Computer Methods in Applied Mechanics and Engineering 428 (2024) 117084

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

D2NO: Efficient handling of heterogeneous input function spaces with distributed deep neural operators

Zecheng Zhang [b], Christian Moya [e], Lu Lu [a], Guang Lin [d,e,*], Hayden Schaeffer [c]

[a] Department of Statistics and Data Science, Yale University, New Haven, 06511, CT, USA
[b] Department of Mathematics, Florida State University, Tallahassee, 32304, FL, USA
[c] Department of Mathematics, UCLA, Los Angeles, 90095, CA, USA
[d] Department of Mechanical Engineering, Purdue University, West Lafayette, 47907, IN, USA
[e] Department of Mathematics, Purdue University, West Lafayette, 47907, IN, USA

ARTICLE INFO

Keywords:
Operator learning
Distributed training
Heterogeneous inputs
Multiscale problems

ABSTRACT

Neural operators have been applied in various scientific fields, such as solving parametric partial differential equations, dynamical systems with control, and inverse problems. However, challenges arise when dealing with input functions that exhibit heterogeneous properties, requiring multiple sensors to handle functions with minimal regularity. To address this issue, discretization-invariant neural operators have been used, allowing the sampling of diverse input functions with different sensor locations. However, existing frameworks still require an equal number of sensors for all functions. We propose a novel distributed approach to further relax the discretization requirements and solve the heterogeneous dataset challenges. Our method involves partitioning the input function space and processing individual input functions using independent and separate neural networks. A centralized neural network is used to handle shared information across all output functions. This distributed methodology reduces the number of gradient descent back-propagation steps, improving efficiency while maintaining accuracy. We demonstrate that the corresponding neural network is a universal approximator of continuous nonlinear operators and present three numerical examples to validate its performance.

1. Introduction

Operator Learning, introduced by [1], involves learning operators that map one function to another. It extends classical function learning, which focuses on the mapping between vector spaces, and functional learning [2], which deals with functions mapping to fields. Operator Learning has become a pivotal tool in scientific machine learning [3]. Recent works on this topic include various neural operators [4–8] and extensive studies on the approximation and convergence aspects of these neural operators [1,9–11]. Additionally, applications of neural operators span various scientific and engineering domains. For example, researchers have developed algorithms that use neural operators to address parametric Partial Differential Equation (PDE) problems [4,12–16], dynamical systems with control [1,17,18], power engineering applications [19–21], design optimization problems [22,23], and challenges related to multifidelity and multiscale downscaling [22,24,25]. Notably, researchers have recently applied neural operators to climate predictions [26] and uncertainty quantification [27–29].

* Corresponding author at: Department of Mechanical Engineering, Purdue University, West Lafayette, 47907, IN, USA.
E-mail addresses: zzhang14@fsu.edu (Z. Zhang), cmoyacal@purdue.edu (C. Moya), lu.lu@yale.edu (L. Lu), guanglin@purdue.edu (G. Lin), hayden@math.ucla.edu (H. Schaeffer).

https://doi.org/10.1016/j.cma.2024.117084
Received 4 February 2024; Received in revised form 29 April 2024; Accepted 21 May 2024
Available online 8 June 2024

One of the popular approaches in this direction is the Deep Operator Neural Network (DeepONet) [4,7,12,22,30–32]. DeepONet utilizes two subnetwork architectures consisting of a branch network and a trunk network to approximate operators by constructing a basis for the output function space. The trunk networks are responsible for learning the basis, while the branch network acquires the basis coefficients. This results in an approximation achieved through a linear combination of the learned basis and coefficients. Another approach for neural operators is the Fourier neural operator (FNO) [5,26,33]. FNO applies the Fourier neural transformation to convolve the input function with a learned translation invariant kernel function. It then utilizes skip connections and other mechanisms to nonlinearly generalize the approximation. A comprehensive comparison of DeepONet and FNO can be found in [30]. Additionally, there are several other noteworthy neural operators documented, as cited here [6,8,10,34–36].

A significant advantage of DeepONet [1,4,7] and related structures [8,10] is the ability to discretize output functions freely [8, 30,37]. The neural operator generates function values at specific domain points, which are determined by the trunk network input, resulting in enhanced adaptability. This flexibility allows the network to predict output function values at any domain point, accommodating different mesh definitions for output functions. However, a major challenge is achieving discretization invariance [8,30]. Specifically, the neural operator must remain invariant to the different discretizations applied to input functions. Several algorithmic extensions to DeepONet have been proposed, allowing different input functions to be discretized differently [30,38]. Additionally, the Basis Enhanced Learning (BelNet) extension of DeepONet, proposed by [8,10], has mathematically demonstrated discretization invariance. It is worth noting that these extensions still require the number of sensors used to sample input function values to be consistent across distinct input functions. In this paper, we introduce a novel and efficient distributed network architecture and training approach for neural operators. This approach permits individualized discretization of input functions using dedicated sensor distribution strategies.

Discretization invariance offers several advantages, particularly when working with datasets that have diverse properties [33]. For example, a dataset may consist of input functions with different levels of regularity. When using a uniform set of sensors to sample and discretize these input functions, it becomes necessary to accommodate the functions with the lowest regularity. This leads to a significant increase in the number of required sensors. However, it also results in higher costs for functions that are inherently smoother. To accommodate input functions with varying properties and regularity, we will draw inspiration from federated learning. Introduced in [39], federated learning is a machine learning framework that enables multiple clients to collaboratively train a consensus neural network model in a distributed manner, while keeping their training data local. This approach protects data privacy and reduces communication costs by coordinating local model updates with a central server, resulting in a significant reduction in data transfer volume.

A standard formulation of federated learning is a distributed optimization framework that addresses communication costs, client robustness, and data heterogeneity across different clients [40]. Communication efficiency is central to this formulation and motivates the most well-known efficient communication algorithm in federated learning: the federated averaging (FedAvg) algorithm [39]. FedAvg has been studied under realistic scenarios in [41,42]. Furthermore, several works have provided convergence proofs of the algorithm within the field of optimization [43–47].

Traditionally, neural operators have been trained using a centralized strategy that involves transferring the training data to a central location. However, this approach hinders our ability to leverage high-performance distributed/parallel computing platforms. To address these limitations, the authors in [48] proposed Fed-DeepONet. Fed-DeepONet enables distributed training of deep operator networks and has been empirically shown to achieve accuracy comparable to the centralized vanilla DeepONet while handling heterogeneous input functions. However, it should be noted that Fed-DeepONet requires the number of sensors for each local DeepONet to be the same, as it trains a consensus global DeepONet. In this paper, we provide a theoretical demonstration of Fed-DeepONet's ability to approximate nonlinear operators. Additionally, we design a dedicated distributed training strategy that does not require the same number of sensors, allowing the approach to handle heterogeneous input function spaces.

We summarize the contributions of this paper as follows:

1. We propose the distributed deep neural operator (D2NO) method that allows for sampling different input functions using distinct sets of sensors. This training method can reduce the total number of back-propagation steps and the number of trainable parameters by utilizing dedicated neural networks to handle the diverse input function spaces.
2. D2NO is motivated by the mathematical theory that underlies the algorithm. Since all functions share the same output basis, we construct dedicated networks to handle the diverse input functions. We then proceed to demonstrate the universal approximation of the network and algorithms we propose.
3. The proposed methodology can be implemented using different neural operators. In this study, we apply it to the novel Deep Operator Neural Network (DeepONet). By utilizing the Distributed-DeepONet (D-DeepONet), we tackle complex heterogeneous datasets that contain input functions with different regularities and geometries. D2NO enhances efficiency while preserving predictive accuracy.

The rest of the paper is organized as follows. In Section 2, we will review the DeepONet and the universal approximation theory. Next, we present the methodology in Section 3. Then, we prove the universal approximation theorem for Distributed-DeepONet in Section 4. Finally, we present several numerical experiments containing heterogeneous input function spaces in Section 5.

