ASME Journal of Computing and Information Science in Engineering
Online journal at:
https://asmedigitalcollection.asme.org/computingengineering

# Bayesian, Multifidelity Operator Learning for Complex Engineering Systems–A Position Paper

**Christian Moya**
Department of Mathematics,
Purdue University,
West Lafayette, IN 47907
e-mail: cmoyacal@purdue.edu

**Guang Lin**[1]
Department of Mathematics,
Purdue University,
West Lafayette, IN 47907;
Department of Mechanical Engineering,
Purdue University,
West Lafayette, IN 47907
e-mail: guanglin@purdue.edu

*Deep learning has significantly improved the state-of-the-art in computer vision and natural language processing, and holds great potential to design effective tools for predicting and simulating complex engineering systems. In particular, scientific machine learning seeks to apply the power of deep learning to scientific and engineering tasks, with operator learning (OL) emerging as a particularly effective tool. OL can approximate nonlinear operators arising in complex engineering systems, making it useful for simulating, designing, and controlling those systems. In this position paper, we provide a comprehensive overview of OL, including its potential applications to complex engineering domains. We cover three variations of OL approaches: deterministic OL for modeling nonautonomous systems, OL with uncertainty quantification (UQ) capabilities, and multifidelity OL. For each variation, we discuss drawbacks and potential applications to engineering, in addition to providing a detailed explanation. We also highlight how multifidelity OL approaches with UQ capabilities can be used to design, optimize, and control engineering systems. Finally, we outline some potential challenges for OL within the engineering domain.*
[DOI: 10.1115/1.4062635]

*Keywords: artificial intelligence, data-driven engineering, operator learning, machine learning for engineering applications*

## 1 Introduction

Powerful numerical simulators have allowed the scientific and engineering communities to accurately predict and simulate complex systems, such as power grids, mechanical engineering systems, and robotics. However, these simulators are computationally expensive, which makes them impractical for tasks that require many forward simulations, such as control, optimization design, and uncertainty quantification (UQ). Therefore, it is essential for the scientific and engineering communities to develop efficient alternatives that can accelerate the simulation of complex dynamical systems without compromising accuracy, while keeping the computational costs under control.

By providing efficient alternatives to numerical simulators, scientific machine learning (SciML) aims to leverage the power of deep learning to improve our ability to simulate and predict complex dynamical systems. Hence, a recent wave of SciML methods has demonstrated the potential of using observational data and physics to accelerate the numerical simulation of complex systems [1,2]. These methods can be categorized as deep learning based, with the goal of (i) approximating the governing equations of the system [3–5], (ii) incorporating the underlying physics into training protocols [6–8], or (iii) predicting the next state of the system [2,9,10].

For example, in their article [3], Brunton et al. used data to create an approximation of the unknown governing equations of a dynamical system. In Ref. [4], the same authors expanded their previous work to allow for an approximation of the input/state governing equations of a controlled dynamical system. In a similar endeavor, Schaeffer [5] used streams of data to learn a sparse approximation of partial differential equations (PDEs).

In the seminal paper [6], Raissi et al. introduced physics-informed neural networks, which are neural networks trained by incorporating the underlying mathematical equations of the system into the training loss function. PINNs [6,8] have been used to create faster approximations for scientific and engineering systems, such as power grids [7,11]. However, PINNs require retraining when dealing with operating conditions not included during training.

Other studies [2,9,10] have proposed training neural networks to predict the system's response for a given time horizon by approximating the system's next-step state based on the current state. For instance, Raissi et al. [10] used feed-forward neural networks, while Qin et al. [9] implemented residual networks for next-step prediction. However, these learning methods require vast amounts of data to avoid overfitting and error accumulation, and they may need to be retrained when dealing with unseen operating conditions. Therefore, the effective design of deep learning-based solutions for

[1]Corresponding author.

Manuscript received March 30, 2023; final manuscript received May 22, 2023; published online June 15, 2023. Assoc. Editor: Satyandra K. Gupta.

predicting and simulating large-scale complex dynamical systems remains a widely open research question.

