parametric PDE problems, $\mathcal{V}$ and $\mathcal{U}$ are denoted as the input function space and the solution space, respectively. Following the original works of [48,52], an unstacked DeepONet $\mathcal{G}_\theta$ is trained to approximate the target solution operator $\mathcal{G}$, where $\mathcal{G}_\theta$ prediction of a function (an input parameter) $v\in\mathcal{V}$ evaluated at a point $\mathbf{x}$ can be expressed as

$$\begin{aligned}\mathcal{G}_\theta(v)(\mathbf{x}) &= \underbrace{\mathcal{N}_b(v(\mathbf{y}_1),v(\mathbf{y}_2),\cdots,v(\mathbf{y}_m))^T}_{branch\ net}\underbrace{\mathcal{N}_t(\mathbf{x})}_{trunk\ net}+\underbrace{b_0}_{bias}\\ &=\sum_{k=1}^{K}b_k t_k+b_0,\end{aligned}$$

where θ denotes all the trainable parameters, i.e., the set consisting of the parameters in the branch net $\mathcal{N}_b$ and the trunk net $\mathcal{N}_t$ and the bias $b_0\in\mathbb{R}$. Here, $[b_1,b_2,\cdots,b_K]^T\in\mathbb{R}^K$ denotes the output of $\mathcal{N}_b$ as a feature embedding of the input function v, $[t_1,t_2,\cdots,t_K]^T\in\mathbb{R}^K$ represents the output of $\mathcal{N}_t$, and $\{\mathbf{y}_1,\mathbf{y}_2,\cdots,\mathbf{y}_m\}$ is a collection of fixed point locations referred to as "sensors", where we discretize the input function v. DeepONets are capable of approximating arbitrary continuous operators [54,48], making them a powerful tool in the field of scientific computing.

MIONet [50] extends the architecture and approximation theory of DeepONet to the case of operators defined on multiple Banach spaces. Let $\mathcal{G}$ be a multi-input operator defined on the product of Banach spaces:

$$\mathcal{G}: \mathcal{V}_1 \times \mathcal{V}_2 \times \cdots \times \mathcal{V}_q \to \mathcal{U},$$

where $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_q$ are q different input Banach spaces that can be defined on different domains, and $\mathcal{U}$ denotes the output Banach space. Then, when we employ a MIONet $\mathcal{G}_\theta$ to approximate the operator $\mathcal{G}$, for a given input function $(v_1, v_2, \cdots, v_q) \in \mathcal{V}_1 \times \mathcal{V}_2 \times \cdots \times \mathcal{V}_q$, the prediction of $\mathcal{G}_\theta(v_1, v_2, \cdots, v_q)$ at a point $\mathbf{x}$ is formulated as

$$\begin{aligned}\mathcal{G}_\theta(v_1, v_2, \cdots, v_q)(\mathbf{x}) &= \mathcal{S}\left(\underbrace{\mathcal{N}_{b_1}(\boldsymbol{v_1})}_{branch_1} \odot \underbrace{\mathcal{N}_{b_2}(\boldsymbol{v_2})}_{branch_2} \odot \cdots \odot \underbrace{\mathcal{N}_{b_q}(\boldsymbol{v_q})}_{branch_q} \odot \underbrace{\mathcal{N}_t(\mathbf{x})}_{trunk}\right) + \underbrace{b_0}_{bias} \\ &= \sum_{k=1}^{K} t_k \prod_{i=1}^{q} b_k^i + b_0.\end{aligned}$$

Here, $\mathcal{S}$ is the sum of all the components of a vector, and $\odot$ represents the Hadamard product. Each input function v_i is projected onto finite-dimensional spaces $\mathbb{R}^{m_i}$ as $\boldsymbol{v_i} := [v_i(\mathbf{y}_1^i), v_i(\mathbf{y}_2^i), \cdots, v_i(\mathbf{y}_{m_i}^i)]^T$ in the same manner as in DeepONet, where $\{\mathbf{y}_j^i\}_{j=1}^{m_i}$ is the set of sensors in the domain of v_i. Similarly, $[b_1^i, b_2^i, \cdots, b_K^i]$ and $[t_1, t_2, \cdots, t_K]$ denote the output of the i-th branch net $\mathcal{N}_{b_i}(\boldsymbol{v_i})$ and the trunk net $\mathcal{N}_t(\mathbf{x})$, respectively.

In the framework of vanilla DeepONet, a data-driven (DD) approach is used to train the network. Specifically, the training dataset consists of paired input-output observations, and the trainable parameters θ can be identified by minimizing the following empirical loss function:

$$\text{Loss}(\theta) = \frac{1}{NP}\sum_{n=1}^{N}\sum_{p=1}^{P}\left|\mathcal{G}_\theta(v^n)(\mathbf{x}_{n,p}) - \mathcal{G}(v^n)(\mathbf{x}_{n,p})\right|^2,$$

where $\{v^n\}_{n=1}^N$ denotes N input functions sampled from the parameter space $\mathcal{V}$. For each input function of DeepONet, the training data points $\{\mathbf{x}_{n,p}\}_{p=1}^P$ are randomly sampled from the computational domain of $\mathcal{G}(v^n)$ and can be set to vary for different n.

Given that the DeepONet architecture provides a continuous approximation of the target functions that is independent of the resolution, the derivatives of the output function can be computed during training. This import feature motivated the work of PI-DeepONet [52], where the trainable parameters can be optimized by minimizing the residuals of the governing equations and the corresponding boundary conditions through the use of automatic differentiation [55]. Specifically, consider a generic parametric PDE expressed as:

$$\begin{aligned}\mathcal{L}(v, u) &= 0, \quad \text{in } \Omega, \\ u &= h, \quad \text{on } \partial\Omega,\end{aligned}$$

where v and u denote the input function and latent solution, respectively. Then, the physics-informed loss function of PI-DeepONet can be formulated as

$$\text{Loss}(\theta) = \lambda_r \text{Loss}_r(\theta) + \lambda_b \text{Loss}_b(\theta).$$

Here, λ_r and λ_b are non-negative weights, the loss term

$$\text{Loss}_r(\theta) = \frac{1}{NP_r}\sum_{n=1}^{N}\sum_{p=1}^{P_r}\left|\mathcal{L}(v^n, \mathcal{G}_\theta(v^n))(\mathbf{x}_{n,p}^r)\right|$$

forces the operator network to satisfy the underlying physical constraints, and

$$\text{Loss}_b(\theta) = \frac{1}{NP_b}\sum_{n=1}^{N}\sum_{p=1}^{P_b}\left|\mathcal{G}_\theta(v^n)(\mathbf{x}_{n,p}^b) - h(\mathbf{x}_{n,p}^b)\right|$$

penalizes the violation of the boundary conditions, where $\{\mathbf{x}_{n,p}^r\}_{p=1}^{P_r}$ and $\{\mathbf{x}_{n,p}^b\}_{p=1}^{P_b}$ denote the training data points randomly sampled from the interior and the boundary of the domain Ω, respectively.

3. Interfaced operator network

In this section, we discuss neural network-based methods for numerically solving parametric interface problems. The main idea of our new method is to approach the solution operator through multiple suboperators while remaining consistent with the potential physical constraints. To simplify the explanation, we mainly present our method for the case of two subdomains. Note that this setting can be easily generalized to a multi-domain scenario, depending on the number of distinct domains involved. Specifically, we consider Eq. (1) as a parametric interface problem of general form. In the following, we illustrate our method with the example of learning the solution operator $\mathcal{G}$ that maps the coefficient function $a(\mathbf{x})$ to the solution $u(\mathbf{x})$ of Eq. (1), i.e., $\mathcal{G}: a(\mathbf{x}) \to u(\mathbf{x})$.

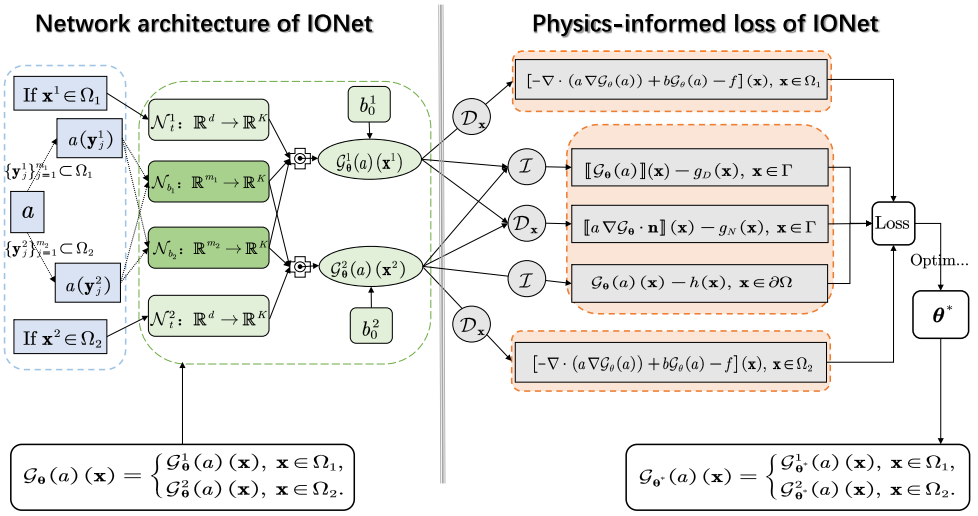


Fig. 2. A schematic diagram of the IONet for solving the parametric elliptic interface problem by minimizing the physics-informed loss function. Here, the input function is the coefficient $a(\mathbf{x})$.

3.1. Network architecture of IONet

To preserve the inherent discontinuity of interface problems, we decompose the computational domain into two subdomains according to the interface and leverage two operator networks that share some parameters, each of which is responsible for the solution in one subdomain. In particular, the IONet architecture is given as follows:

$$\mathcal{G}_\theta(a)(\mathbf{x}) = \begin{cases} \mathcal{G}_\theta^1(a)(\mathbf{x}), & \text{if } \mathbf{x} \in \Omega_1, \\ \mathcal{G}_\theta^2(a)(\mathbf{x}), & \text{if } \mathbf{x} \in \Omega_2, \end{cases} \tag{2}$$

where a is the input function and $\mathbf{x}$ denotes the location where the output function is evaluated. Note that input functions are discretized and evaluated at a set of sensors typically. To retain the irregularity of the input function on the interface, we divide the set of sensors according to the interface and use two branch nets, denoted as $\mathcal{N}_{b_1}$ and $\mathcal{N}_{b_2}$, to extract latent representations of input functions on the corresponding subdomains. Similar to the vanilla DeepONet [48], within each suboperator $\mathcal{G}_\theta^i$, we use a trunk net denoted as $\mathcal{N}_t^i$ to extract continuous input coordinates where the output functions are evaluated. Finally, following the MIONet [50], we merge the outputs of all the sub-networks through a Hadamard product and a summation, followed by the addition of a bias in the last stage. More specifically, the suboperator in $\mathcal{G}_\theta$ (2) is constructed as follows:

$$\begin{aligned} \mathcal{G}_\theta^i(a)(\mathbf{x}) &= S\left(\underbrace{\mathcal{N}_{b_1}(a(\mathbf{y}_1^1), \cdots, a(\mathbf{y}_{m_1}^1))}_{branch_1} \odot \underbrace{\mathcal{N}_{b_2}(a(\mathbf{y}_1^2), \cdots, a(\mathbf{y}_{m_2}^2))}_{branch_2} \odot \underbrace{\mathcal{N}_t^i(\mathbf{x})}_{trunk} \right) + \underbrace{b_0^i}_{bias} \\ &= \sum_{k=1}^{K} t_k^i b_{1k} b_{2k} + b_0^i. \end{aligned} \tag{3}$$

Here, θ denotes the trainable parameters in this architecture. For $i = 1, 2$, $\{\mathbf{y}_j^i\}_{j=1}^{m_i}$ represents the collection of sensors for evaluating $a(\mathbf{x})$ in subdomain Ω_i, $[b_{i1}, b_{i2}, \cdots, b_{iK}]$ and $[t_1^i, t_2^i, \cdots, t_K^i]$ denote the output features of the branch nets $\mathcal{N}_{b_i}$ and the trunk net $\mathcal{N}_t^i$, respectively. The network architecture of IONet is schematically visualized on the left side of Fig. 2. To demonstrate the capability and performance alone, we apply the simplest feedforward neural networks (FNNs) as the branch and trunk nets in this paper, and we note that other neural networks such as ResNet and CNN can be chosen as the sub-networks in IONet according to specific problems.

The IONet structure can be easily generalized to a multi-domain scenario. Next, we show that IONet is able to approximate arbitrary continuous operators with discontinuous inputs and outputs. For later analysis, we define the following space

$$X(\Omega) = \bigcap_{i=1}^{I} H^2(\Omega_i) \bigcap H^0(\Omega)$$

equipped with the norm

$$\|u\|_{X(\Omega)} = \sum_{i=1}^{I} \|u\|_{H^2(\Omega_i)}.$$

Then, the approximation theorem of IONet is given as follows.