2. Overview

2.1. Deep Operator Neural Network (DeepONet)

We start by defining an operator $G : V \to U$, where V and U are two function spaces. Next, we examine the Deep Operator Neural Network (DeepONet) [4,7,12,22,30], as demonstrated in Figure 1. The DeepONet structure contains a branch and trunk

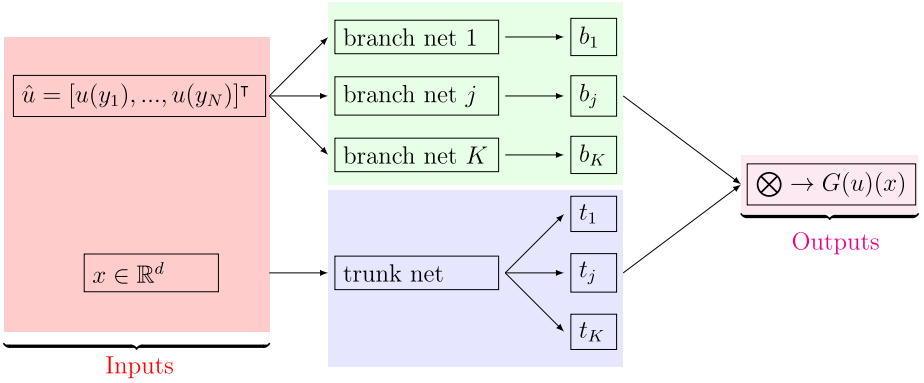

Fig. 1. Stacked version DeepONet. $\otimes$ denotes the inner product in $\mathbb{R}^K$. Specifically, $t_j \in \mathbb{R}$ is the trunk net output, $b_j \in \mathbb{R}$ is the brunch net output, and u is a function in the input function space .

network. The branch net processes a discretized input function, while the trunk network processes an arbitrary location within a given output function space. The DeepONet structure avoids needing to discretize the output function. This allows more flexibility in both the training and prediction processes, distinguishing it from neural operators that rely on output function discretization on fixed grid mesh (see Fig. 1).

The DeepONet is theoretically based on the universal approximation theorem of nonlinear operators, which was established in [1,2,4], and extended in [4,10]. The main results are stated below.

Theorem 2.1 (*Theorem 5, [1]*). *Suppose $G : V \to U$ is continuous. Specifically, let V be compact in the continuous function space $C(K_1)$, where K_1 is compact in a Banach space. Additionally, suppose $G(V) \subset C(K_2)$, where $K_2 \subset \mathbb{R}^d$ is also compact. Then, for any ϵ, there exist sensors $\{y_i\}_{i=1}^N \subset K_1$, networks $p_k : \mathbb{R}^N \to \mathbb{R}$, and $b_k : K_2 \to \mathbb{R}$ such that,*

$$|G(u)(x) - \sum_{k=1}^{K} p_k(\hat{u}) b_k(x)| < \epsilon,$$

for all $x \in K_2$ and $u \in V$. Here $\hat{u} = [u(y_1), \dots, u(y_N)]^\intercal$.

2.2. Discretization-invariant extension of DeepONet

Theorem 2.1 presents an existence argument stating that the sensors are the same across all input functions. This implies that all distinct input functions must be discretized on the same mesh. However, this limitation hinders the use of DeepONet, especially in cases where the dataset exhibits heterogeneous properties (for a demonstration, refer to the numerical experiments Section 5). A neural operator is considered discretization-invariant if it remains unaffected by variations in input function discretizations. In [8,10], an extension of DeepONet was shown to be discretization-invariant, and specifically, the authors proved the following theorem:

Theorem 2.2 (*Discretization-Invariance [8,10]*). *Suppose that $a \in TW$, Y is a Banach space, $K_1 \subset Y$, and $K_2 \subset \mathbb{R}$ are all compact. Let $V \subset C(K_1)$ be a compact set and $G : V \to C(K_2)$ be a continuous and nonlinear operator. For any $\epsilon > 0$, there exist integers N, C, K, I, weights and biases $W_x^k \in \mathbb{R}^d$, $b_x^k \in \mathbb{R}$, $W^k \in \mathbb{R}^{I \times C}$, $b^k \in \mathbb{R}^I$, $c^k \in \mathbb{R}^I$, subset of sensors $K_y \subset K_1^N$ and a trainable network $\mathcal{N} : K_y \to \mathbb{R}^{C \times N}$ with $K_y \subset K_2$. Then the following inequality holds*

$$\left| G(u)(x) - \sum_{k=1}^{K} a(W_x^k \cdot x + b_x^k)(c^k)^\intercal a\left(W^k \mathcal{N}(\hat{y})\hat{u} + b^k\right) \right| < \epsilon,$$

for all $x \in K_2$, $\hat{u} = [u(y_1), u(y_2), \dots, u(y_N)]^\intercal$, $\{y_i\} \subset K_y$, and $u \in V$.

The basis enhanced learning (BelNet) extension partially relaxes the restriction on the input function discretization. However, the number of sensors (input function mesh size) should still be the same for all input functions. Additionally, $K_y \subset K$ which supports the discretization-invariant property must be carefully crafted (see Remarks 3 and 4 in [10]). This limits the sensor placement and input function discretizations.

When dealing with datasets that have input functions with different properties, it is important to place sensors that can adapt to functions with the lowest regularity within the dataset.

Fig. 2 illustrates various functions with different regularities. To effectively capture functions with features like black curves with sharp peaks, a large number of sensors must be used. Similarly, for the magenta curves with smooth oscillations, sensors need to be strategically placed across the entire domain. Using a uniform sensor configuration for all input functions would require a significant increase in the number of sensors. This would result in higher computational costs and additional expenses, especially for smoother functions. We propose the D2NO which can effectively address this problem.

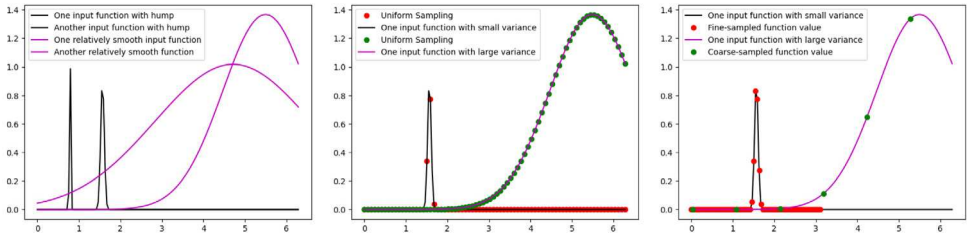

Fig. 2. Left: illustrations of functions exhibiting varied degrees of smoothness. The black curves represent two prototypes from a category of input functions characterized by sharp peaks, while the magenta curves display two instances of smooth functions. Right: presentation of different sensor counts for discretizing functions depicted in the left visual. The functions with humps are finely discretized with 75 sensors, whereas the smooth magenta function is captured using only 6 sensors (indicated by the green dots). Middle: uniform sampling. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.3. Distributed/federated training of DeepONets

In [48], the authors proposed Fed-DeepONet, a distributed/federated strategy that trains a consensus/universal DeepONet model for C clients and a centralized server. This federated training strategy involves looping over the following three steps:

1. *Broadcast to clients:* The centralized server broadcasts the most up-to-date DeepONet consensus model to all clients.
2. *Client local updates:* For any $c \in \{1, 2, \dots, C\}$, the cth client receives the most up-to-date DeepONet consensus model and performs $K \geq 1$ local stochastic gradient (or variants, e.g., Adam) DeepONet updates using the client's dedicated dataset.
3. *Global synchronization:* The centralized server aggregates the local DeepONet parameters into a unique consensus/universal DeepONet every K local update.

Since Fed-DeepONet learns a consensus model, each client must use the same number of sensors. In the next section, we will expand on the previously mentioned federated training strategy and develop a dedicated distributed training strategy for DeepONets with C clients. This strategy will efficiently handle heterogeneous input function spaces by allowing different numbers of sensors for each client $c \in \{1, 2, \dots, C\}$.

3. Methodology

Consider a nonlinear operator G that maps an input function $u \in V$ to an output function $G(u)(x)$ evaluated at an arbitrary location $x \in K_2$. According to Theorem 2.1, we can approximate $G(u)(x)$ as $\sum_{k=1}^{K} p_k(u) b_k(x)$, where $b_k(x)$ and $p_k(u)$ are learnable functions and functionals. In the DeepONet literature, $p_k(\cdot)$ is commonly referred to as the branch net, while $b_k(\cdot)$ is known as the trunk network. To simplify our notation, we will omit the parameters of the learnable functions whenever possible. It is important to note that $b_k(x)$ can be seen as the basis of the output function space, which is a shared element among all u and is independent of $u \in V$. On the other hand, $p_k(u)$ represents the coefficients in the linear combination, which depend on the specified input function u. Consequently, the approach taken is to develop a dedicated distributed approach that shares $b_k(x)$ across all samples to create a universal/consensus trunk network model. In this dedicated approach, individual clients independently manage their respective input function datasets using their branch nets (or projection in BelNet). They send their individual basis $b_k(x)$ to a centralized server to learn a universal/consensus model applicable to all samples. Please refer to Fig. 3 for an illustration and a comparison with classical training (Fig. 4).