*Operator Learning.* In their seminal paper [12], Lu et al. expanded on the universal approximation theorem for nonlinear operators [13] to introduce the deep operator network (DeepONet). DeepONet is capable of effectively learning nonlinear operators (i.e., mappings between infinite-dimensional spaces) with relatively small datasets. Compared to traditional neural networks, DeepONet exhibits minimal generalization errors, as evidenced by its performance in various applications, including control systems [14], power grids [15], and multiphysics problems [16]. Furthermore, other researchers have applied physics-informed training [17] and quantified uncertainty [15,18,19] to DeepONet.

As a parallel effort, Li et al. [20] proposed the Fourier neural operator (FNO), which approximates nonlinear operators by parameterizing integral kernels directly in Fourier space. The authors expanded on this work in Ref. [21] by proposing neural operators that can learn mappings between infinite-dimensional function spaces. Specifically, they formulated the neural operator as a composition of linear integral operators and nonlinear activation functions. The authors supported their work with a theoretical proof of a universal approximation theorem for the proposed neural operator, demonstrating its ability to approximate any given nonlinear continuous operator. It is worth noting that neural operators are designed for function spaces, allowing for various discretization methods and levels of resolution without requiring retraining. This makes neural operators superior to traditional deep learning methods when modeling complex engineering systems.

Thus, operator learning (OL) (which includes DeepONets and neural operators) has the potential to provide solutions to multiple problems in complex engineering systems. However, there are many challenges and open research problems that must be resolved before designing effective operator learning-based tools for engineering tasks. In this position paper, we provide a comprehensive overview of OL, including its potential applications to complex engineering domains. We cover three variations of OL approaches: deterministic OL for modeling nonautonomous systems, OL with UQ capabilities, and multifidelity OL. For each variation, we discuss drawbacks and potential applications to engineering, in addition to providing a detailed explanation. We also highlight how multifidelity OL approaches with UQ capabilities can be used to design, optimize, and control engineering systems. Lastly, we outline some potential challenges for OL within the engineering domain.

The rest of this position paper is organized as follows. First, in Sec. 2, we use nonautonomous dynamics to describe complex engineering systems and discuss deterministic operator learning techniques that can approximate the corresponding solution operator. Then, Sec. 3 explains how quantifying the epistemic uncertainty in operator learning can help provide robust predictions for engineering tasks. In Sec. 4, we provide an overview of how operator learning strategies can learn from different datasets and models. Additionally, we detail in this section some potential applications of Bayesian, multifidelity operator learning in engineering. Finally, Sec. 5 outlines the challenges that operator learning faces in the engineering domain, and Sec. 6 concludes the article.

## 2 Operator Learning and Engineering Systems

In this position paper, we model complex engineering systems using the following parametric, nonautonomous initial value problem:

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = f(x(t), u(t); \lambda), \quad t \in [0, T], \qquad x(0) = x_0 \tag{1}$$

Here, $x(t) \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ is the state vector, $u(t) \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ is the design control (or optimization) input, $\lambda \in \Lambda \subset \mathbb{R}^{n_\lambda}$ is the vector of parameters that describe a specific system, $[0, T] \subset [0, +\infty)$ is the finite time horizon of interest, and $x_0 \in \mathcal{X}$ is the initial condition. Note that we have chosen to use nonautonomous dynamics for engineering systems, as they (1) result from discretized PDEs in space, such as those representing robotic or mechanical systems, or (2) model microgrids interacting with a large-scale power grid. However, we emphasize that the OL techniques presented in this position paper can be applied to even more complex modeling choices, such as the differential algebraic equations [7] used to describe large-scale power grids.

**2.1 The Solution Operator.** One can describe the time response of an engineering system using the *solution operator*, which we denote as $G$. $G$ takes three inputs: the initial condition $x_0 \in \mathcal{X}$, the input as a continuous sequence, denoted as $u_{[0,t)} := \{u(s) \in \mathcal{U} : s \in [0, t)\}$, and the vector of parameters $\lambda$. At time $t \in [0, T]$, $G$ produces the state $x(t) \in \mathcal{X}$ according to the following equation:

$$G\big(x_0, u_{[0,t)}; \lambda\big)(t) \equiv x(t) = x_0 + \int_0^t f(x(s), u(s); \lambda)\,\mathrm{d}s \tag{2}$$

In this position paper, we discuss three different operator learning methods that can offer accurate and reliable approximations of the solution operator $G$.