Theorem 1. *Let $\Omega \subset \mathbb{R}^d$ be a bounded domain, Ω_i with $i = 1, \cdots, I-1$ be disjoint open domains and $\Omega_I = \Omega \setminus \bigcup_{i=1}^{I-1} \Omega_i$. Assume $\mathcal{G} : \bigcap_{i=1}^{I} C(\Omega_i) \bigcap L^\infty(\Omega) \to X(\Omega)$ is a continuous operator and $T \subset \bigcap_{i=1}^{I} C(\Omega_i) \bigcap L^\infty(\Omega)$ is a compact set. Then for any $\varepsilon > 0$, there exist positive integers m_i, K, tanh FNNs $\mathcal{N}_{b_i} : \mathbb{R}^{m_i} \to \mathbb{R}^K$, $\mathcal{N}_t^i : \mathbb{R}^d \to \mathbb{R}^K$, and $\mathbf{y}_1^i, \cdots, \mathbf{y}_{m_i}^i \in \Omega_i$ with $i = 1, \cdots, I$, such that*

$$\sup_{a \in T} \left\| \mathcal{G}(a)(\cdot) - \mathcal{S}\left(\mathcal{N}_{b_1}(a(\mathbf{y}_1^1), \cdots, a(\mathbf{y}_{m_1}^1)) \odot \cdots \odot \mathcal{N}_{b_I}(a(\mathbf{y}_1^I), \cdots, a(\mathbf{y}_{m_I}^I)) \odot \mathcal{N}_t^i(\cdot) \right) \right\|_{H^2(\Omega_i)} \le \varepsilon,$$

where $\mathcal{S}$ is the summation of all the components of a vector, and $\odot$ is the Hadamard product.

Proof. The proof can be found in Appendix A. □

Remark 1. There exist various continuous operator $\mathcal{G} : \bigcap_{i=1}^{I} C(\Omega_i) \bigcap L^\infty(\Omega) \to X(\Omega)$. For example, for interface problem (1), the operator mapping from the source term $f \in \bigcap_{i=1}^{I} C(\Omega_i) \bigcap L^\infty(\Omega)$ to the solution $u \in X(\Omega)$ is continuous, due to the estimate [56] that $\|u\|_X \le C \|f\|_{L^2(\Omega)} \le C \sum_{i=1}^{I} \|f\|_{C(\Omega_i)}$.

3.2. Loss function of IONet

Similar to DeepONet, a data-driven approach can be used to train IONet and optimize the parameters θ by minimizing the following mean square error loss:

$$L_{operator}(\theta) = \frac{1}{N_o P_o} \sum_{n=1}^{N_o} \sum_{p=1}^{P_o} \left| \mathcal{G}_\theta(a_o^n)(\mathbf{x}_{n,p}^o) - \mathcal{G}(a_o^n)(\mathbf{x}_{n,p}^o) \right|^2, \tag{4}$$

where $\{a_o^n\}_{n=1}^{N_o}$ denotes N_o input functions sampled from the parameter space; for $n = 1, \cdots, N_o$, the training data points $\{\mathbf{x}_{n,p}^o\}_{p=1}^{P_o} \subset \Omega$ denotes the set of locations to evaluate the output function and can be set to vary for different n; $\mathcal{G}(a_o^n)(\mathbf{x}_{n,p}^o)$ and $\mathcal{G}_\theta(a_o^n)(\mathbf{x}_{n,p}^o)$ are evaluated values of the output functions of the solution operator $\mathcal{G}$ and IONet $\mathcal{G}_\theta$ at location $\mathbf{x}_{n,p}^o$ when a_o^n is the input function. This type of training method relies on the assumption that there is sufficient labeled data

$$\left\{ \left(a_o^n(\mathbf{y}_1^1), \cdots, a_o^n(\mathbf{y}_{m_1}^1),\ a_o^n(\mathbf{y}_1^2), \cdots, a_o^n(\mathbf{y}_{m_2}^2),\ \mathbf{x}_{n,p}^o,\ \mathcal{G}(a_o^n)(\mathbf{x}_{n,p}^o) \right) \right\}_{n=1,\cdots,N_o,\ p=1,\cdots,P_o}$$

to train the model. However, the costs associated with experimental data acquisition and high-quality numerical simulation are generally expensive. In many practical scenarios, we are inevitably faced with limited or even intractable training data. In the following, we introduce the physics-informed loss function for IONet.

By the definitions of IONet (2) and (3), the output function of $\mathcal{G}_\theta$, such as

$$\mathcal{G}_\theta(a^n)(\mathbf{x}) = \begin{cases} \mathcal{S}\left(\mathcal{N}_{b_1}(a^n(\mathbf{y}_1^1), \cdots, a^n(\mathbf{y}_{m_1}^1)) \odot \mathcal{N}_{b_2}(a^n(\mathbf{y}_1^2), \cdots, a^n(\mathbf{y}_{m_2}^2)) \odot \mathcal{N}_t^1(\mathbf{x}) \right) + b_0^1, & \text{if } \mathbf{x} \in \Omega_1, \\ \mathcal{S}\left(\mathcal{N}_{b_1}(a^n(\mathbf{y}_1^1), \cdots, a^n(\mathbf{y}_{m_1}^1)) \odot \mathcal{N}_{b_2}(a^n(\mathbf{y}_1^2), \cdots, a^n(\mathbf{y}_{m_2}^2)) \odot \mathcal{N}_t^2(\mathbf{x}) \right) + b_0^2, & \text{if } \mathbf{x} \in \Omega_2, \end{cases}$$

has a continuous representation in each subdomain. Provided that the trunk net $\mathcal{N}_t^i$ are smooth enough, the derivatives of $\mathcal{G}_\theta(a^n)$ at $\mathbf{x}_{n,p}^i$ can be easily obtained by automatic differentiation [55]. Inspired by PINN [57] and PI-DeepONet, for given parametric elliptic interface problems, we define

$$L_{r_i}(\theta) := \sum_{n=1}^{N} \sum_{p=1}^{P_i} \left| -\nabla \cdot \left(a^n(\mathbf{x}_{n,p}^i) \nabla \mathcal{G}_\theta(a^n)(\mathbf{x}_{n,p}^i) \right) + b(\mathbf{x}_{n,p}^i) \mathcal{G}_\theta(a^n)(\mathbf{x}_{n,p}^i) - f(\mathbf{x}_{n,p}^i) \right|^2 \tag{5}$$

and

$$L_b(\theta) := \sum_{n=1}^{N} \sum_{p=1}^{P_b} \left| \mathcal{G}_\theta(a^n)(\mathbf{x}_{n,p}^b) - h(\mathbf{x}_{n,p}^b) \right|^2,$$

where $\{\mathbf{x}_{n,p}^i\}_{p=1}^{P_i}$ with $i = 1, 2$ and $\{\mathbf{x}_{n,p}^b\}_{p=1}^{P_b}$ are randomly sampled from the subdomain Ω_i and its boundary $\partial\Omega$, respectively. Let $L_\Gamma(\theta) = L_{\Gamma_D} + L_{\Gamma_N}$, where

$$L_{\Gamma_D}(\theta) = \sum_{n=1}^{N} \sum_{p=1}^{P_\gamma} \left| [\![\mathcal{G}_\theta(a^n)]\!](\mathbf{x}_{n,p}^\gamma) - g_D(\mathbf{x}_{n,p}^\gamma) \right|^2 = \sum_{n=1}^{N} \sum_{p=1}^{P_\gamma} \left| \mathcal{G}_\theta^2(a^n)(\mathbf{x}_{n,p}^\gamma) - \mathcal{G}_\theta^1(a^n)(\mathbf{x}_{n,p}^\gamma) - g_D(\mathbf{x}_{n,p}^\gamma) \right|^2,$$

and

$$L_{\Gamma_N}(\theta) = \sum_{n=1}^{N}\sum_{p=1}^{P_\gamma}\left|[\![a^n\nabla\mathcal{G}_\theta(a^n)\cdot\mathbf{n}]\!](\mathbf{x}_{n,p}^\gamma) - g_N(\mathbf{x}_{n,p}^\gamma)\right|^2$$
$$= \sum_{n=1}^{N}\sum_{p=1}^{P_\gamma}\left|a_2^n(\mathbf{x})\nabla\mathcal{G}_\theta^2(a^n)(\mathbf{x}_{n,p}^\gamma)\cdot\mathbf{n} - a_1^n(\mathbf{x})\nabla\mathcal{G}_\theta^1(a^n)(\mathbf{x}_{n,p}^\gamma)\cdot\mathbf{n} - g_N(\mathbf{x}_{n,p}^\gamma)\right|^2.$$

Here, $\{\mathbf{x}_{n,p}^\gamma\}_{p=1}^{P_\gamma}$ represents a set of training points sampled from the interface Γ for the n-th input function. Then, the physics-informed loss function for IONet is formulated as follows:

$$L_{physics}(\theta) = \lambda_1 L_{r_1}(\theta) + \lambda_2 L_{r_2}(\theta) + \lambda_3 L_\Gamma(\theta) + \lambda_4 L_b(\theta), \tag{6}$$

where $L_{r_i}(\theta)$ (5) with $i = 1, 2$ are to approximately restrict the IONet output function to obey the given governing PDE (1a), while $L_b(\theta)$ and $L_\Gamma(\theta)$ penalize IONet for violating the boundary condition (1d) and the interface conditions (1b) and (1c), respectively. Such physics-informed loss function of IONet is schematically depicted on the right side of Fig. 2. By incorporating physics constraints to ensure that the IONet output function aligns with the given interface PDE (1), the proposed IONet can effectively learn the solution operator for parametric interface problems, even in the absence of labeled training data (excluding boundary and interface conditions). If both data and PDEs are available, we combine the loss functions (4) and (6) and minimize the following composite loss function to obtain the parameter θ of IONet:

$$L(\theta) = \lambda_p L_{physics}(\theta) + \lambda_o L_{operator}(\theta). \tag{7}$$

4. Numerical Results

In this section, the proposed IONet is tested on a range of parametric elliptic interface problems. Throughout all benchmarks, the branch nets and the trunk nets are FNNs. Particularly when $L_{physics}$ is included in (7), i.e., $\lambda_p \neq 0$, they are FNNs with smooth activation function Tanh, due to the necessity for high-order derivatives. All operator network models are trained via stochastic gradient descent using Adam optimizer with default settings. The learning rate is set to exponential decay with a decay-rate of 0.95 per #Epochs/100 iterations, where #Epochs denotes the maximum number of optimization iterations. Unless otherwise specified, the training data points used to evaluate the loss function are scattered points randomly sampled in the computational domain, while those used to evaluate the output solution of neural models are equidistant grid points. After training, the average relative L^2 error between the reference solution operator $\mathcal{G}$ and the numerical solution operator $\mathcal{G}_\theta$ is measured as

$$L^2(\mathcal{G},\mathcal{G}_\theta) = \frac{1}{N}\sum_{n=1}^{N}\sqrt{\frac{\int_\Omega\left|\mathcal{G}(a_{test}^n)(x) - \mathcal{G}_\theta(a_{test}^n)(x)\right|^2 dx}{\int_\Omega\left|\mathcal{G}(a_{test}^n)(x)\right|^2 dx}},$$

where N denotes the number of test input functions $\{a_{test}^n\}_{n=1}^N$, and the integration is computed by the Monte Carlo method. For simplicity, IONet using the loss function (7) with $\lambda_p = 1$ and $\lambda_o = 0$ is referred to as "PI-IONet", while IONet using the loss function (7) with $\lambda_p = 0$ and $\lambda_o = 1$ is denoted as "DD-IONet". All experiments are tested on one NVIDIA Tesla V100 GPU. The code and experimental data used in this paper are publicly available from the GitHub repository https://github.com/bzlu-Group/IONet.

4.1. Parametric elliptic interface problems in one dimension

Example 1. As the first example, we investigate the effectiveness of the proposed method in handling non-zero interface conditions in the elliptic interface problem (1), defined on the interval $\Omega = [0, 1]$ with an interface point at $x_\gamma = 0.5$:

$$\begin{aligned} -\nabla\cdot(a(x)\nabla u(x)) &= 0, \quad x \in \Omega, \\ g_D(x_\gamma) = 1, g_N(x_\gamma) &= 0, \\ u(0) = 1,\ u(1) &= 0. \end{aligned} \tag{8}$$

Here, our goal is to learn a solution operator $\mathcal{G}$ that maps the discontinuous coefficient function $a(x)$ to the latent solution $u(x)$ that is explicitly discontinuous across the interface.