This means that each client independently manages input functions in its dataset through its unique branch (or projection) nets. Additionally, they transmit their local trunk networks to the central server responsible for acquiring and sending the universal/consensus basis $b_k(x)$ across all samples. Since each dataset is managed by its individual branch (or projection), it becomes possible to use designated sensors tailored to the input function within each dataset.

Distributed Deep Neural Operator (D2NO). Let D denote the entire heterogeneous dataset, and $D^1, \dots, D^C$ to be sub-datasets. Moreover, each D^i exclusively comprises functions belonging to a single class, sharing similar regularity assumptions. Let us denote α^c as the trainable parameters of cth client's dedicated neural network. Let β denote the trainable parameter of the shared network. Thus in this task, α^c are the branch net parameters, and β are the trunk net parameters. We then denote $\alpha = \bigcup_{c=1}^{C} \alpha^c$ and $\theta = \alpha \bigcup \beta$. The local loss function for cth client based on its corresponding dataset D^c will be:

$$L^c(\alpha^c; \beta) = \sum_{i=1}^{N_c} \|G(u_i^c) - G_{\alpha^c,\beta}(\hat{u}_i^c)\|^2, \tag{1}$$

where N_c denotes the number of input functions in dataset D^c, u_i^c and $\hat{u}_i^c$ are the ith function in cth sub-dataset and its discretization at all sensors. We use the local loss to update the local parameters α^c associated with its own dedicated dataset D^c, the shared parameter β remains unchanged during this process but plays a role in formulating the loss functions and optimizing the system. Please note that we normalize the loss function with the number of samples during the implementation, and present the unnormalized version

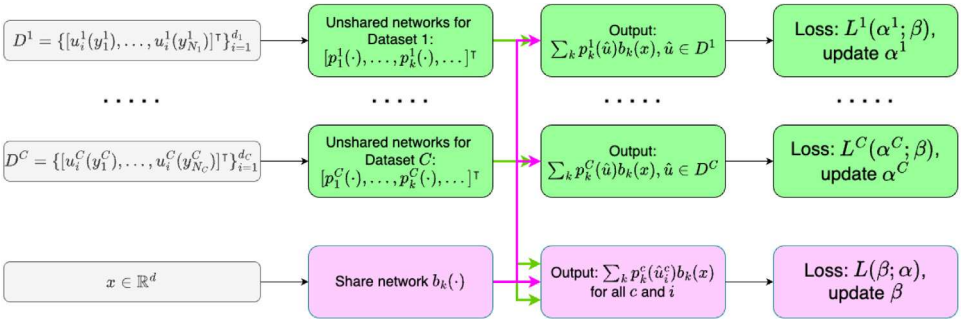

Fig. 3. D2NO process demonstration. Here y_i^c is ith sensor for c-th dataset functions, d_c denotes the number of functions in c-th dataset. We process the sub-dataset D^c by its dedicated branch nets (green box), together with the shared basis b_k (pink box), we can construct the local loss L^c, and use the local loss to optimize the specialized branch net parameters. The global loss construction relies on the entire dataset as it is used to update the trunk net parameters (pink box) which are shared across different clients. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

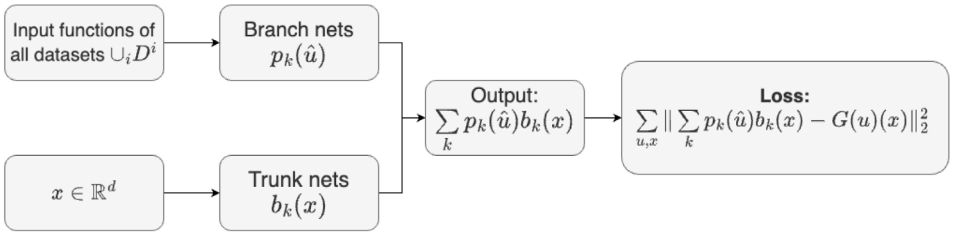

Fig. 4. Classical DeepONet training process demonstration.

for a better illustration of the concept. We can then define the global loss function used to update the shared parameter α given all the local parameters β and using all data,

$$L(\beta; \alpha) = \sum_{c=1}^{C} L^c(\alpha^c; \beta),$$

where C denotes the total number of clients and datasets. We now summarize the pseudo-algorithm in Algorithm 1 with an illustration in Fig. 3.

Algorithm 1: Distributed Deep Neural Network (D2NO) Algorithm

Initialization: weights w_c, all networks trainable parameters θ, learning rate η_c and η, the total number of iterations N. **for** $k = 1$ *to* N **do**

  Update the unshared parameters α_k^c for cth client using its dedicated sub-dataset, k denotes the kth optimization step. **for** $c = 1$ *to* C **do**

$$\alpha_k^c = \alpha_k^c - \eta_c \nabla_{\alpha^c} L^c(\alpha^c; \beta).$$

  Update the shared construction net parameter β using all data

$$\beta_k = \beta_k - \eta \nabla_\beta L(\beta; \alpha).$$

3.1. Efficiency of Distributed Deep Neural Operator (D2NO)

This section provides a brief analysis of the efficiency of the proposed D2NO in terms of data and computational cost. To train scientific machine learning models, a significant amount of data is needed, much of which could be privacy-sensitive. Surrogates trained using this data have the potential to greatly enhance scientific and engineering discovery and simulation. However, transferring this data to a centralized location is costly, and the quantity and security of the data may make training in such a location impractical using the traditional DeepONet approach. The proposed D2NO addresses this issue by allowing the training

data to remain distributed locally. Instead of sharing large datasets, the client only shares Deep Neural Operator parameters (or a partial model, e.g., the trunk network), which are aggregated using locally computed updates that reduce training costs.

To compare the computational training costs of the traditional DeepONet and D2NO, we proceed as follows. First, consider a centralized model with a vector of trainable parameters $\theta \in \mathbb{R}^p$. Without loss of generality, let us assume we select one point $x^{(i)}$ in each output function's domain, we then minimize the following loss function,

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}^i = \frac{1}{N}\sum_{i=1}^{N}\left|G_\theta(\hat{u}^{(i)})(x^{(i)}) - G(u^{(i)})(x^{(i)})\right|^2$$

over the dataset of N triplets $\mathcal{D}_{\text{cent}} = \{(\hat{u}^{(i)}, x^{(i)}), G(u^{(i)})(x^{(i)})\}_{i=1}^{N}$, here $\hat{u}^{(i)}$ is the function value sampled at sensors.

A standard deep learning package can compute the gradient of the loss function $\mathcal{L}$ with respect to each trainable parameter in β. In this analysis, we will focus on the number of gradients computed for each data loss function $\mathcal{L}^i$, where $i = 1, \dots, N$, with respect to the trainable parameters. This results in a total of $N \cdot p \cdot n_{\text{epochs}}$ gradient computations over n_{epochs} of training. This analysis does not take into account batching, but extending it to the stochastic setting is direct.

For the D2NO case, we assume that the data is distributed among C clients, so that $N = \sum_{j=1}^{C} n_j$, where n_j represents the number of data points available to the jth client for training a local model with a vector of $p' \le p$ trainable parameters. Therefore, for the jth client, the number of gradients computed for each data loss function over n_{epochs} is $n_j \cdot p' \cdot n_{\text{epochs}}$. As a result, we can establish the following inequality:

$$\begin{aligned}\text{Cost of D2NO} &= \sum_{j=1}^{C} n_j \cdot p' \cdot n_{\text{epochs}}\\ &= N \cdot p' \cdot n_{\text{epochs}}\\ &\le N \cdot p \cdot n_{\text{epochs}}\\ &= \text{Cost of Traditional DeepONet.}\end{aligned}$$

The inequality above indicates that D2NO has a training cost similar to or even less than that of the traditional DeepONet. Furthermore, this inequality is strict for certain examples presented later in this paper. For instance, when comparing the proposed D2NO (with two clients having a number of sensors $m_1 = 10$ and $m_2 = 100$) to the traditional DeepONet, which uses $m = 100$ sensors, they both perform an equal number of stochastic gradient updates and use the same amount of training data samples. However, our proposed framework is more efficient because over half of these stochastic gradient updates reduce the input function size by 10x, resulting in a decrease in the computational cost of training.