**2.2 Approximating $G$ With Operator Learning.** The goal of operator learning is to approximate the solution operator $G$ using a parametric trainable representation/architecture $G_\theta$, i.e., $G(x_0, u_{[0,t)}, \lambda)(t) \approx G_\theta(x_0, u_{[0,t)}, \lambda)(t)$, for $t \in [0, T]$, where $\theta$ denotes all trainable parameters.

Let us focus on two of the most well-known architectures for operator learning: the DeepONet [12] and the FNO [20]. It is worth noting that any operator learning framework for complex engineering systems must meet three key criteria [22]: (i) the framework must be *discretization invariant*, which is important in situations such as power grids or robotics where input functions may have irregular sampling domains; (ii) the framework must be *prediction free*, given that the discretization of the input $u_{[0,t)}$ and the solution $G_\theta(\cdot, u_{[0,t)}, \cdot)(\cdot)$ may differ. This is particularly relevant in areas like material engineering, where the stress of the material may be predicted at different sampling times than the input load; (iii) the framework must be *domain independent*, recognizing that the inputs and the solution output may belong to different domains. This is a key consideration in power grids, where the parameter and state spaces may differ. When all three of these criteria are met, the operator learning framework is referred to as *mesh free* [22]. Table 1 presents an overview of the criteria that DeepONet and FNO satisfy.

As shown in Table 1, neither DeepONet nor FNO is suitable for approximating the solution operator of the parametric nonautonomous system, which is used to describe general complex engineering systems. To address this issue, in Refs. [14,23], the authors developed a resolution-independent version of DeepONet, i.e., discretization invariant, by using the sampling locations of the input function as inputs. Then, in Ref. [22], Zhang et al. extended DeepONet to design BelNet, the first *mesh-free* operator learning framework with theoretical guarantees. Note that since BelNet and the resolution-independent DeepONet are built upon the vanilla DeepONet, this article focuses mostly on DeepONet as the main operator learning framework for simplicity. However, it is worth mentioning that other neural operators can be used in specific applications for

**Table 1 Mesh-free criteria for DeepONet and FNO**

| | Disc.-invariant | Pred.-free | Domain-ind. |
|---|---|---|---|
| DeepONet | ✗ | ✓ | ✓ |
| FNO | ✓ | ✗ | ✗ |

engineering systems. For instance, in Ref. [24], the authors used FNO (which we briefly describe in Sec. 2.4) to build a surrogate model for predicting the transient response of power engineering systems.

**2.3 The Deterministic DeepONet.** To approximate the solution operator of an engineering system, one can use a trainable linear representation. In particular, the coefficients of the trainable linear representation will process the inputs $x_0$, $\tilde{u}$ (which denotes the discretized input), and $\lambda$, and the corresponding basis functions will process the prediction time $t$ within the output function domain $[0, T]$. Such a linear representation leads to a deterministic DeepONet $G_\theta$, with a vector of trainable parameters $\theta \in \mathbb{R}^p$. DeepONet consists of two neural networks: the Branch Net and the Trunk Net.

The *Branch* Net observes the initial state $x_0 \in \mathcal{X}$, the design input $\tilde{u}$, and the vector of parameters $\lambda$. Then, the *Branch* Net produces a vector of trainable coefficients $\beta \in \mathbb{R}^{qn_x}$. Similarly, the *Trunk* Net observes the prediction time $t \in (0, T]$ and produces a vector of trainable basis functions: $\varphi := (\varphi_1(t), \varphi_2(t), \ldots, \varphi_{qn_x}(t))^\top \in \mathbb{R}^{qn_x}$.

Finally, we calculate each component of the vector output $G_\theta(x_0, \tilde{u}, \lambda)(t) \equiv x(t)$ by employing the following dot product (for all $i = 1, \ldots, n_x$) between the vectors $(\beta_{(i-1)q}, \beta_{(i-1)q+1}, \ldots, \beta_{iq})^\top$ and $(\varphi_{(i-1)q}, \varphi_{(i-1)q+1}, \ldots, \varphi_{iq})^\top$:

$$G_\theta^{(i)} = \sum_{j=1}^{q} \beta_{(i-1)q+j}(x_0, \tilde{u}, \lambda) \cdot \varphi_{(i-1)q+j}(t) \tag{3}$$

We train the vector of DeepONet parameters $\theta \in \mathbb{R}^p$ by minimizing the loss function:

$$\mathcal{L}(\theta; \mathcal{D}_{\text{train}}) = \frac{1}{N_{\text{train}}} \sum_{k=1}^{N_{\text{train}}} \left\| G_k - G_\theta(x_{0,k}, \tilde{u}_k, \lambda_k)(t_k) \right\|_2^2 \tag{4}$$

using the dataset of $N_{\text{train}} := |\mathcal{D}_{\text{train}}|$ triplets: $\mathcal{D}_{\text{train}} := \left\{(x_{0,k}, \tilde{u}_k, \lambda_k), t_k, G_k\right\}_{k=1}^{N_{\text{train}}}$.