To make the input function

$$a(x) = \begin{cases} a_1(x), & x \in \Omega_1 := [0, 0.5] \\ a_2(x), & x \in \Omega_2 := (0.5, 1] \end{cases}$$

strictly positive, we let $a_i(x) = \tilde{a}_i(x) - \min_x \tilde{a}_i(x) + 1$ with $i = 1, 2$, where $\tilde{a}_i(x)$ is randomly sampled from a mean-zero Gaussian random field (GRF) with a radial basis function (RBF) kernel

Table 1
Test errors and training costs for DD-DeepONet, DD-IONet, PI-DeepONet, and PI-IONet. The error corresponds to the relative L^2 error, recorded in the form of mean ± standard deviation based on all test input functions in Example 1.

Models	Activation	Depth	Width	#Parameters	$L^2(\mathcal{G}_\theta, \mathcal{G})$	Training time (hours)
DD-DeepONet	ReLU	5	140	172 K	9.82e-2±1.71e-2	0.10
DD-IONet	ReLU	5	100	172 K	3.95e-3±1.43e-3	0.17
PI-DeepONet	Tanh	5	140	172 K	4.70e-1±1.02e-1	0.36
PI-IONet	Tanh	5	100	172 K	8.30e-3 ±7.92e-3	0.44

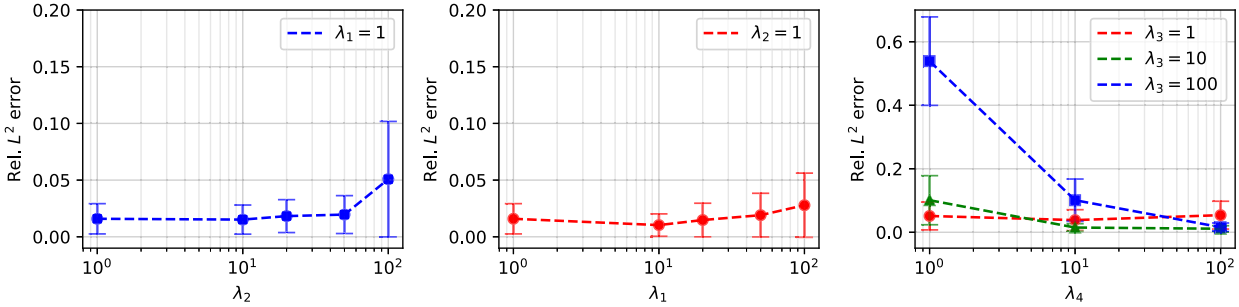


Fig. 3. The mean and one standard deviation of relative L^2 error for PI-IONet with different weights in the physics-informed loss function (6). Left: $\lambda_1 = 1$ and $\lambda_3 = \lambda_4 = 100$. Middle: $\lambda_2 = 1$ and $\lambda_3 = \lambda_4 = 100$. Right: $\lambda_1 = \lambda_2 = 1$.

$$k_l(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{2l^2}\right)$$

using a length scale $l = 0.25$ (see the left panel of Fig. 4 for an illustration). We randomly sample 10,000 and 1,000 input functions $a(x)$ for training and testing, respectively. The sensor of the input function consists of 100 equidistant grid points in the interval $[0, 1]$. For each input function, we solve Eq. (8) on a uniform mesh of size 1000 using the matched interface and boundary (MIB) method with second-order accuracy [24] to obtain the reference solution and paired input-output training data. The test error of all neural models is measured on the same mesh of size 1000.

In this example, we investigate the performance of DD-IONet and PI-IONet as well as two state-of-the-art neural models, namely vanilla DeepONet (DD-DeepONet) [48] and physics-informed DeepONet (PI-DeepONet) [52], in solving Eq. (8) with variable coefficients $a(x)$. For PI-DeepONet, the neural network $\mathcal{G}_\theta$ is trained by minimizing a physics-informed loss function of the form $L_{physics}(\theta) = \lambda_r L_r(\theta) + \lambda_3 L_\Gamma(\theta) + \lambda_4 L_b(\theta)$, where $L_\Gamma = L_{\Gamma_D} + L_{\Gamma_N}$, $\lambda_r = (\lambda_1 + \lambda_2)/2$, and weights λ_i with $i = 1, 2, 3$ and 4 are those in PI-IONet. Note that the output function space of DeepONet is a subset of the space of continuous functions, implying that $[\![\mathcal{G}_\theta(a)(x_\gamma)]\!] = 0$ holds for any input function a. Hence, we approximate the interface loss functions L_{Γ_D} and L_{Γ_N} in PI-DeepONet by difference schemes

$$L_{\Gamma_D}(\theta) = \sum_{n=1}^{N} \left| \mathcal{G}_\theta(a^n)(x^\gamma + \epsilon) - \mathcal{G}_\theta(a^n)(x^\gamma - \epsilon) - 1 \right|^2$$

and

$$\begin{aligned} L_{\Gamma_N}(\theta) &= \sum_{n=1}^{N} \left| [\![a^n \nabla \mathcal{G}_\theta(a^n) \cdot \mathbf{n}]\!](x^\gamma) \right|^2 \\ &= \sum_{n=1}^{N} \left| a_2^n(x^\gamma) \frac{\mathcal{G}_\theta(a^n)(x^\gamma + \epsilon) - \mathcal{G}_\theta(a^n)(x^\gamma)}{\epsilon} - a_1^n(x^\gamma) \frac{\mathcal{G}_\theta(a^n)(x^\gamma) - \mathcal{G}_\theta(a^n)(x^\gamma - \epsilon)}{\epsilon} \right|^2 \end{aligned} \tag{9}$$

with $\epsilon = 10^{-5}$ in practice. In all cases, the neural networks are trained after 4×10^4 parameter updates. The network architecture details and training costs are shown in Table 1.

We first investigate the effect of the weights in loss function (6) on the accuracy of PI-IONet. It can be observed in Fig. 3 that the weights of interface and boundary loss terms (i.e. λ_3 and λ_4) have more significant impacts on the numerical results compared with those of PDE residuals (i.e., λ_1 and λ_2). This phenomenon could be attributed to the fact that the investigated interface problem exhibits low-contrast. In the following, we fix the weights in physics-informed loss function of PI-IONet as $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = 10$, and $\lambda_4 = 100$.

Table 1 reports the relative L^2 errors between the reference solution and the numerical solution for DD-DeepONet, DD-IONet, PI-DeepONet, and PI-IONet. Under different training frameworks, it can be observed that the accuracy of DD-IONet is significantly superior to that of DD-DeepONet with the same paired input-output training data, while the error of PI-DeepONet is about 50 times

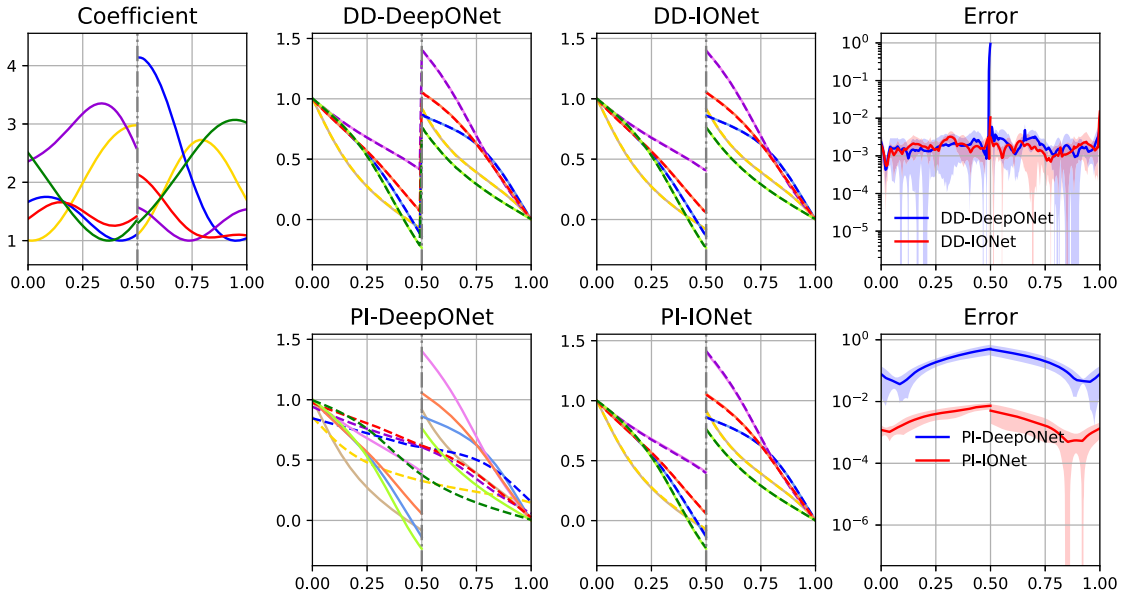


Fig. 4. Left column: Five input functions randomly selected from the test set (distinguished by different colors). Second and third columns: The reference solutions (solid lines) versus the numerical solutions (dashed lines) of DD-DeepONet, DD-IONet, PI-DeepONet, and PI-IONet. Fourth column: The mean and one standard deviation of the numerical solutions, averaged over these 5 test examples.

that of PI-IONet when trained by minimizing the physics-informed loss functions. Furthermore, one can also see that DD-IONet outperforms PI-IONet, as the latter is trained without relying on any high-quality paired training data but instead through solving a highly complex optimization problem involving derivatives. In terms of the training time for neural models, as shown in this table, training physics-informed models (PI-DeepONet and PI-IONet) generally takes longer than training data-driven models (DD-DeepONet and DD-IONet). It is mainly due to the fact that physics-informed models require the computation of the PDE and interface residuals via automatic differentiation, and the loss terms are computed in a serial manner.

Fig. 4 shows a comparison between the reference and the numerical solutions for five randomly sampled input functions from the test dataset. The second column gives the numerical results of DD-DeepONet and PI-DeepONet. It can be observed that the numerical solution of DD-DeepONet agrees well with the reference solution away from the interface ($x_\Gamma = 0.5$), but there are large errors near the interface (refer to the error plot in the first row), while PI-DeepONet fails to yield accurate results. The main reason is due to the fact that the output function space of DeepONet is a subset of continuous function space, which limits its effectiveness in capturing discontinuities in solution functions to interface problems. In addition, we remark that DD-DeepONet is trained with high-quality paired training data using least squares regression, enabling the continuous output function of DeepONet to approximate the discontinuous output of the target operator point by point. However, PI-DeepONet employs a physics-informed loss function involving derivatives and the loss of interface conditions is approximated using difference schemes, exacerbating the challenge of handling discontinuities. This results in the numerical accuracy of DD-DeepONet being superior to that of PI-DeepONet. The numerical results for DD-IONet and PI-IONet are displayed in the third column, where it can be seen that the numerical solutions naturally maintain the discontinuous nature of the numerical solution at the interface, demonstrating excellent agreement with the reference solutions (see the error plot in the fourth column). These numerical results show that IONet is more adaptable to the irregularities in the input function and the solution, which exhibits a greater ability to represent the solutions of interface problems compared with the conventional DeepONet architecture.

Next, we show that IONet is able to be extended to finite number of subdomains (greater than two). Consider Eq. (8) with three subdomains, i.e.,

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= 0, \quad x \in \Omega, \\ g_D(x_{\gamma_1}) = 1, g_N(x_{\gamma_1}) &= 0, \\ g_D(x_{\gamma_2}) = -\frac{1}{2}, g_N(x_{\gamma_2}) &= 0, \\ u(0) = 1,\ u(1) &= 0, \end{aligned} \tag{10}$$

where $x_{\gamma_1} = 0.3$ and $x_{\gamma_1} = 0.7$.

We approximate the solution operator $\mathcal{G}$ of Eq. (10) by PI-IONet. The chosen model architecture and other hyperparameters remain consistent with those used for two subdomain scenarios, except for the use of three branch nets and three trunk nets. Fig. 5 shows the numerical results of PI-IONet. The final mean of relative L^2 error over five test input functions in this figure is measured at 3.83×10^{-3}. These results demonstrate that IONet also performs well in multi-subdomain scenarios.

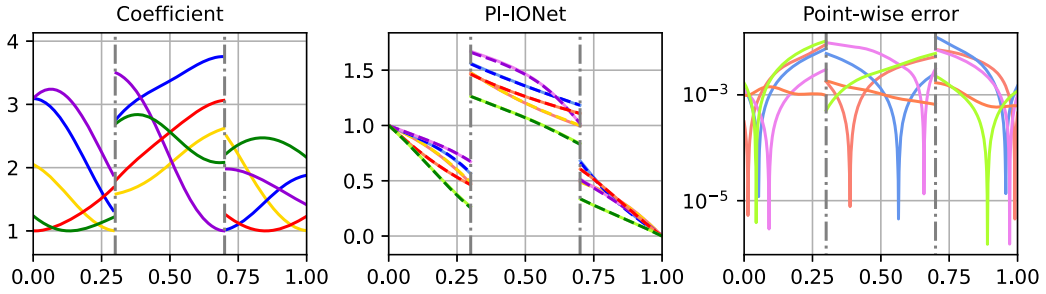


Fig. 5. Left: Five input functions randomly selected from the test set (distinguished by different colors). Middle: Reference solutions (solid lines) versus the numerical solutions (dashed lines) of PI-IONet. Right: Absolute point-wise errors over the whole domain. The gray point-dashed lines represent the location of the interfaces.