We conclude this section with the following remark. Although Algorithm 1 uses multiple client networks, which increases the total number of trainable parameters, it is important to note that, as demonstrated in this section, the total number of gradient updates does not increase.

4. Theoretical analysis

In this section, we demonstrate that the proposed D2NO satisfies a universal approximation theorem. We begin by stating the following lemma [1], which establishes a universal approximation theorem for functions in a compact subspace with compact support.

Lemma 4.1 (*[1], Theorem 3*)**.** *Suppose H is compact in $\mathbb{R}^d$, $U \subset C(H)$ is also compact. Let $f \in U$ and for any $\epsilon > 0$, there exists an integer $K > 0$ independent of f, continuous linear functionals c_k on U, networks $b_k : \mathbb{R}^d \to \mathbb{R}$ such that*

$$\left|f(y) - \sum_{k=1}^{K} c_k(f) b_k(y)\right| < \epsilon,$$

for all $y \in H$ and $f \in U$. Specifically, the networks have the following structure,

$$b_k(y) = g(w_k \cdot y + p_k), y \in H \tag{2}$$

$w_k \in \mathbb{R}^d$, $p_k \in \mathbb{R}$ and g is a Tauber-Wiener function [1].

The next lemma extends the above approximation to functionals defined on a compact subspace of continuous functions.

Lemma 4.2 (*[1], Theorem 4*)**.** *Suppose that Y is a Banach space, $K \subset Y$ is compact, and $U \subset C(K)$ is also compact. Let f be a continuous functional on U. For any $\epsilon > 0$, there exist weights c_k, $\{y_i\}_{i=1}^{I} \subset K$ and networks $b_k : \mathbb{R}^I \to \mathbb{R}$ such that,*

$$\left|f(u) - \sum_{k=1}^{K} c_k b_k(\hat{u})\right| < \epsilon,$$

for all $u \in V$, here $\hat{u} = [u(y_1), \dots, u(y_I)]^\top$. Specifically, b_k has a structure:

$$b_k(\hat{u}) = g(w_k \cdot \hat{u} + p_k), \tag{3}$$

where $w_k \in \mathbb{R}^I$, $p_k \in \mathbb{R}$ and g is a Tauber-Wiener function.

Without loss of generality, we consider two clients ($C = 2$) in the remainder of this paper. Let $G : V \to U$ be a continuous operator. We assume that $G_1 : V_1 \to U$ and $G_2 : V_2 \to U$ are the restrictions of G on V_1 and V_2 respectively. Specifically, $G(u) = G_1(u)$ for any $u \in V_1$ and $G(u) = G_2(u)$ for any $u \in V_2$. We will construct the universal approximation for both G_1 and G_2 using a shared output function basis. Let us make the following assumptions.

Assumption 4.1. We make the following assumptions on the input/output function spaces and the operator.

1. $V_1 \subset C(K_1)$, $V_2 \subset C(K_2)$ are compact and are defined on compact domains $K_1 \subset K_1'$ and $K_2 \subset K_2'$, where K_1' and K_2' are Banach spaces. Additionally, $V = V_1 \cup V_2$.
2. Let G, G_1 and G_2 are continuous, and $G(u) = G_1(u)$ for any $u \in V_1$ and $G(u) = G_2(u)$ for any $u \in V_2$.
3. $U = G_1(V_1) \cup G_2(V_2)$ has a compact domain $K_x \subset \mathbb{R}^d$.

The following theorem establishes the universal approximation of continuous nonlinear operators.

Theorem 4.3. *Let $G : V \to U$, $G_1 : V_1 \to U$ and $G_2 : V_2 \to U$ satisfy Assumption 4.1. For any $\varepsilon > 0$, there exist weights $c_{i,k}^1, c_{i,k}^2 \in \mathbb{R}$, $[y_1, \dots, y_{N_1}]^\top \subset K_1$, $[z_1, \dots, z_{N_2}]^\top \subset K_2$, networks $b_{i,k}^1 : \mathbb{R}^{N_1} \to \mathbb{R}$, $b_{i,k}^2 : \mathbb{R}^{N_2} \to \mathbb{R}$, specified in Eq. (3), and shared $b_k : K_x \to \mathbb{R}$ specified in (2) such that,*

$$|G_1(u_1)(x) - \sum_{k=1}^{K} \sum_{i=1}^{I_1} c_{i,k}^1 b_{i,k}^1(\hat{u}_1) b_k(x)| < \varepsilon,$$

$$|G_2(u_2)(x) - \sum_{k=1}^{K} \sum_{i=1}^{I_2} c_{i,k}^2 b_{i,k}^2(\hat{u}_2) b_k(x)| < \varepsilon,$$

for any $u_1 \in V_1$ and $u_2 \in V_2$, here $\hat{u}_1 = [u(y_1), \dots, u(y_{N_1})]^\top$ and $\hat{u}_2 = [u(z_1), \dots, u(z_{N_2})]^\top$.

Proof. Without loss of generality, we will prove the approximation for the first client. For any $\frac{\varepsilon}{2} > 0$, by Lemma 4.1, there exist K networks $b_k(x)$, functional $a_k(\cdot)$ on U such that,

$$|G(u)(x) - \sum_{k=1}^{K} a_k\left(G(u)\right) b_k(x)| < \frac{\varepsilon}{2}, \tag{4}$$

for any $u \in V$ and $x \in K_x$. We prove the case for the first client, hence assume $u \in V_1$, and $G_1(u) = G(u)$ for $u \in V_1$, it follows that $|G_1(u)(x) - \sum_{k=1}^{K} a_k\left(G_1(u)\right) b_k(x)| < \frac{\varepsilon}{2}$. Denote $L = \sum_{k=1}^{K} \sup_{x \in K_x} |b_k(x)|$, it follows by Lemma 4.2 that, for $\frac{\varepsilon}{2L}$ and any $k = 1, \dots, K$, there exist weights $c_{i,k}^1$, $[y_1, \dots, y_{N_1}] \subset K_1$ and networks $b_{i,k}^1 : \mathbb{R}^{N_1} \to \mathbb{R}$ such that,

$$|a_k\left(G(u)\right) - \sum_{i=1}^{I_1} c_{i,k}^1 b_{i,k}^1(\hat{u}_1)| < \frac{\varepsilon}{2L}, \tag{5}$$

where $\hat{u}_1 = [u(y_1), \dots, u(y_{N_1})]^\top$. Let I_1 be large enough and set $c_{i,k}^1 = 0$ if necessary, substitute Eq. (5) into Eq. (4), it follows that,

$$|G_1(u)(x) - \sum_{k=1}^{K} \sum_{i=1}^{I_1} c_{i,k}^1 b_{i,k}^1(\hat{u}_1) b_k(x)| < \varepsilon,$$

for any $u \in V_1$ and $x \in K_x$. The approximation for G_2 is similar given $u \in V_2$. □

5. Numerical experiments

In this section, we use four numerical experiments to demonstrate the effectiveness of D2NO in approximating nonlinear operators with heterogeneous input function spaces.

Notably, in real engineering and industrial scenarios, each client possesses its distinct dataset, facilitating automatic dataset division. In our simulation experiments, we partition the dataset according to the input function generators. For instance, as depicted in Fig. 2 of the numerical experiments detailed in Section 5.1.2, we employ Gaussian distributions to generate these functions. Subsequently, the input functions are automatically segregated based on the different variances of the distribution.

5.1. Viscous Burgers' equation

We begin by studying the viscous Burgers' equation. Our main objective is to understand and approximate the operator that maps the initial condition of the PDE to the solution at the terminal time. Specifically, the viscous Burgers' equation is defined as follows:

$$\frac{\partial u_s}{\partial t} + \frac{1}{2} \frac{\partial (u_s^2)}{\partial x} = \alpha \frac{\partial^2 u_s}{\partial x^2}, \; x \in [0, 2\pi], \; t \in [0, T], \tag{6}$$

with the initial condition $u_s(x, 0) = u_s^0(x)$ and the boundary conditions $u_s(0, t) = u_s(2\pi, t)$. Here, $u_s^0(x)$ is the initial condition that depends on the parameter s, and α is the viscosity of the equation. Our goal is to learn the mapping from u_s^0 to $u_s(t)$ [8,10,49,50].

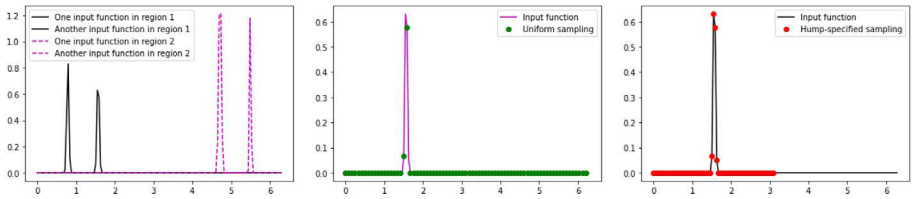

Fig. 5. Left: the input functions exhibit peaks, with these peaks being concentrated in two distinct regions (black and magenta). Such a dataset poses significant challenges to the conventional operator learning framework, which mandates uniform sensor placement for all input functions. This is due to the necessity for dense sampling to effectively capture the sparsely distributed peaks. However, this approach results in a substantial increase in problem dimensionality, leading to the presence of numerous redundant entries in the discretized vectors, please check the green dots in the middle image. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
The average L_2-relative error computed from 50 independent training runs for both the DeepONet and D2NO described in Algorithm 1. As a reference, we predict by the average of the training labels, the relative error is 97.59%.

	DeepONet	D2NO
L_2-relative error	18.64%	8.10%

5.1.1. Heterogeneous input functions with double sharp peaks

Firstly, we focus our attention on the challenging scenario described in Fig. 5, where input functions display heterogeneous properties. Specifically, these input functions exhibit distinct sharp peaks, which are concentrated in two separate regions (depicted by the black and magenta curves in Fig. 5).

In the traditional operator learning or DeepONet framework, uniform sensor placement is required for all input functions. Thus, only a limited number of sensors are used to sample the peak of one input function, and most of these sensors provide the same function values. This uniformity of data significantly affects the quality of training.

By using Algorithm 1, we can use two clients ($C = 2$) and two dedicated branch networks to capture the input functions individually in the clustered regions. As a result, we can deploy separate sensor sets for these two datasets. Fig. 5 illustrates how a function from the left region is sampled using dedicated sensors tailored to that specific region. In contrast, Fig. 5 shows the same function sampled using uniform sensors for comparison.

To evaluate the performance of D2NO (see Algorithm 1), we trained a total of 50 independent D2NO models and calculated the average L_2-relative error. We compared D2NO with the traditional DeepONet model. Specifically, we trained 50 traditional DeepONet models using the same network structure and hyperparameters as D2NO. It is important to note that for the traditional DeepONet, we used uniform sensors for all input functions. Table 1 presents a summary of the results, which demonstrate that the proposed D2NO algorithm significantly reduces relative errors while maintaining a comparable computational cost.

5.1.2. Heterogeneous input functions with different properties

In this experiment, we explore the scenario depicted in Fig. 2. This illustration reflects many other situations where using different numbers of sensors for different input functions can result in improved solutions. Previous attempts have focused on expanding the capabilities of neural operators to discretize different input functions in different ways. However, these efforts have been limited by the need for an equal number of sensors. Our new algorithm overcomes this limitation by allowing the use of varying quantities of sensors for function discretization.

In our example, we present two different types of input functions: (a) functions with humps, represented by the black curve in Fig. 2, and (b) functions with relatively smooth profiles, represented by the magenta curve in the same figure. Typically, the conventional approach involves using the same number of sensors for all functions. However, due to the presence of humps in the first function category, a larger number of sensors is required for both types of functions. The numerical examples demonstrate that DeepONet-style networks can effectively handle challenges presented by smooth input functions, even with a limited number of sensors. However, for functions with peaks, a higher number of sensors is required, which increases the computational cost when dealing with smoother functions.

Since the dedicated branch networks are not shared among clients, our algorithm enables the creation of separate branch networks for these two function classes. As a result, the sensor count and sensor placements can vary depending on the specific function being considered. An illustration of this is in Fig. 2.

We use two clients, each with its own dedicated branch network. For functions that have peak characteristics, we use a larger two-layer branch network with dimensions $75 \times 100 \to 100 \times 1$. On the other hand, for smoother functions, we employ a smaller two-layer branch network with dimensions $6 \times 50 \to 50 \times 1$. Thus, the proposed D2NO requires the use of 75 sensors and 6 sensors, respectively.

Note that our proposed distributed algorithm (D2NO) adjusts the sensor count for different input function spaces, unlike the traditional DeepONet approach which uses a uniform configuration of 75 sensors for all input functions. Table 2 presents the aggregated results after training 50 independent models using both the traditional DeepONet and the proposed strategy.

Table 2
The average L2-relative error of 50 independent trainings for DeepONet and D2NO (see Algorithm 1). Two dedicated branch networks with distinct structures are used to capture hump input functions and smooth input functions, respectively. Note that this approach helps to reduce the number of trainable parameters. As a reference, if we were to predict the solution by the average of the training samples, the relative error would be 810.61%.

	DeepONet	D2NO
L_2-relative error	7.53%	5.34%
# unshared parameters	$77K$	$41.5K$
# shared parameters	$51.8K$	$51.8K$

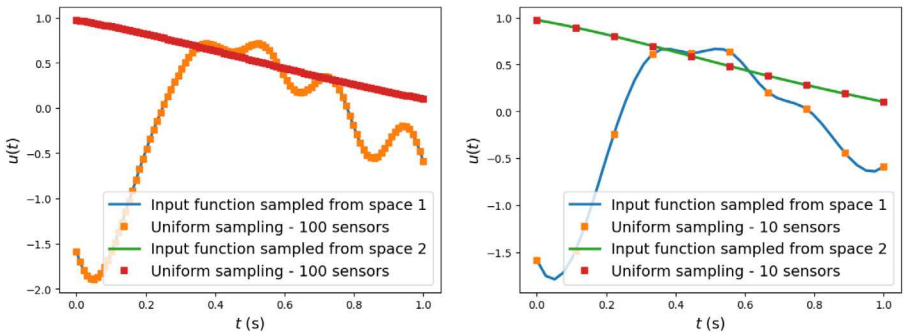

Fig. 6. *(Left):* Input functions discretized with uniform sampling and $m = 100$ sensors. *(Right):* Input functions discretized with uniform sampling and $m = 10$ sensors. The input functions exhibit different frequencies. Such a dataset poses significant challenges to the conventional operator learning framework, which mandates uniform sensor placement for all input functions. This is due to the necessity for dense sampling to effectively capture the oscillatory behavior. However, this approach results in a substantial increase in problem dimensionality, leading to the presence of numerous redundant entries in the discretized vectors.

5.2. The nonlinear pendulum system

In this experiment, we use the proposed distributed strategy D2NO to approximate the solution operator of the nonlinear pendulum system with dynamics:

$$\begin{aligned} \frac{ds_1(t)}{dt} &= s_2(t), \\ \frac{ds_2(t)}{dt} &= -k\sin(s_1(t)) + u(t). \end{aligned} \tag{7}$$

Here, $s(t) = (s_1(t), s_2(t))^\top$ is the state defined over the time domain $t \in [0, 1]$ and $u(t)$ is the external force. We aim to approximate the solution operator $G : u(t) \mapsto s(t)$ for a fixed initial condition $(s_1(0), s_2(0)) = (0.0, 0.0)$, $k = 1.0$, and an external force $u(t)$ sampled from the mean-zero Gaussian Random Field (GRF):

$$u \sim \mathcal{G}(0, \kappa_\ell(x_1, x_2)), \tag{8}$$

where the covariance kernel $\kappa_\ell(x_1, x_2) = \exp\left(-\|x_1 - x_2\|^2/2\ell^2\right)$ is the radial-basis function (RBF) with length-scale parameter $\ell > 0$, which we will use to generate heterogeneous input function spaces, as described next.