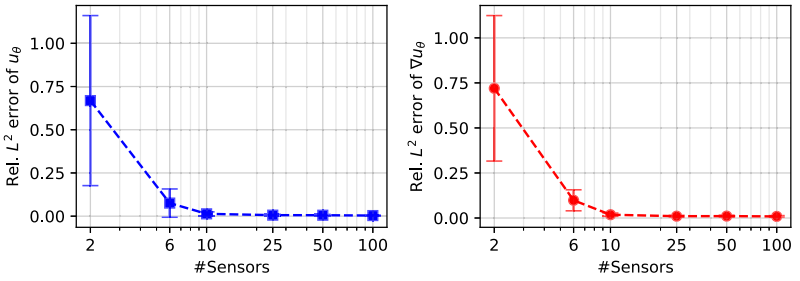


Fig. 6. The mean and one standard deviation of test L^2 errors for PI-IONet with varying numbers of sensors.

Example 2. Next, we consider Eq. (1) with zero interface conditions, defined on $\Omega = [0,1]$ with an interface point at $x^\gamma = 0.5$:

$$\begin{aligned} -\nabla\cdot(a(x)\nabla u(x)) + b(x)u(x) &= f(x), \quad x \in \Omega, \\ g_D(x_\gamma) = 0,\ g_N(x_\gamma) &= 0, \\ u(0) = h_0,\ u(1) &= h_1. \end{aligned} \tag{11}$$

From the setting of the interface conditions, we know that the latent solution u to Eq. (11) is continuous, while its derivative may be discontinuous on the interface due to the different values of a in different subdomains. In this example, our goal is to learn an operator $\mathcal{G}$ that maps the source term $f(x)$, boundary condition $h(x)$, and coefficients $a(x)$ and $b(x)$ to the solution u, i.e.,

$$\mathcal{G} : (f(x), h(x), a(x), b(x)) \to u(x). \tag{12}$$

To obtain the dataset, we randomly sample 10,000 and 1,000 sets of input functions (f, h, a, b) for training and testing, respectively. Specifically, the input functions $f_1 := f|_{\Omega_1}$ and $f_2 := f|_{\Omega_2}$ are independently sampled from a zero-mean GRF with length scales $l_1 = 0.2$ and $l_1 = 0.1$, respectively. The coefficients $b_i := b|_{\Omega_i}$ with $i = 1, 2$ are independently sampled via $b(x) = \tilde{b}(x) - \min_x \tilde{b}(x) + 1$, where $\tilde{b}(x)$ is randomly sampled from a zero-mean GRF with length scale $l = 0.25$. The coefficient $a(x)$ is modeled as a piece-wise constant function, with $a|_{\Omega_1} = a_1$ and $a|_{\Omega_2} = a_2$, where a_1 and a_2 are sampled from uniform distributions over the intervals $[0.5, 1]$ and $[2, 3]$, respectively. Additionally, we sample h_0 and h_1 from uniform distributions over the intervals $[-0.1, 0]$ and $[0, 0.1]$, respectively, to set the variable boundary conditions $h(0) = h_0$ and $h(1) = h_1$. In this setting, it is easy to verify the existence and uniqueness of the solution to problem (11). For each set of input, the reference solution $u(x)$ is obtained by the MIB method with a uniform mesh of size 1000. The test error of all neural models is measured on the same mesh of size 1000.

In this case, we investigate the performance of PI-IONet and PI-DeepONet in solving Eq. (11) without any paired input-output training data. In order to adapt to multiple input functions, for PI-IONet, we first divide the set of sensors $\{y_i\}_{i=1}^n$ into two subsets $\{y_i^1\}_{i=1}^{n_1}$ and $\{y_i^2\}_{i=1}^{n_2}$ with $n_1 + n_2 = n$ according to the interface. Then we employ 8 branch nets to extract latent representations of the 4 input functions on the corresponding sub-domain. Specifically, the inputs for branch nets are $[f(y_1^1), \cdots, f(y_{n_1}^1)]$, $[f(y_1^2), \cdots, f(y_{n_2}^2)]$, $[b(y_1^1), \cdots, b(y_{n_1}^1)]$, $[b(y_1^2), \cdots, b(y_{n_2}^2)]$, $[a_1]$, $[a_2]$, $[h_0]$ and $[h_1]$, respectively. For PI-DeepONet, we employ the most direct approach by concatenating the four input functions together, i.e., $[f(y_1), \cdots, f(y_n), b(y_1), \cdots, b(y_n), a_1, a_2, h_0, h_1]$, to serve as a single input for the branch net.

We first utilize the PI-IONet network architecture, employing branch and trunk nets consisting of 5-layer FNN with 70 units per layer, to approximate the solution operator (12). The neural network is trained by minimizing the loss function (6) with $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = 10$, and $\lambda_4 = 100$, over 4×10^4 iterations of optimization. Fig. 6 illustrates the variation in relative L^2 errors of the numerical solution and its derivative with respect to the number of sensors (denoted as #Sensors). Note that, in each case, the set of sensors

Table 2
Test errors and training costs for PI-IONet and PI-DeepONet. The error corresponds to the relative L^2 error, recorded in the form of mean ± standard deviation based on all test input functions in Example 2.

Model	Activation	Depth	Width	#Parameters	$L^2(\mathcal{G}_\theta, \mathcal{G})$	$L^2(\nabla\mathcal{G}_\theta, \nabla\mathcal{G})$	Training time (hours)
PI-IONet	Tanh	5	70	213 K	3.48e-3± 3.05e-3	8.84e-3±4.51e-3	0.76
PI-DeepONet	Tanh	5	151	214 K	2.96e-1± 1.99e-1	3.06e-1± 1.97e-1	0.30

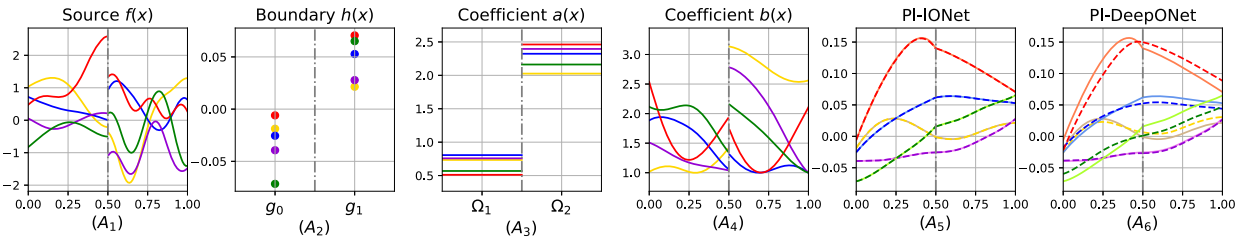


Fig. 7. An illustration of randomly sampled input functions (distinguished by different colors): Source term f (A_1), boundary condition h (A_2), coefficient a (A_3) and coefficient b (A_4); and the corresponding reference solutions (solid lines) obtained by MIB method and numerical solutions (dashed lines) obtained by PI-IONet (A_5) and PI-DeepONet (A_6). The gray point-dashed lines represent the location of the interface.

consists of equidistant grid points in the interval $[0, 1]$. As we can see from this figure, the test errors generally decrease as #Sensors increases until it is sufficient to capture all the necessary frequency information for the input function.

In the following experiments, we keep #Sensors = 100, while keeping other hyperparameters consistent with those used in PI-IONet. Here, $L_{\Gamma_D}(\theta) = 0$ holds for any output function of PI-DeepONet, and L_{Γ_N} is approximated using Eq. (9). Table 2 records the network structures and the test errors for two neural models. As can be seen from the table, the final relative L^2 errors of the numerical solutions and their derivatives obtained by PI-IONet can be of the order of 10^{-3}. Some visualizations of the input function, the reference solution, and the numerical result are shown in Fig. 7. As can be seen from the figure, the numerical solution obtained by PI-IONet aligns more consistently with the reference solution, even in the case where the ground truth is continuous. These results suggest that IONet is able to effectively handle multiple inputs for parametric interface problems.

It is remarked that other advantages of DeepONet also hold true for IONet, for example, the capability of providing accurate predictions for out-of-distribution test data [52,58]. Fig. 8 illustrates the numerical results of PI-IONet and PI-DeepONet for regenerated test input functions. In this study, we not only use GRF to generate the input functions (see the first row), but also include two certain functions, i.e., $\exp(x)$ and $\sin(x)$, as input functions (see the second and third rows). As shown in the third and fourth columns of this figure, both PI-IONet and PI-DeepONet integrate the constraints of physical laws as well as boundary conditions directly into model training, enabling the models to capture the fundamental behavior of the system. Remarkably, PI-IONet handles discontinuities in the input functions and output solutions across the interface more effectively, resulting in more accurate numerical results. Specifically, the average relative L^2 errors of PI-IONet in the first to third rows are measured at 4.32×10^{-3}, 8.50×10^{-3} and 2.01×10^{-2}, respectively. These results further underscore the robustness and generalization capability of PI-IONet.

4.2. Parametric elliptic interface problems in two dimensions

Example 3. To further investigate the capability of IONet, we consider a parametric interface problem (1) with a sharp and complicated interface Γ which is given as

$$x_1(\vartheta) = 0.65\cos(\vartheta)^3,$$
$$x_2(\vartheta) = 0.65\sin(\vartheta)^3,\ 0 \le \vartheta \le \pi.$$

Here, the source term f is the input parameter of the target solution operator. In this example, we model the input function in the following way:

$$f_i(\mathbf{x}) := f(\mathbf{x})\Big|_{\Omega_i} = \frac{p_1^i}{[1 + 10(x_1^2 + x_2^2)]^2} - \frac{p_2^i(x_1^2 + x_2^2)}{[1 + 10(x_1^2 + x_2^2)]^3},\ i = 1, 2,$$

where (p_1^i, p_2^i) comes from $[50, 100] \times [1550, 1650]$. The computational domain is a regular square $\Omega = [-1, 1]^2$ (see Fig. 9 for an illustration). The coefficient $a(\mathbf{x})$ is a piece-wise constant, which is given by $a(\mathbf{x})|_{\Omega_1} = 2$ and $a(\mathbf{x})|_{\Omega_2} = 1$. The interface conditions on Γ are set as

$$g_D(\mathbf{x}) = \frac{1}{1 + 10(x_1^2 + x_2^2)},$$
$$g_N(\mathbf{x}) = 0,$$

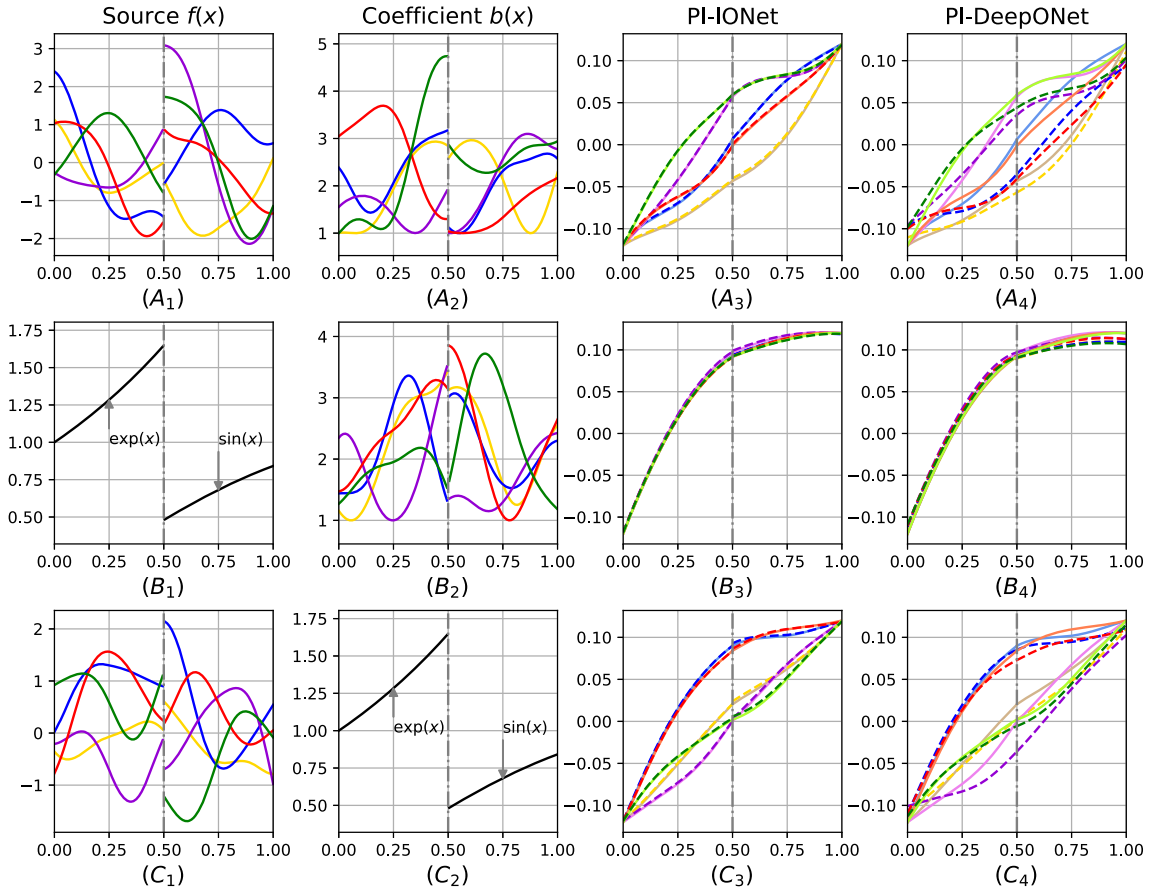


Fig. 8. Numerical results of PI-IONet and PI-DeepONet for out-of-distribution test input samples. Here, we keep $a_1 = 1.2$, $a_2 = 1.8$, $h_0 = -0.12$ and $h_1 = 0.12$. The input functions in (A_1), (A_2), (B_2), and (C_1) are generated using GRF with length scale $l = 0.15$. Input functions f_1 in (B_1) and b_1 in (C_2) are fixed as $\exp(x)$, while f_2 in (B_1) and b_2 in (C_2) are fixed as $\sin(x)$. The reference solutions (solid lines) and numerical solutions (dashed lines) obtained by PI-IONet and PI-DeepONet are shown in the third and fourth columns. The gray point-dashed lines represent the location of the interface.