Heterogeneous Input Functions with Different Frequencies. We first consider two input function spaces with different frequencies. To this end, we let the first input space V_1 be a GRF with length scale $\ell = 0.1$ and the second input space V_2 be a GRF with length scale $\ell = 1.0$. Fig. 6 illustrates sample functions from these two input spaces. The centralized DeepONet framework requires uniformly placing a collection of m sensors to discretize the input function u. In general, as depicted in Fig. 6, a large number of sensors (e.g., $m = 100$) is required to capture effectively the input function u, which, in turn, increases the training cost.

We note that for the input function sampled from the space V_2, using $m = 100$ sensors is excessive. One may require only $m = 10$ sensors (see Fig. 6) to effectively capture input functions sampled from V_2. However, as illustrated in Fig. 6, using less number of sensors will deteriorate our ability to describe the oscillations from the input function sampled from the input space V_1. The proposed distributed framework D2NO can effectively tackle the above problem by using two clients ($C = 2$), each with dedicated branch networks, as depicted in Fig. 7. More specifically, the first client uses a branch network with $m = 100$ sensors to capture and discretize input functions sampled from the input space V_1. The second client uses a branch network with $m = 10$ sensors to capture and discretize input functions sampled from V_2. We let the two clients share and synchronize the trunk network.

Training and Testing. To ensure a fair comparison between the centralized DeepONet and D2NO, we used the same neural network architectures, with the exception of the input layer of the dedicated branch networks for D2NO. Specifically, we used a feed-forward neural network architecture with one hidden layer and leaky-ReLU activation functions for both branch and trunk networks. The hidden and output layers of these networks consist of 50 neurons. Table 3 summarizes the number of parameters for the resulting DeepONets and D2NO.

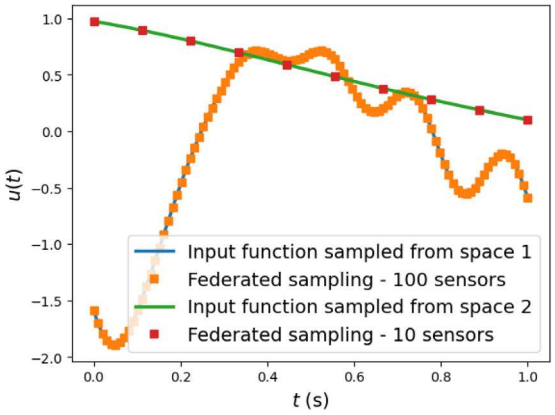

Fig. 7. Input functions discretized with federated sampling. A client with dedicated branch networks uses $m = 100$ sensors to discretize the input function sampled from a GRF with length-scale $\ell = 0.1$. Another client with a dedicated branch network uses $m = 10$ sensors to discretize the input function sampled from a GRF with length-scale $\ell = 1.0$.

Table 3
The number of parameters for the resulting DeepONets with $m = 100$ and $m = 10$ sensors, as well as D2NO. Note that for the DeepONets, we detail the number of parameters for both the trunk and branch networks. On the other hand, for D2NO, we specify the number of parameters for the shared trunk network and the dedicated branch networks for clients 1 and 2.

	DeepONet ($m = 100$)	DeepONet ($m = 10$)	D2NO
Trunk/shared trunk	2650	2650	2650
Branch/branch-1	7600	3100	7600
Branch-2	–	–	3100

Table 4
The average L_2−relative error for 100 test trajectories sampled from the input space V_1 (GRF with length-scale $\ell = 0.1$) and for 100 test trajectories sampled from the input space V_2 (GRF with length-scale $\ell = 1.0$), using the centralized DeepONet ($m = 100$ and $m = 10$ sensors) and the distributed framework D2NO detailed in Algorithm 1.

	DeepONet ($m = 100$)	DeepONet ($m = 10$)	D2NO
L_2−rel. error on V_1	2.90%	12.35%	4.26%
L_2−rel. error on V_2	0.62%	1.25%	1.57%

To train both DeepONets and D2NO, we sampled $N_{V_1} = N_{V_2} = 1,000$ input trajectories from each input space. The traditional/centralized DeepONet was trained using stochastic gradient-based optimization for 1000 epochs. Subsequently, each model within the proposed D2NO approach was trained using 1000 synchronization rounds after one local stochastic gradient update. It should be noted that the accuracy of D2NO could be improved by increasing the number of local gradient updates, as per federated learning approaches. However, we avoided this strategy to ensure a fair comparison of computational costs. Finally, to test the proposed framework, we sampled 100 test trajectories from both V_1 and V_2.

Performance of D2NO. Table 4 shows the performance (i.e., in terms of the L_2- relative error) of the proposed D2NO and the centralized DeepONet (with $m = 100$ and $m = 10$ sensors) for the test trajectories sampled from the input spaces V_1 and V_2. The results clearly illustrate that the proposed D2NO framework is competitive with the centralized DeepONet with $m = 100$ sensors and vastly outperforms the more efficient DeepONet with $m = 10$ sensors. Specifically, D2NO delivers a predictive accuracy for both input spaces V_1 and V_2, comparable to a DeepONet with $m = 100$ sensors, matches the accuracy of a DeepONet with $m = 10$ sensors on the input space V_2, and significantly outperforms a DeepONet with $m = 10$ sensors on V_1.

Computational Cost Analysis. Table 5 compares the number gradient computations for D2NO and DeepONet using $m = 10$ and $m = 100$ sensors after $n_{\text{epochs}} = 1000$ epochs of training. The results highlight the advantages of adapting the number of sensors with D2NO. Specifically, D2NO delivers a predictive accuracy for both input spaces V_1 and V_2, comparable to a DeepONet with $m = 100$ sensors (as seen in Table 4) but with fewer gradient computations. D2NO also matches the accuracy of a DeepONet with $m = 10$ sensors on the input space V_2. Moreover, D2NO significantly outperforms a DeepONet with $m = 10$ sensors on test trajectories generated from inputs sampled from V_1. Therefore, D2NO strikes a balance between efficiency and accuracy for DeepONets trained using heterogeneous input spaces. Finally, from these results, we anticipate that as the number of distributed clients increases (for example, due to more heterogeneous input functions), the savings from D2NO will also increase.

Using Distinct Number of Training Samples. Here, we test the proposed D2NO framework on an imbalanced scenario where the number of training trajectories sampled from V_1 is $N_{V_1} = 400$ and the number of training trajectories sampled from V_2 is $N_{V_2} = 1,600$.

Table 5
Comparison of the number of gradient computations for D2NO, which balances efficiency and accuracy, a DeepONet that uses $m = 10$ sensors for efficiency but lacks accuracy, and a costly but accurate DeepONet that uses $m = 100$ sensors, after $n_{\text{epochs}} = 1000$ epochs of training for the pendulum example.

	DeepONet ($m = 100$)	DeepONet ($m = 10$)	D2NO
grad. computations	3.28×10^8	1.84×10^8	2.56×10^8

Table 6
The average L_2-relative error for 100 test trajectories sampled from the input space V_1 (GRF with length-scale $\ell = 0.1$) and for 100 test trajectories sampled from the input space V_2 (GRF with length-scale $\ell = 1.0$), using the centralized DeepONet ($m = 100$ and $m = 10$ sensors) and the distributed framework D2NO detailed in Algorithm 1. The frameworks were trained using $N_{V_1} = 400$ input trajectories sampled from V_1 and $N_{V_2} = 1600$ input trajectories sampled from V_2.

	DeepONet ($m = 100$)	DeepONet ($m = 10$)	D2NO
L_2-rel. error on V_1	5.87%	16.09%	8.09%
L_2-rel. error on V_2	0.84%	1.01%	1.48%

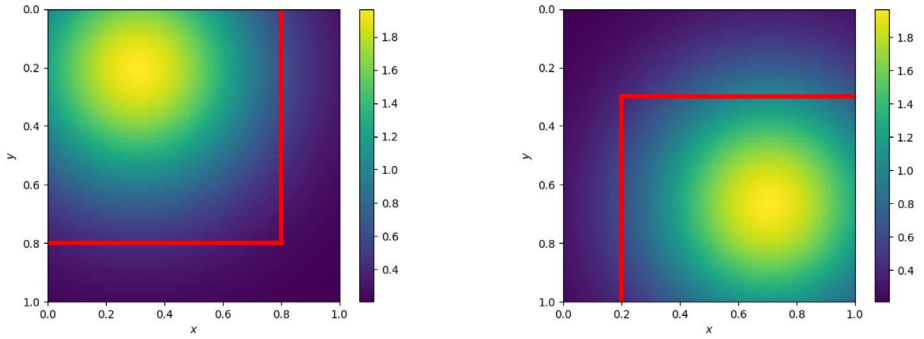

Fig. 8. *(Left):* The input is sampled from the input space, V_1. To train the centralized DeepONet, sensor locations are defined over the domain $[0.0, 1.0] \times [0.0, 1.0]$. In contrast, to train D2NO, sensor locations are defined over the smaller domain $[0.0, 0.8] \times [0.0, 0.8]$, where the non-zero input values are concentrated. *(Right):* The input is sampled from the input space, V_2. To train the centralized DeepONet, sensor locations are defined over the domain $[0.0, 1.0] \times [0.0, 1.0]$. In contrast, to train D2NO, sensor locations are defined over the smaller domain $[0.2, 1.0] \times [0.3, 1.0]$, where the non-zero input values are concentrated.

Table 6 describes the performance of the proposed D2NO and the centralized DeepONet. The results clearly demonstrate the advantages of the proposed D2NO framework. Specifically, it outperforms the DeepONet with $m = 10$ sensors on V_1 and matches its performance on V_2, even in a scenario that favors this DeepONet architecture.

5.3. 2D-elliptic partial differential equation

In our last experiment, we used the proposed D2NO to approximate the solution operator of a two-dimensional PDE, specifically, the 2D-elliptic equation:

$$-\nabla \cdot \kappa(x) \nabla u = f, x \in [0, 1]^2, \tag{9}$$

and we consider the nonlinear mapping from the permeability $\kappa(x)$ to the solution $u(x)$.

Heterogeneous Input Spaces. To approximate the solution operator, we considered heterogeneous functions sampled from two input spaces, denoted as V_1 and V_2. We constructed samples from V_1 using the Gaussian distribution with the mean centered in the region $[0, 0.5] \times [0, 0.5]$ as shown in Fig. 8 Left. Similarly, we obtained samples from V_2, but the mean of the Gaussian distribution is from region $[0.5, 1] \times [0.5, 1]$ (shown in Fig. 8 Right). We constructed a centralized DeepONet and a D2NO with two clients. These were used to approximate the solution operator of the 2D-elliptic equation using inputs sampled from the heterogeneous spaces V_1 and V_2.

Fig. 8 illustrates random samples from V_1 and V_2. We generated input samples for V_1 and V_2 using a uniform grid of size 100×100 over the entire input domain $[0.0, 1.0] \times [0.0, 1.0]$. The inputs to the branch network of the centralized DeepONet were discretized using $m = 1000$ subsamples from the uniform grid that covers the entire input domain. In contrast, the dedicated branch networks of D2NO used discretized input samples with $m_{V_1} = 640$ and $m_{V_2} = 560$ sensors, respectively. These sensors were located by subsampling the uniform grid over smaller regions: $[0.2, 0.8] \times [0.2, 0.8]$ for V_1 and $[0.2, 1.0] \times [0.3, 1.0]$ for V_2.

Training and Testing. As with the pendulum example, we ensured a fair comparison between the centralized DeepONet and D2NO by using identical neural network architectures, except for the input layer of the dedicated branch networks for D2NO. Specifically, we employed a feed-forward neural network architecture with two hidden layers and leaky-ReLU activation functions for both branch

Table 7
The number of parameters for the centralized DeepONet and the proposed D2NO. Note that for the DeepONet, we detail the number of parameters for both the trunk and branch networks. On the other hand, for D2NO, we specify the number of parameters for the shared trunk network and the dedicated branch networks for clients 1 and 2.

	DeepONet	D2NO
Trunk/shared trunk	20,500	20,500
Branch/branch-1	120,300	84,300
Branch-2	–	76,300

Table 8
The average L_2−relative error for 20 input functions sampled from the input space V_1 and for 20 test trajectories sampled from the input space V_2 for the 2D-elliptic PDE, using the centralized DeepONet and the distributed framework D2NO detailed in Algorithm 1.

	DeepONet	D2NO
L_2−rel. error on V_1	1.99%	1.54%
L_2−rel. error on V_2	1.75%	2.12%

Table 9
Comparison of the number of gradient computations for D2NO and a centralized DeepONet for the 2D-elliptic PDE example, after $n_{\text{epochs}} = 1000$ epochs of training.

	DeepONet	D2NO
grad. computations	2.53×10^9	1.81×10^9

and trunk networks. Each network's hidden and output layers contain 100 neurons. Table 7 summarizes the parameter count for the resulting centralized DeepONet and D2NO.

To train both DeepONets and D2NO, we sampled $N_{V_1} = N_{V_2} = 80$ input functions from each input space. The traditional/centralized DeepONet was trained using stochastic gradient-based optimization for 1000 epochs. Subsequently, each model within the proposed D2NO approach was trained using 1000 synchronization rounds after one local stochastic gradient update. Finally, to test the proposed framework, we sampled 20 test input functions from both V_1 and V_2.

Performance of D2NO. Table 8 displays the performance of the proposed D2NO and the centralized DeepONet, evaluated by the L_2- relative error, for the test input functions sampled from the input spaces V_1 and V_2. The results reveal that D2NO provides predictive accuracy that is comparable to or even surpasses, the centralized DeepONet for both input spaces, V_1 and V_2.

Computational Cost Analysis. We conclude this example by illustrating the savings in gradient computations for D2NO applied to the 2D elliptic PDE in Table 9. These savings occurred after $n_{\text{epochs}} = 1000$ epochs of training. It is important to note that D2NO matches the accuracy of the centralized DeepONet, as demonstrated in Table 8, while carrying out fewer gradient computations.

6. Conclusion

In this paper, we have developed a dedicated distributed training strategy for deep neural operators (D2NO) that can efficiently handle heterogeneous input function spaces. Our strategy involves partitioning the input function space based on the properties and regularity of the input functions. Each subset of the input function space is then processed using a dedicated branch network, which is trained locally. Additionally, a consensus trunk network is trained in a centralized manner. This trunk network acts as a global basis for the local coefficients obtained from the dedicated branch networks. We have demonstrated that D2NO serves as a universal approximator of continuous nonlinear operators. Finally, we have demonstrated that D2NO not only addresses the challenges posed by input functions with heterogeneous properties but also offers improved flexibility and computational efficiency using four numerical examples, each with heterogeneous input function spaces. In the future, we will consider non-uniform sensors within each sub-dataset and apply the algorithms to discretization invariant neural operators such as BelNet. Another potential work is to study reducing the samples used in training the shared structure and study the error growth.

CRediT authorship contribution statement

Zecheng Zhang: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Christian Moya:** Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Lu Lu:** Writing – review & editing. **Guang Lin:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition. **Hayden Schaeffer:** Writing – review & editing, Supervision, Resources, 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

Data will be made available on request.

Acknowledgments

Z. Zhang was supported in part by AFOSR MURI, USA FA9550-21-1-0084. H. Schaeffer was supported in part by AFOSR MURI, USA FA9550-21-1-0084, National Science Foundation (NSF), USA DMS-2331033. G. Lin acknowledges the support of the National Science Foundation, USA (DMS-2053746, DMS-2134209, ECCS-2328241, and OAC-2311848), and U.S. Department of Energy (DOE) Office of Science Advanced Scientific Computing Research program DE-SC0023161, and the Uncertainty Quantification for Multifidelity Operator Learning (MOLUcQ) project (Project No. 81739), and DOE–Fusion Energy Science, USA, under grant number: DE-SC0024583. L. Lu was supported by the U.S. Department of Energy, USA [DE-SC0022953].