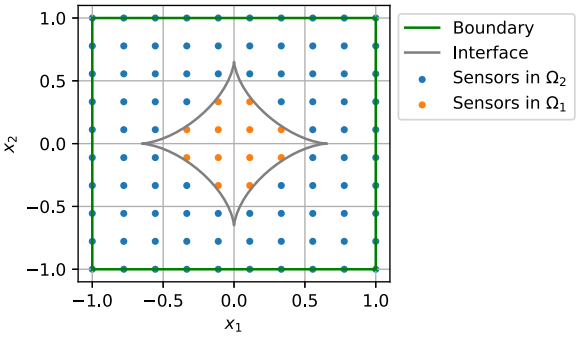


Fig. 9. Computational domain and sensor locations in Example 3. Here, the number of sensors is 100.

and $h(\mathbf{x})$ on boundary $\partial\Omega$ is given as

$$h(\mathbf{x}) = \frac{2}{1 + 10(x_1^2 + x_2^2)}.$$

One can observe that if we take $f_1(\mathbf{x}) = f_2(\mathbf{x})$ with $(p_1^1, p_2^1) = (p_1^2, p_2^2) = (80, 1600)$, then Eq. (1) has following exact solution [33]

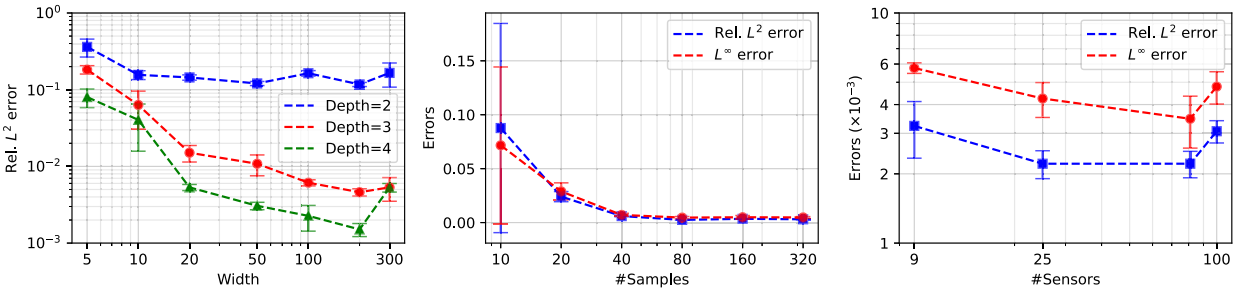


Fig. 10. The mean and one standard deviation of test errors of PI-IONet in Example 3. (Left) Training PI-IONet using different depth and width of the network architecture, where #Sensors = 100 and #Samples = 320. (Middle) Training PI-IONet using different number of training samples, where width = 50, depth = 4 and #Sensors = 100. (Right) Training PI-IONet using different number of sensors, where width = 50, depth = 4 and #Samples = 320. These errors represent the average of 3 different runs corresponding to different set of training input functions and network initialization.

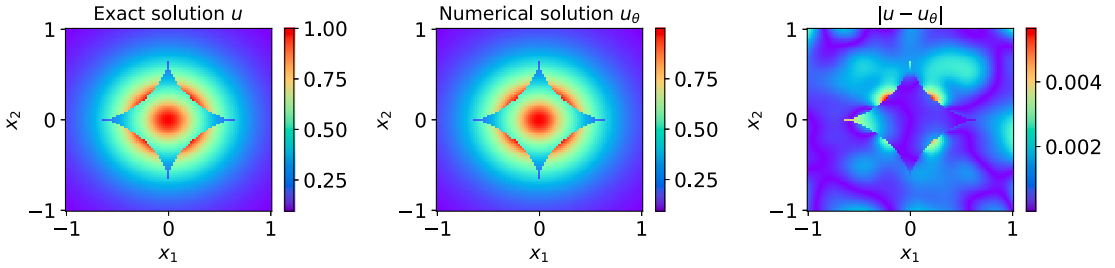


Fig. 11. The profile of the exact solution, the numerical solution obtained by PI-IONet, and the corresponding point-wise error are showcased from left to right. Here, width = 50, depth = 4, #Sensors = 100, and #Samples = 320.

$$u(\mathbf{x}) = \begin{cases} \dfrac{1}{1+10(x_1^2+x_2^2)}, & \text{in } \Omega_1, \\ \dfrac{2}{1+10(x_1^2+x_2^2)}, & \text{in } \Omega_2. \end{cases}$$

To this end, we randomly sample some input functions from the given data distribution, except for the case $(p_1^1, p_2^1) = (p_1^2, p_2^2) = (80, 1600)$, which is reserved for testing purposes. The sensors for the input functions in the whole domain are equidistant grid points in the square $[-1, 1]^2$. Take advantage of being mesh-free, IONet can easily handle problems with irregular interfaces.

In this study, we discuss the effects of the depth and width of the network, as well as the number of sensors and training samples, on the performance of the PI-IONet. We train PI-IONet by minimizing the physics-informed loss function (6) with $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = 10$ and $\lambda_4 = 100$ for 4×10^4 iterations of optimization. Fig. 10 illustrates the average error of three different runs of PI-IONet. Specifically, the left panel shows the relative L^2 error, i.e.

$$\frac{\|u - u_\theta\|_2}{\|u\|_2} = \sqrt{\frac{\sum_{i=1}^N \left(u(\mathbf{x}_i) - u_\theta(\mathbf{x}_i)\right)^2}{\sum_{i=1}^N u(\mathbf{x}_i)^2}},$$

between the exact solution u and the PI-IONet solution u_θ measured at $N = 101 \times 101$ test points over the whole domain, with varying depths (ranging from 2 to 4) and widths (ranging from 5 to 300) of the network architecture. We observe that increasing the expressiveness of the network leads to improved solution accuracy, eventually reaching a plain where the error reduction levels off. A similar trend is also observed on the middle panel as the number of training input samples (denoted as #Samples) increases. These might be caused by optimization errors. Additionally, as the input function $f(\mathbf{x})$ in this case is controlled by four parameters (i.e., p_1^i and p_2^i with $i = 1, 2$), a small number of sensors could be enough to capture all necessary frequency information of the input functions. As depicted on the right panel of this figure, although the accuracy of PI-IONet does not significantly improve with an increase in #Sensors, the average L^∞ error and relative L^2 error of PI-IONet with different #Sensors can both reach the order of 10^{-3}. Moreover, it can be seen from Fig. 11, the numerical solution is in excellent agreement with the exact solution, where the relative L^2 error is measured at 2.60×10^{-3}. These results demonstrate the consistent and reliable performance of IONet in generating accurate numerical results, even in scenarios where the interface is irregular.

Example 4. This example aims to investigate the performance of IONet in handling a two-dimensional parametric interface problem with variable boundary conditions and coefficients. Computational domain is defined as $\Omega := [0, 1]^2$, and the interface is defined as

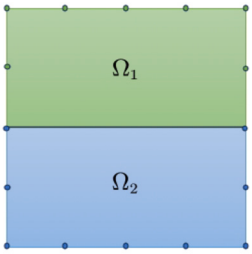


Fig. 12. Computational domain for Example 4. The black solid line indicates the interface location.

Table 3
Test errors and training costs for PI-IONet, DD-IONet and DD-DeepONet. The error corresponds to the relative L^2 error, recorded in the form of mean ± standard deviation based on all test input functions in Example 4. Here, #Sensors = 128 and #Samples = 1600.

Model	Activation	Depth	Width	#Parameters	$L^2(\mathcal{G}_\theta, \mathcal{G})$	Training time (hours)
PI-IONet	Tanh	5	120	365 K	5.03e-2±1.58e-2	1.26
DD-IONet	ReLU	5	120	365 K	1.70e-2±4.79e-3	0.47
DD-DeepONet	ReLU	5	206	368 K	7.56e-2±1.99e-2	0.20

$\Gamma := \{\mathbf{x} := (x_1, x_2) \mid x_2 = 0.5, \mathbf{x} \in \Omega\}$. Without losing generality, we defined $\Omega_1 := \{\mathbf{x} \mid x_2 > 0.5, \mathbf{x} \in \Omega\}$ and $\Omega_2 := \{\mathbf{x} \mid x_2 < 0.5, \mathbf{x} \in \Omega\}$ (see Fig. 12 for an illustration). Specifically, the interface problem takes the following specific form:

$$\begin{aligned} -\nabla \cdot (a \nabla u(\mathbf{x})) &= 0, && \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= h(\mathbf{x}), && \mathbf{x} \in \partial\Omega, \end{aligned} \tag{13}$$

with interface conditions $g_D = 0$ and $g_N = 0$. Our goal here is to learn the solution operator that maps the coefficient $a(x)$ and the boundary condition $h(\mathbf{x})$ to the solution $u(\mathbf{x})$ to Eq. (13), i.e.,

$$\mathcal{G} : (h(x), a(x)) \to u(x).$$

In this example, we randomly sample 3,200 and 100 pairs of input functions (h, a) for training and testing, respectively. Here, the boundary conditions $h_i(\mathbf{x}) := h(\mathbf{x})|_{\partial\Omega \cap \Omega_i}$ with $i = 1, 2$ are independently generated using GRF according to $h_i \sim \mu|_{\partial\Omega\cap\Omega_i}$, where $\mu \sim \mathcal{N}(0, 10^3(-\Delta + 100I)^{-4})$ with zero Neumann boundary conditions on the Laplacian,[1] while the coefficient $a(x)$ is modeled as a piece-wise constant function, where $a_1 := a|_{\Omega_1}$ and $a_2 := a_2|_{\Omega_2}$ are uniformly sampled from the intervals $[0.5, 1]$ and $[2, 3]$, respectively. For each pair of input, the reference solution is obtained by the $\mathbb{P}_1$ Lagrangian finite element method[2] on a uniform mesh of 1025 by 1025, while the test error of the numerical solution is measured at its 65 by 65 submesh.

In this study, to accommodate two input functions, the IONet architecture consists of four branch nets and two trunk nets, while the DeepONet architecture consists of one branch net and one trunk net. Specifically, the inputs for branch nets in IONet are $[h(y_1^1), \cdots, h(y_{n_1}^1)]$, $[h(y_1^2), \cdots, h(y_{n_2}^2)]$, $[a_1]$ and $[a_2]$, while for DeepONet it is $[h(y_1), \cdots, h(y_n), a_1, a_2]$, where $\{y_i\}_{i=1}^n = \{y_i^1\}_{i=1}^{n_1} \cup \{y_i^2\}_{i=1}^{n_2}$ is the set of sensors over the whole domain.