References

[1] 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.
[2] T. Chen, H. Chen, Approximations of continuous functionals by neural networks with application to dynamic systems, IEEE Trans. Neural Netw. 4 (6) (1993) 910–918.
[3] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440.
[4] 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.
[5] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, 2020, arXiv preprint arXiv:2010.08895.
[6] E. Qian, I.-G. Farcas, K. Willcox, Reduced operator inference for nonlinear partial differential equations, SIAM J. Sci. Comput. 44 (4) (2022) A1934–A1959.
[7] P. Jin, S. Meng, L. Lu, MIONet: Learning multiple-input operators via tensor product, SIAM J. Sci. Comput. 44 (6) (2022) A3490–A3514.
[8] Z. Zhang, W.T. Leung, H. Schaeffer, BelNet: Basis enhanced learning, a mesh-free neural operator, 2022, arXiv preprint arXiv:2212.07336.
[9] S. Lanthaler, A.M. Stuart, The curse of dimensionality in operator learning, 2023, arXiv preprint arXiv:2306.15924.
[10] Z. Zhang, W.T. Leung, H. Schaeffer, A discretization-invariant extension and analysis of some deep operator networks, 2023, arXiv preprint arXiv:2307.09738.
[11] B. Deng, Y. Shin, L. Lu, Z. Zhang, G.E. Karniadakis, Approximation rates of DeepONets for learning operators arising from advection–diffusion equations, Neural Netw. 153 (2022) 411–426.
[12] 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. Engrg. 412 (2023) 116064.
[13] 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.
[14] S. Mao, R. Dong, L. Lu, K.M. Yi, S. Wang, P. Perdikaris, PPDONet: Deep operator networks for fast prediction of steady-state solutions in disk–planet systems, Astrophys. J. Lett. 950 (2) (2023) L12.
[15] Z. Mao, L. Lu, O. Marxen, T.A. Zaki, G.E. Karniadakis, DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators, J. Comput. Phys. 447 (2021) 110698.
[16] S. Cai, Z. Wang, L. Lu, T.A. Zaki, G.E. Karniadakis, DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks, J. Comput. Phys. 436 (2021) 110296.
[17] G. Lin, C. Moya, Z. Zhang, Learning the dynamical response of nonlinear non-autonomous dynamical systems with deep operator neural networks, Eng. Appl. Artif. Intell. 125 (2023) 106689.
[18] L. Bhan, Y. Shi, M. Krstic, Operator learning for nonlinear adaptive control, in: Learning for Dynamics and Control Conference, PMLR, 2023, pp. 346–357.
[19] C. Moya, S. Zhang, G. Lin, M. Yue, Deeponet-grid-uq: A trustworthy deep operator framework for predicting the power grid's post-fault trajectories, Neurocomputing 535 (2023) 166–182.
[20] Y. Sun, C. Moya, G. Lin, M. Yue, Deepgraphonet: A deep graph operator network to learn and zero-shot transfer the dynamic response of networked systems, IEEE Syst. J. (2023).
[21] K. Ye, J. Zhao, X. Liu, C. Moya, G. Lin, DeepONet based uncertainty quantification for power system dynamics with stochastic loads, in: 2023 IEEE Power & Energy Society General Meeting, PESGM, IEEE, 2023, pp. 1–6.
[22] L. Lu, R. Pestourie, S.G. Johnson, G. Romano, Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport, Phys. Rev. Res. 4 (2) (2022) 023210.
[23] I. Sahin, C. Moya, A. Mollaali, G. Lin, G. Paniagua, Deep operator learning-based surrogate models with uncertainty quantification for optimizing internal cooling channel rib profiles, Int. J. Heat Mass Transfer 219 (2024) 124813.
[24] A.A. Howard, M. Perego, G.E. Karniadakis, P. Stinis, Multifidelity deep operator networks, 2022, arXiv preprint arXiv:2204.09157.
[25] Z. Zhang, C. Moya, W.T. Leung, G. Lin, H. Schaeffer, Bayesian deep operator learning for homogenized to fine-scale maps for multiscale PDE, 2023, arXiv preprint arXiv:2308.14188.
[26] J. Pathak, S. Subramanian, P. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z. Li, K. Azizzadenesheli, et al., Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators, 2022, arXiv preprint arXiv:2202.11214.
[27] G. Lin, C. Moya, Z. Zhang, B-DeepONet: An enhanced Bayesian DeepONet for solving noisy parametric PDEs using accelerated replica exchange SGLD, J. Comput. Phys. 473 (2023) 111713.
[28] A.F. Psaros, X. Meng, Z. Zou, L. Guo, G.E. Karniadakis, Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons, J. Comput. Phys. 477 (2023) 111902.
[29] Y. Yang, G. Kissas, P. Perdikaris, Scalable uncertainty quantification for deep operator networks using randomized priors, Comput. Methods Appl. Mech. Engrg. 399 (2022) 115399.
[30] 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. Engrg. 393 (2022) 114778.
[31] P.C. Di Leoni, L. Lu, C. Meneveau, G.E. Karniadakis, T.A. Zaki, Neural operator prediction of linear instability waves in high-speed boundary layers, J. Comput. Phys. 474 (2023) 111793.
[32] Z. Jiang, M. Zhu, D. Li, Q. Li, Y.O. Yuan, L. Lu, Fourier-MIONet: Fourier-enhanced multiple-input neural operators for multiphase modeling of geological carbon sequestration, 2023, arXiv preprint arXiv:2303.04778.

[33] M. Zhu, S. Feng, Y. Lin, L. Lu, Fourier-DeepONet: Fourier-enhanced deep operator networks for full waveform inversion with improved accuracy, generalizability, and robustness, 2023, arXiv preprint arXiv:2305.17289.
[34] A. Jiao, H. He, R. Ranade, J. Pathak, L. Lu, One-shot learning for solution operators of partial differential equations, 2021, arXiv preprint arXiv:2104.05512.
[35] L. Cao, T. O'Leary-Roseberry, P.K. Jha, J.T. Oden, O. Ghattas, Residual-based error correction for neural operator accelerated infinite-dimensional Bayesian inverse problems, 2022, arXiv preprint arXiv:2210.03008.
[36] Y. Liu, Z. Zhang, H. Schaeffer, PROSE: Predicting operators and symbolic expressions using multimodal transformers, 2023, arXiv preprint arXiv:2309.16816.
[37] C. Lin, Z. Li, L. Lu, S. Cai, M. Maxey, G.E. Karniadakis, Operator learning for predicting multiscale bubble growth dynamics, J. Chem. Phys. 154 (10) (2021).
[38] G. Lin, C. Moya, Z. Zhang, Accelerated replica exchange stochastic gradient langevin diffusion enhanced Bayesian DeepONet for solving noisy parametric PDEs, 2021, arXiv preprint arXiv:2111.02484.
[39] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.
[40] T. Li, A.K. Sahu, A. Talwalkar, V. Smith, Federated learning: Challenges, methods, and future directions, IEEE Signal Process. Mag. 37 (3) (2020) 50–60.
[41] H.B. McMahan, E. Moore, D. Ramage, B.A. y Arcas, Federated learning of deep networks using model averaging, 2016, arXiv preprint arXiv:1602.05629, 2.
[42] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, et al., Large scale distributed deep networks, Adv. Neural Inf. Process. Syst. 25 (2012).
[43] X. Li, K. Huang, W. Yang, S. Wang, Z. Zhang, On the convergence of fedavg on non-iid data, 2019, arXiv preprint arXiv:1907.02189.
[44] X. Li, M. Jiang, X. Zhang, M. Kamp, Q. Dou, Fedbn: Federated learning on non-iid features via local batch normalization, 2021, arXiv preprint arXiv:2102.07623.
[45] A. Khaled, K. Mishchenko, P. Richtárik, First analysis of local gd on heterogeneous data, 2019, arXiv preprint arXiv:1909.04715.
[46] S.P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, A.T. Suresh, Scaffold: Stochastic controlled averaging for federated learning, in: International Conference on Machine Learning, PMLR, 2020, pp. 5132–5143.
[47] W. Deng, Q. Zhang, Y.-A. Ma, Z. Song, G. Lin, On convergence of federated averaging langevin dynamics, 2021, arXiv preprint arXiv:2112.05120.
[48] C. Moya, G. Lin, Fed-deeponet: Stochastic gradient-based federated training of deep operator networks, Algorithms 15 (9) (2022) 325.
[49] H. Schaeffer, R. Caflisch, C.D. Hauck, S. Osher, Sparse dynamics for partial differential equations, Proc. Natl. Acad. Sci. 110 (17) (2013) 6634–6639.
[50] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 473 (2197) (2017) 20160446.