In the following experiments, the numerical results are recorded after 1×10^5 optimization iterations. The detailed network sizes are provided in Table 3. Fig. 13 illustrates the accuracy of PI-IONet, DD-IONet and DD-DeepONet with respect to the number of sensors or training samples. Specifically, the left panel of this figure depicts the variation of the relative L^2 error for PI-IONet, which is trained by minimizing the physics-informed loss function (6) with $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = 10$ and $\lambda_4 = 100$. It can be seen that the relative L^2 error measured at two subdomains decreases rapidly when #Sensors is less than 64; however, it tends to level off as #Sensors is further augmented due to other factors such as optimization errors and generalization errors. In addition, the errors in the two subdomains exhibit close proximity to one another, indicating the effectiveness of the proposed method in balancing errors across the subdomains. In order to ascertain the effect of #Samples on the performance of PI-IONet, DD-IONet and DD-DeepONet, we maintain #Sensors = 128. As illustrated in the right panel of Fig. 13, the relative L^2 errors tend to decrease with an increase in the number of samples. This observation aligns with the findings of Example 3, which concerns a parameterized interface problem with a single input source. It is noteworthy that the accuracy of PI-IONet is less sensitive to #Samples compared to DD-IONet and DD-DeepONet. For instance, when the number of training samples is limited to #Samples = 200, PI-IONet achieves the lowest relative L^2 error among the three models without any paired input-output measurements. Table 3 shows the relative L^2 error and the training cost of PI-IONet, DD-IONet and DD-DeepONet when #Sensors = 128 and #Samples = 1600. It is observed that with a sufficiently large

[1] One common approach is to use a random number generator to sample from a normal distribution with zero mean and unit variance and then apply a spectral representation to generate the desired spatial correlation structure, see [49] for more details.

[2] The implementation is based on the Fenics platform [59].

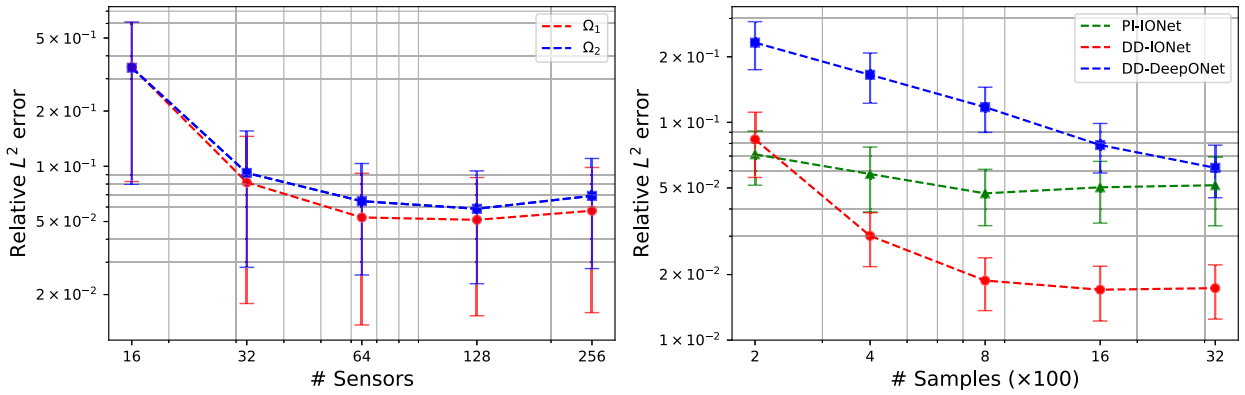


Fig. 13. Numerical results of Example 4. Left: the relative L^2 error of PI-IONet with respect to #Sensors, where #Samples = 1600. Right: The relative L^2 error of PI-IONet, DD-IONet and DD-DeepONet with different number of training samples, where #Sensors = 128.

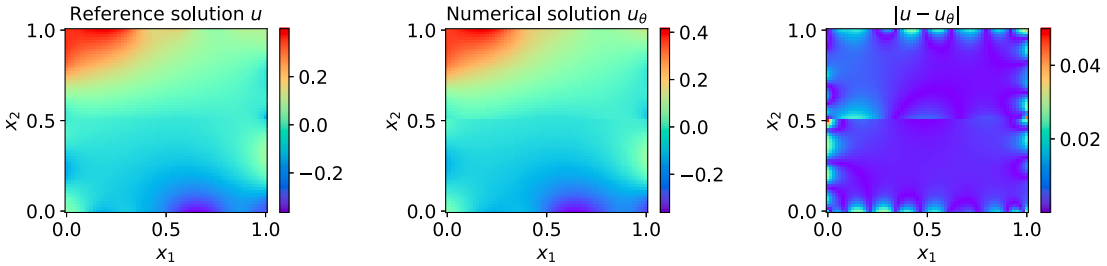


Fig. 14. The profile of the reference solution, the numerical solution obtained by PI-IONet, and the corresponding point-wise error in the whole domain for a representative example in the test dataset.

number of samples, the relative L^2 error of all three models can reach the order of 10^{-2}, and DD-IONet outperforms DD-DeepONet. An illustration of a numerical solution obtained using PI-IONet is presented in Fig. 14, with the relative L^2 error measured at 4.13×10^{-2}. The numerical solution derived from PI-IONet exhibits consistency with the reference solution.

4.3. Parametric elliptic interface problems in three dimensions

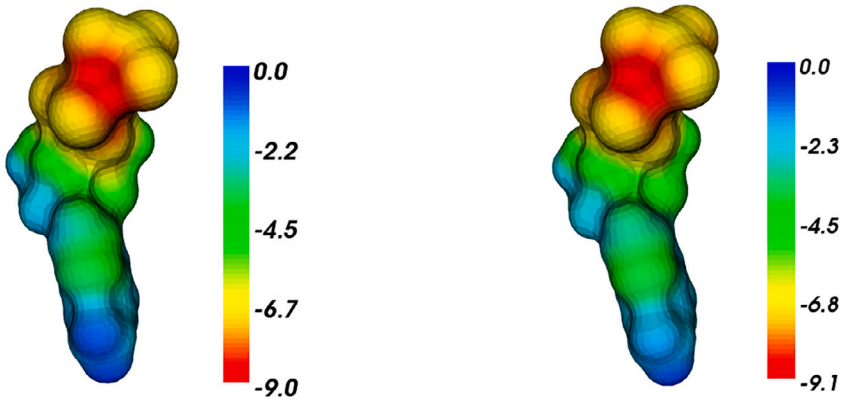
Example 5. To illustrate the capability of the proposed method for solving nonlinear interface problems, we consider the Poisson-Boltzmann equation (PBE), a prevalent implicit continuum model utilized in the estimation of biomolecular electrostatic potentials $\Phi(\mathbf{x})$. Similar equations occur in various applications, including electrochemistry and semiconductor physics. The molecule in the PBE is represented by a series of N_m charges q_i at positions $\mathbf{c}_i$, where $q_i = z_i e_c$, $z_i \in \mathbb{R}$, $i = 1, \cdots, N_m$. Specifically, we choose a real molecule (PDBID: ADP) with $N_m = 39$ atoms as an example. Without loss of generality, the molecule is translated from the average coordinate center of all atoms to the center of $\Omega = [-10, 10]^3$. Then, in the special case of $1:1$ electrolyte, the PBE can be formulated for dimensionless potential $u(\mathbf{x}) = e_c k_B^{-1} T^{-1} \Phi(\mathbf{x})$ as follows:

$$
\begin{aligned}
-\nabla \cdot (\epsilon(\mathbf{x}) \nabla u(\mathbf{x})) + \bar{\kappa}^2(\mathbf{x}) \sinh(u(\mathbf{x})) &= \alpha \sum_{i=1}^{N_m} z_i \delta(\mathbf{x} - \mathbf{c_i}), && \mathbf{x} \in \Omega, \\
[\![u(\mathbf{x})]\!] &= 0, && \mathbf{x} \in \Gamma, \\
[\![\epsilon(\mathbf{x}) \frac{\partial u(\mathbf{x})}{\partial \mathbf{n}}]\!] &= 0, && \mathbf{x} \in \Gamma, \\
u(\mathbf{x}) &= \frac{\alpha}{4\pi\epsilon(\mathbf{x})} \sum_{i=1}^{N_m} z_i \frac{e^{-\kappa \|\mathbf{x} - \mathbf{c}_i\|}}{\|\mathbf{x} - \mathbf{c}_i\|}, && \mathbf{x} \in \partial\Omega,
\end{aligned}
\tag{14}
$$

where $\delta(\cdot)$ is the Dirac delta function, the permittivity $\epsilon(\mathbf{x})$ takes the values of $\epsilon_m \epsilon_0$ and $\epsilon_s \epsilon_0$ in the molecular region Ω_1 and the solution region Ω_2, respectively. The modified Debye-Hückel takes the values $\bar{\kappa} = 0$ in Ω_1 and $\bar{\kappa} = \sqrt{\epsilon_m \epsilon_0} \kappa$ in Ω_2, and constant $\alpha = \frac{e_c^2}{k_B T}$. Here, constants $\epsilon_0, e_c, \beta, \kappa$ and T represent the vacuum dielectric constant, fundamental charge, Boltzmann's constant, Debye-Hückel constant and absolute temperature, respectively. Our goal is to learn an operator $\mathcal{G}$ mapping from the permittivity $\epsilon(\mathbf{x})$ to the solution $u(\mathbf{x})$ to PBE. Note that ϵ has a piece-wise constant nature, allowing us to directly utilize the function values as inputs for IONet without requiring sensor-based discretization.

Table 4
Test errors and training costs for PI-IONet, DD-IONet, and DD-DeepONet. The error corresponds to the relative L^2 error, recorded in the form of mean ± standard deviation based on all test input functions in Example 5.

Model	Activation	Depth	Width	#Parameters	$L^2(\mathcal{G}_\theta, \mathcal{G})$	Training time (hours)
PI-IONet	Tanh	5	150	364 K	1.24e-2±8.31e-5	0.51
DD-IONet	ReLU	5	150	364 K	4.33e-3±1.61e-3	0.17
DD-DeepONet	ReLU	5	215	373 K	1.54e-2±3.80e-4	0.10



(a) Surface potential obtained by FEM. (b) Surface potential obtained by PI-IONet.

Fig. 15. Surface potentials for protein ADP. Here, $\epsilon_m = 2$ and $\epsilon_s = 80$.

To numerically solve PBE (14), we use a solution decomposition scheme to overcome the singular difficulty caused by the Dirac delta distributions. Similar to our former work [33], u is decomposed as $u(\mathbf{x}) = \bar{G}(\mathbf{x}) + \bar{u}(\mathbf{x})$, where

$$\bar{G}(\mathbf{x}) = \frac{\alpha}{4\pi\epsilon_m\epsilon_0}\sum_{i=1}^{N_m}\frac{z_i}{\|\mathbf{x}-\mathbf{c}_i\|},\quad \nabla\bar{G}(\mathbf{x}) = -\frac{\alpha}{4\pi\epsilon_m\epsilon_0}\sum_{i=1}^{N_m} z_i\frac{\mathbf{x}-\mathbf{c}_i}{\|\mathbf{x}-\mathbf{c}_i\|^3}$$

are restricted to Ω_1, and $\bar{u}(\mathbf{x})$ satisfies the following PDE:

$$\begin{aligned}
-\nabla\cdot(\epsilon_m\epsilon_0\nabla\bar{u}(\mathbf{x})) &= 0, && \mathbf{x}\in\Omega_1,\\
-\nabla\cdot(\epsilon_s\epsilon_0\nabla\bar{u}(\mathbf{x})) + \bar{\kappa}^2\sinh(\bar{u}(\mathbf{x})) &= 0, && \mathbf{x}\in\Omega_2,\\
[\![\bar{u}(\mathbf{x})]\!] &= \bar{G}(\mathbf{x}), && \mathbf{x}\in\Gamma,\\
[\![\epsilon(\mathbf{x})\frac{\partial\bar{u}(\mathbf{x})}{\partial\mathbf{n}}]\!] &= \epsilon_m\epsilon_0\frac{\partial\bar{G}(\mathbf{x})}{\partial\mathbf{n}}, && \mathbf{x}\in\Gamma,\\
\bar{u}(\mathbf{x}) &= \frac{\alpha}{4\pi\epsilon_s\epsilon_0}\sum_{i=1}^{N_m} z_i\frac{e^{-\kappa\|\mathbf{x}-\mathbf{c}_i\|}}{\|\mathbf{x}-\mathbf{c}_i\|}, && \mathbf{x}\in\partial\Omega.
\end{aligned}$$

The training dataset comprises 1,000 input functions (constants) uniformly and randomly sampled from the space $(\epsilon_m, \epsilon_s) \in [1,2]\times[80,100]$, while the test dataset is composed of equidistant grid points arranged in a 6×6 grid within this space. For each test sample, we solved PBE (14) using piece-wise linear FEM [7] on an interface-fitted mesh to generate the reference solution. Specifically, the grid points of the FEM mesh consist of 1407, 3403, 2743 and 2402 points in Ω_1, Ω_2, Γ and $\partial\Omega$, respectively. During the training phase, for each input function, the training points used to evaluate the loss function consist of a collection of one-tenth randomly sampled grid points, rather than a set of randomly sampled scatter points within the domain. This approach ensures consistency in the interface across different methods. Note that the unit outward normal vector for each point on the interface Γ is approximated by taking the average of the outward normal directions of all elements that contain the corresponding point.

In this example, we approximate the solution operator of PBE (14) using PI-IONet, DD-IONet and DD-DeepONet. Herein, the weights in the loss function (6) of PI-IONet are set as $\lambda_1 = \lambda_2 = \lambda_3 = 1$ and $\lambda_4 = 100$. Table 4 records test errors measured at 4810 grid points in Ω_1 and Ω_2 and training costs of these three models after 5×10^4 parameter updates. It can be observed that DD-IONet achieves the lowest relative L^2 error. Additionally, despite lacking any paired input-output measurements, except for the boundary conditions, the error accuracy of PI-IONet is comparable to that of DD-DeepONet, albeit with slightly higher training costs. A visual comparison of the reference and predicted surface potentials of the protein ADP is shown in Fig. 15. These findings further emphasize the capability of IONet handle parametric interface problems within irregular interface. While our current work demonstrates the effectiveness of IONet in solving PBE with a real small-molecule ADP, further research is needed to investigate the computational

efficiency of PI-IONet and other neural network-based methods in solving large-scale computational problems in biophysics, such as solving PBE with real macromolecules. We will postpone this part of the work to future research.

In this example, we approximate the solution operator of PBE (14) using PI-IONet, DD-IONet and DD-DeepONet. Herein, the weights in the loss function (6) of PI-IONet are set as $\lambda_1 = \lambda_2 = \lambda_3 = 1$ and $\lambda_4 = 100$, and the network hyperparameters are presented in Table 4. In all cases, the neural networks are trained after 5×10^4 parameter updates. The test errors and training costs of these three models are shown in Table 4. It can be observed that DD-IONet achieves the lowest relative L^2 error. Additionally, despite lacking any paired input-output measurements, except for the boundary conditions, the error accuracy of PI-IONet is comparable to that of DD-DeepONet, albeit with slightly higher training costs. In addition, a visual comparison of the reference and predicted surface potentials of the protein ADP is shown in Fig. 15. These findings further emphasize the capability of IONet to effectively handle parametric interface problems within irregular interface. While our current work demonstrates the effectiveness of IONet in solving PBE with a real small-molecule ADP, further research is needed to investigate the computational efficiency of PI-IONet and other neural network-based methods in solving large-scale computational problems in biophysics, such as solving PBE with real macromolecules. We will postpone this part of the work to future research.

4.4. Parametric elliptic interface problems in six dimensions

Example 6. Our final example aims to highlight the ability of the proposed framework to handle high-dimensional parametric interface problems. Here, we consider Eq. (1) defined on a 6-dimension sphere of radius 0.6 domain Ω enclosing another smaller 6-dimension sphere of radius 0.5 as the interior domain Ω_1. Our goal is to learn the solution operator mapping from the source term f to the latent solution of Eq. (1), i.e., $\mathcal{G} : f(\mathbf{x}) \to u(\mathbf{x})$. In this case, f has the following specific forms

$$f(\mathbf{x}) = \begin{cases} -p_1 \prod_{i=1}^{6} \exp(x_i), & \mathbf{x} \in \Omega_1, \\ -p_2 \prod_{i=1}^{6} \sin(x_i), & \mathbf{x} \in \Omega_2, \end{cases}$$

where (p_1, p_2) randomly sample from $[1, 10] \times [-10^{-2}, -10^{-3}]$. For the problem setup, the coefficient

$$a(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in \Omega_1, \\ 10^{-3}, & \mathbf{x} \in \Omega_2, \end{cases}$$

has a large contrast ($a_1/a_2 = 10^3$), the boundary condition is given as $h(\mathbf{x}) = \prod_{i=1}^{6} \sin(x_i)$, the interface conditions are chosen as $g_D(\mathbf{x}) = \prod_{i=1}^{6} \sin(x_i) - \prod_{i=1}^{6} \exp(x_i)$, and

$$g_N(\mathbf{x}) = \frac{5}{3} \left(10^{-3} \sum_{i=1}^{6} \left(x_i \cos(x_i) \prod_{j=1, j\neq i}^{6} \sin(x_j) \right) - \left(\sum_{i=1}^{6} x_i \right) \prod_{j=1}^{6} \exp(x_j) \right).$$

Note that, when we take f with $p_1 = 6$ and $p_2 = -6 \times 10^{-3}$, Eq. (1) exists an exact solution [40]

$$u(\mathbf{x}) = \begin{cases} \prod_{i=1}^{6} \exp(x_i), & \mathbf{x} \in \Omega_1, \\ \prod_{i=1}^{6} \sin(x_i), & \mathbf{x} \in \Omega_2. \end{cases}$$

In this study, we train PI-IONet $\mathcal{G}_\theta$ with different scales of architecture to approximate the solution operator and then test the trained model in the case of input f with $p_1 = 6$ and $p_2 = -6 \times 10^{-3}$. In all cases, we train PI-IONet by minimizing the loss function (6) with $\lambda_1 = \lambda_2 = \lambda_3 = 1$ and $\lambda_4 = 100$ for 4×10^4 iterations of parameter updates, utilizing 100 randomly sampled input functions. The sensors for discretizing the input functions comprise a set of randomly sampled points in the computational domain, as opposed to lattice points in a higher dimensional space. Specifically, we set the number of sensors as 40.

Table 5 illustrates the test errors measured at $10,000$ test data points over the whole domain and the computational costs for training PI-IONet. It is shown that the accuracy of the numerical solution improves while the training cost grows as the number of trainable parameters increases. The final L^∞ error and relative L^2 error are about 0.7% and 0.2%, respectively, with low deviations. These findings suggest that PI-IONet has the potential to achieve high performance in dealing with high-dimensional output solutions of parametric elliptic interface problems, even in scenarios involving high-contrast coefficients.

5. Conclusions

In this work, we have investigated deep neural network-based operator learning methods and proposed the interfaced operator network (IONet) to tackle parametric elliptic interface problems. The main contribution is that we first combine the domain-

Table 5
L^∞ and relative L^2 errors for PI-IONet with different widths and depths. These errors are obtained by averaging the results from three independent experiments, each involving different network random initialization and randomly generated training dataset.

Depth	Width	#Parameters	$\|u-u_\theta\|_\infty$	$\frac{\|u-u_\theta\|_2}{\|u\|_2}$	Training time (hours)
3	30	9k	3.35e-2±1.95e-2	1.38e-2±6.18e-4	0.42
4	40	21k	1.36e-2±6.06e-3	8.04e-3±4.01e-3	0.55
5	50	43k	7.67e-3±1.46e-3	1.56e-3±6.84e-4	0.66

decomposed method with the operator learning methods and employ multiple branch nets and trunk nets to explicitly handle the discontinuities across the interface in the input and output functions. In addition, we introduce tailored physics-informed loss designed to constrain the physical consistency of the proposed model. This strategy reduces the requirement for training data and empowers the IONet to remain effective even in the absence of paired input-output training data. We also provide theory and numerical experiments to demonstrate that the proposed IONet is effective and reliable for approximating the solution operator of parametric interface problems. In our simulations, we systematically studied the effects of different factors on the accuracy of IONet and existing state-of-the-art operator networks. The results show that the proposed IONet is more robust and accurate in dealing with many kinds of parametric interface problems due to its discontinuity-preserving architecture.

Despite the preliminary success, there are still many issues that need further investigation. As an advantage of neural operators is their fast predictions, an important aspect of interest regarding IONet is the systematic comparison of its computational cost with other numerical methods for solving interface problems. In addition, one limitation is the absence of treating geometry configuration as an input function in our current work. Integrating the geometry configuration into IONet could potentially enhance its capabilities and further broaden its range of applications. Furthermore, we have not yet obtained the convergence rate for IONet, which would provide valuable insights into both the accuracy and stability of the model. Inspired by the works on the error estimates for DeepONets [60] and generalization performance analysis of deep learning for PDEs [61–64], including interface problems [65], it is interesting to improve the convergence properties and the error estimation of IONet for solving parametric elliptic interface problems. Moreover, IONet can be viewed as a specific DeepONet preserving discontinuity, and we are also interested in exploring the feasibility of integrating recent advancements and extensions from DeepONet (e.g., DeepONet with proper orthogonal decomposition [51], DeepONet based on latent representations and autoencoders [66], DeepONet using Laplace transform [67]) into IONet.

CRediT authorship contribution statement

Sidi Wu: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Aiqing Zhu:** Writing – review & editing, Validation, Methodology, Formal analysis, Conceptualization. **Yifa Tang:** Writing – review & editing, Supervision, Resources, Funding acquisition. **Benzhuo Lu:** Writing – review & editing, Validation, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link to my data.

Acknowledgements

This research was funded by the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDB0500000) and the National Natural Science Foundation of China (Grant Nos. 12371413, 22073110 and 12171466).

Appendix A. Proof of Theorem 1

Here we provide the proof of Theorem 1, which relies on the universal approximation theorem of FNNs (see, e.g., [68]) and the tensor product decomposition of operators [50].

Proof. Denote $T_i = \{a|_{\overline{\Omega}_i} \mid a \in T\} \subset C(\overline{\Omega}_i)$ where $i = 1, \cdots, I$. Consider an operator $\tilde{\mathcal{G}}$ mapping from $T_1 \times \cdots \times T_I$ to $X(\Omega)$ as

$$\tilde{\mathcal{G}}(a_1, \cdots, a_I) := \mathcal{G}(a), \quad \text{where } a(x) = a_i(x), \quad \text{if } x \in \Omega_i. \tag{A.1}$$

Observing the fact that the function values evaluated at given points are equivalent to the piece-wise linear functions of Faber-Schauder basis, and by corollary 2.6 in [50], for any $\varepsilon > 0$, there exist positive integers m_i, K and continuous functions $g_{i,k} \in C(\mathbb{R}^{m_i})$ and $u_k \in X(\Omega)$ and $\mathbf{y}_i^1, \cdots, \mathbf{y}_{m_i}^i \in \Omega_i$, where $k = 1, \cdots, K$, $i = 1, \cdots, I$, such that

$$\sup_{a_1 \in T_1,\ a_2 \in T_2} \left\| \tilde{\mathcal{G}}(a_1, \cdots, a_I)(\cdot) - \sum_{k=1}^{K} \prod_{i=1}^{I} g_{i,k}(a_i(\mathbf{y}_i^1), \cdots, a_i(\mathbf{y}_{m_i}^i)) \cdot u_k(\cdot) \right\|_{X(\Omega)} \leq \frac{\varepsilon}{2}. \tag{A.2}$$

Since T_i is a collection of continuous functions, and Ω is bounded, we have that $A_i := \{(a_i(\mathbf{y}_i^1), \cdots, a_i(\mathbf{y}_i^{m_i})) | a_i \in T_i\} \in \mathbb{R}^{m_i}$ is bounded for $i = 1, \cdots, I$, and there exists a cuboid containing A_i. Consequently, by the approximation theorems of tanh FNN [68], for any $\delta > 0$, there exist tanh FNNs $\mathcal{N}_{b_i} : \mathbb{R}^{m_i} \to \mathbb{R}^K$, $\mathcal{N}_t^i : \Omega_i \to \mathbb{R}^K$, such that

$$\left\| [\mathcal{N}_{b_i}]_k - g_{i,k} \right\|_{C^\infty(A_i)} \leq \delta, \quad \left\| [\mathcal{N}_t^i]_k - u_k|_{\Omega_i} \right\|_{H^2(\Omega_i)} \leq \delta, \quad i = 1, \cdots, I, \quad k = 1, \cdots, K,$$

where $[\mathcal{N}]_k$ denotes the k-th component of $\mathcal{N}$. Denote

$$M_k = \max\{ \|g_{1,k}\|_{C^\infty(A_1)}, \cdots, \|g_{I,k}\|_{C^\infty(A_I)}, \|u_k\|_{X(\Omega)} \},\ k = 1, \cdots, K.$$

Subsequently, we can choose a sufficiently small δ such that

$$\begin{aligned} &\left\| \sum_{k=1}^{K} \prod_{i=1}^{I} [\mathcal{N}_{b_i}]_k(a_i(\mathbf{y}_i^1), \cdots, a_i(\mathbf{y}_{m_i}^i)) \cdot [\mathcal{N}_t^i]_k(\cdot) - \sum_{k=1}^{K} \prod_{i=1}^{I} g_{i,k}(a_i(\mathbf{y}_i^1), \cdots, a_i(\mathbf{y}_{m_i}^i)) \cdot u_k(\cdot) \right\|_{H^2(\Omega_i)} \\ &\leq \sum_{k=1}^{K} (M_k + \delta)^{I+1} - M_k^{I+1} \leq \frac{\varepsilon}{2}. \end{aligned} \tag{A.3}$$

Finally, combining definition (A.1), estimates (A.2) and (A.3), we conclude that for $i = 1, \cdots, I$,

$$\begin{aligned} &\sup_{a \in T} \left\| \mathcal{G}(a)(\cdot) - \mathcal{S}\left(\mathcal{N}_{b_1}(a(\mathbf{y}_1^1), \cdots, a(\mathbf{y}_{m_1}^1)) \odot \mathcal{N}_{b_2}(a(\mathbf{y}_1^2), \cdots, a(\mathbf{y}_{m_2}^2)) \odot \mathcal{N}_t^i(\cdot) \right) \right\|_{H^2(\Omega_i)} \\ &\leq \sup_{a \in T} \left\| \tilde{\mathcal{G}}(a|_{\Omega_1}, \cdots, a|_{\Omega_I})(\cdot) - \sum_{k=1}^{K} \prod_{i=1}^{I} g_{i,k}(a_i(\mathbf{y}_i^1), \cdots, a_i(\mathbf{y}_{m_i}^i)) \cdot u_k(\cdot) \right\|_{H^2(\Omega_i)} \\ &\quad + \sup_{a \in T} \left\| \sum_{k=1}^{K} \prod_{i=1}^{I} [\mathcal{N}_{b_i}]_k(a_i(\mathbf{y}_i^1), \cdots, a_i(\mathbf{y}_{m_i}^i)) \cdot [\mathcal{N}_t^i]_k(\cdot) - \sum_{k=1}^{K} \prod_{i=1}^{I} g_{i,k}(a_i(\mathbf{y}_i^1), \cdots, a_i(\mathbf{y}_{m_i}^i)) \cdot u_k(\cdot) \right\|_{H^2(\Omega_i)} \\ &\leq \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon, \end{aligned}$$

which completes the proof. □

References

[1] M. Sussman, E. Fatemi, An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow, SIAM J. Sci. Comput. 20 (4) (1999) 1165–1191.

[2] E.A. Fadlun, R. Verzicco, P. Orlandi, J. Mohd-Yusof, Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations, J. Comput. Phys. 161 (1) (2000) 35–60.

[3] Y. Liu, M. Sussman, Y. Lian, M.Y. Hussaini, A moment-of-fluid method for diffusion equations on irregular domains in multi-material systems, J. Comput. Phys. 402 (2020) 109017.

[4] L. Wang, H. Zheng, X. Lu, L. Shi, A Petrov-Galerkin finite element interface method for interface problems with Bloch-periodic boundary conditions and its application in phononic crystals, J. Comput. Phys. 393 (2019) 117–138.

[5] J.S. Hesthaven, High-order accurate methods in time-domain computational electromagnetics: a review, Adv. Imaging Electron Phys. 127 (2003) 59–123.

[6] B. Lu, Y. Zhou, M. Holst, J. McCammon, Recent progress in numerical methods for the Poisson-Boltzmann equation in biophysical applications, Commun. Comput. Phys. 3 (5) (2008) 973–1009.

[7] N. Ji, T. Liu, J. Xu, L.Q. Shen, B. Lu, A finite element solution of lateral periodic Poisson–Boltzmann model for membrane channel proteins, Int. J. Mol. Sci. 19 (3) (2018) 695.

[8] J. Philip, Flow in porous media, Annu. Rev. Fluid Mech. 2 (1) (1970) 177–204.

[9] Y. Khoo, J. Lu, L. Ying, Solving parametric pde problems with artificial neural networks, Eur. J. Appl. Math. 32 (3) (2021) 421–435.

[10] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.

[11] I. Babuška, The finite element method for elliptic equations with discontinuous coefficients, Computing 5 (3) (1970) 207–213.

[12] J.H. Bramble, J.T. King, A finite element method for interface problems in domains with smooth boundaries and interfaces, Adv. Comput. Math. 6 (1996) 109–138.

[13] Y. Chen, S. Hou, X. Zhang, A bilinear partially penalized immersed finite element method for elliptic interface problems with multi-domain and triple-junction points, Results Appl. Math. 8 (2020) 100100.

[14] B. Zhang, J. DeBuhr, D. Niedzielski, S. Mayolo, B. Lu, T. Sterling, DASHMM accelerated adaptive fast multipole Poisson-Boltzmann solver on distributed memory architecture, Commun. Comput. Phys. 25 (4) (2019) 1235–1258.

[15] L. Mu, J. Wang, X. Ye, S. Zhao, A new weak Galerkin finite element method for elliptic interface problems, J. Comput. Phys. 325 (2016) 157–173.

[16] T. Liu, M. Chen, B. Lu, Efficient and qualified mesh generation for Gaussian molecular surface using adaptive partition and piecewise polynomial approximation, SIAM J. Sci. Comput. 40 (2) (2018) B507–B527.
[17] C.S. Peskin, The immersed boundary method, Acta Numer. 11 (2002) 479–517.
[18] R.J. LeVeque, Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, SIAM J. Numer. Anal. 31 (4) (1994) 1019–1044.
[19] Z. Chen, Y. Xiao, L. Zhang, The adaptive immersed interface finite element method for elliptic and Maxwell interface problems, J. Comput. Phys. 228 (14) (2009) 5000–5019.
[20] R.P. Fedkiw, T. Aslam, B. Merriman, S. Osher, A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method), J. Comput. Phys. 152 (2) (1999) 457–492.
[21] R. Egan, F. Gibou, xGFM: recovering convergence of fluxes in the ghost fluid method, J. Comput. Phys. 409 (2020) 109351.
[22] D. Bochkov, F. Gibou, Solving elliptic interface problems with jump conditions on Cartesian grids, J. Comput. Phys. 407 (2020) 109269.
[23] W. Thacher, H. Johansen, D. Martin, A high order Cartesian grid, finite volume method for elliptic interface problems, J. Comput. Phys. 491 (2023) 112351.
[24] K. Xia, M. Zhan, G.-W. Wei, Mib method for elliptic equations with multi-material interfaces, J. Comput. Phys. 230 (12) (2011) 4588–4615.
[25] I. Babuška, U. Banerjee, Stable generalized finite element method (sgfem), Comput. Methods Appl. Mech. Eng. 201 (2012) 91–111.
[26] H. Liu, L. Zhang, X. Zhang, W. Zheng, Interface-penalty finite element methods for interface problems in h1, h (curl), and h (div), Comput. Methods Appl. Mech. Eng. 367 (2020) 113137.
[27] A. Taleei, M. Dehghan, Direct meshless local Petrov–Galerkin method for elliptic interface problems with applications in electrostatic and elastostatic, Comput. Methods Appl. Mech. Eng. 278 (2014) 479–498.
[28] F. Gholampour, E. Hesameddini, A. Taleei, A global rbf-qr collocation technique for solving two-dimensional elliptic problems involving arbitrary interface, Eng. Comput. 37 (4) (2021) 3793–3811.
[29] M. Ahmad, S. ul Islam, E. Larsson, Local meshless methods for second order elliptic interface problems with sharp corners, J. Comput. Phys. 416 (2020) 109500.
[30] Ö. Oruç, An efficient meshfree method based on Pascal polynomials and multiple-scale approach for numerical solution of 2-d and 3-d second order elliptic interface problems, J. Comput. Phys. 428 (2021) 110070.
[31] Z. Wang, Z. Zhang, A mesh-free method for interface problems using the deep learning approach, J. Comput. Phys. 400 (2020) 108963.
[32] C. He, X. Hu, L. Mu, A mesh-free method using piecewise deep neural network for elliptic interface problems, J. Comput. Appl. Math. 412 (2022) 114358.
[33] S. Wu, B. Lu, INN: interfaced neural networks as an accessible meshless approach for solving interface PDE problems, J. Comput. Phys. 470 (2022) 111588.
[34] H. Guo, X. Yang, Deep unfitted Nitsche method for elliptic interface problems, Commun. Comput. Phys. 31 (4) (2022) 1162–1179.
[35] Q. Sun, X. Xu, H. Yi, Dirichlet-Neumann learning algorithm for solving elliptic interface problems, 2023.
[36] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.
[37] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (xpinns): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. 28 (5) (2020) 2002–2041.
[38] Z. Liu, W. Cai, Z.-Q. John Xu, Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains, Commun. Comput. Phys. 28 (5) (2020) 1970–2001.
[39] J. Ying, J. Liu, J. Chen, S. Cao, M. Hou, Y. Chen, Multi-scale fusion network: a new deep learning structure for elliptic interface problems, Appl. Math. Model. 114 (2023) 252–269.
[40] W.-F. Hu, T.-S. Lin, M.-C. Lai, A discontinuity capturing shallow neural network for elliptic interface problems, J. Comput. Phys. 469 (2022) 111576.
[41] M.-C. Lai, C.-C. Chang, W.-S. Lin, W.-F. Hu, T.-S. Lin, A shallow Ritz method for elliptic problems with singular sources, J. Comput. Phys. 469 (2022) 111547.
[42] Y.-H. Tseng, T.-S. Lin, W.-F. Hu, M.-C. Lai, A cusp-capturing pinn for elliptic interface problems, J. Comput. Phys. (2023) 112359.
[43] D.J. Lucia, P.S. Beran, W.A. Silva, Reduced-order modeling: new approaches for computational physics, Prog. Aerosp. Sci. 40 (1–2) (2004) 51–117.
[44] A. Quarteroni, A. Manzoni, F. Negri, Reduced Basis Methods for Partial Differential Equations: An Introduction, vol. 92, Springer, 2015.
[45] A.J. Majda, D. Qi, Strategies for reduced-order models for predicting the statistical responses and uncertainty quantification in complex turbulent dynamical systems, SIAM Rev. 60 (3) (2018) 491–549.
[46] Z. Ye, X. Huang, H. Liu, B. Dong, Meta-auto-decoder: a meta-learning based reduced order model for solving parametric partial differential equations, preprint, arXiv:2302.08263, 2023.
[47] Z. Long, Y. Lu, B. Dong, Pde-net 2.0: learning pdes from data with a numeric-symbolic hybrid deep network, J. Comput. Phys. 399 (2019) 108925.
[48] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218–229.
[49] Z. Li, N.B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A.M. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, in: International Conference on Learning Representations, 2021.
[50] P. Jin, S. Meng, L. Lu, Mionet: learning multiple-input operators via tensor product, SIAM J. Sci. Comput. 44 (6) (2022) A3490–A3514.
[51] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, G.E. Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data, Comput. Methods Appl. Mech. Eng. 393 (2022) 114778.
[52] S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed deeponets, Sci. Adv. 7 (40) (2021), eabi8605.
[53] W. Littman, G. Stampacchia, H.F. Weinberger, Regular points for elliptic equations with discontinuous coefficients, Ann. Sc. Norm. Super. Pisa, Cl. Sci. 17 (1–2) (1963) 43–77.
[54] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Netw. 6 (4) (1995) 911–917.
[55] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018) 1–43.
[56] Z. Chen, J. Zou, Finite element methods and their convergence for elliptic and parabolic interface problems, Numer. Math. 79 (2) (1998) 175–202.
[57] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[58] M. Zhu, H. Zhang, A. Jiao, G.E. Karniadakis, L. Lu, Reliable extrapolation of deep neural operators informed by physics or sparse observations, Comput. Methods Appl. Mech. Eng. 412 (2023) 116064.
[59] A. Logg, G.N. Wells, DOLFIN: automated finite element computing, ACM Trans. Math. Softw. 37 (2) (2010) 20:1–20:28.
[60] S. Lanthaler, S. Mishra, G.E. Karniadakis, Error estimates for deeponets: a deep learning framework in infinite dimensions, Transactions of Mathematics and Its Applications 6 (1) (2022) tnac001.
[61] Y. Lu, H. Chen, J. Lu, L. Ying, J.H. Blanchet, Machine learning for elliptic pdes: fast rate generalization bound, neural scaling law and minimax optimality, in: The Tenth International Conference on Learning Representations, ICLR, 2022.
[62] Y. Shin, J. Darbon, G. Em Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, Commun. Comput. Phys. 28 (5) (2020) 2042–2074.
[63] T. Luo, H. Yang, Two-layer neural networks for partial differential equations: optimization and generalization theory, preprint, arXiv:2006.15733, 2020.
[64] Y. Jiao, Y. Lai, D. Li, X. Lu, Y. Wang, J.Z. Yang, Convergence analysis for the PINNs, preprint, arXiv:2109.01780, 2021.

[65] S. Wu, A. Zhu, Y. Tang, B. Lu, Convergence of physics-informed neural networks applied to linear second-order elliptic interface problems, Commun. Comput. Phys. 33 (2) (2023) 596–627.
[66] K. Kontolati, S. Goswami, G.E. Karniadakis, M.D. Shields, Learning in latent spaces improves the predictive accuracy of deep neural operators, preprint, arXiv: 2304.07599, 2023.
[67] Q. Cao, S. Goswami, G.E. Karniadakis, Lno: Laplace neural operator for solving differential equations, preprint, arXiv:2303.10528, 2023.
[68] T. De Ryck, S. Lanthaler, S. Mishra, On the approximation of functions by tanh neural networks, Neural Netw. 143 (2021) 732–750.