*2.3.1 Applications.* Deterministic OL can provide accurate predictions for engineering systems when extensive data are available for training OL networks, and the time domain of interest is small. For instance, we used the vanilla DeepONet proposed in Ref. [12] to learn the dynamic response of a pendulum for a given arbitrary input $u$. Figure 1 shows excellent agreement between the ground truth and the DeepONet prediction for the pendulum's angular velocity. Moreover, one could design efficient surrogate models for power engineering systems to predict state trajectories after the system experiences a significant disturbance, as described in Ref. [15]. Similarly, for materials engineering, deterministic OL could be used to forecast the response of viscoelastic materials to given arbitrary input load profiles.

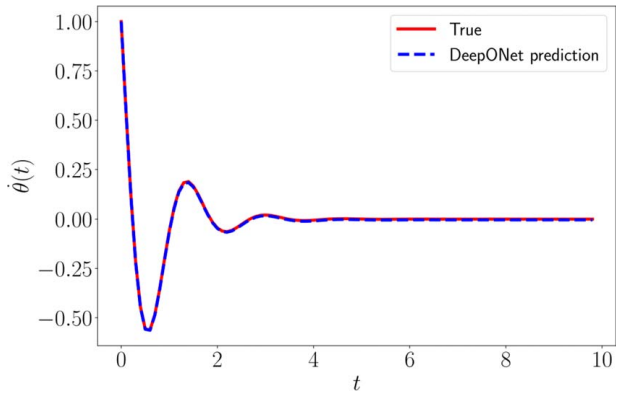


**Fig. 1 Comparison between the ground truth (top) and the DeepONet prediction (bottom) of the solution operator for the pendulum's angular velocity $\dot{\theta}(t)$ given an arbitrary control input**

*2.3.2 Drawbacks.* When dealing with small amounts of training data, which is a common situation in engineering systems, or when working with long time domains, it can be challenging to train deterministic OL networks. This limitation can have a negative impact on the predictive accuracy and reliability of OL, which could compromise its use in engineering tasks.

**2.4 Fourier Neural Operators.** Another framework for learning operators is the Fourier neural operator, which was introduced in Ref. [20]. FNO approximates the operator $G : u \mapsto G(u)$ using an iterative neural network architecture $v_0 \mapsto v_1, \ldots, v_T$, where $v_j$ for $j = 0, \ldots, T-1$ is a sequence of trainable functions. FNO begins by lifting the input $u$ to a higher-dimensional representation $v_0(t) = P(u(t))$, where $P$ is a lifting neural network. FNO then applies several iterations of updates $v_t \mapsto v_{t+1}$, which we define later in this article. Finally, FNO approximates the operator output as $G \approx Q(v_T(t))$, where $Q$ is a neural network-based projection.

Each iterative update is obtained using

$$v_{k+1}(t) := \sigma(W v_k(t) + (\mathcal{K}(u; \theta) v_k)(t)) \tag{5}$$

where $\sigma$ is an activation function, $W$ is a linear transformation that can be trained, and $\mathcal{K}$ is the kernel integral operator defined as follows:

$$(\mathcal{K}(u; \theta) v_k)(t) = \int \kappa_\theta(t - s) v_k(s) ds \tag{6}$$

Here, $\kappa_\theta$ is a neural network with trainable parameters $\theta$. FNO, in particular, approximates the above integral using:

$$(\mathcal{K}(\theta) v_k)(t) = \mathcal{F}^{-1}(R_\theta \cdot (\mathcal{F} v_k))(t) \tag{7}$$

where $R_\theta$ represents the Fourier transform of the periodic function $\kappa_\theta$, $\mathcal{F}$ denotes the Fourier transform, and $\mathcal{F}^{-1}$ represents its inverse.

Finally, to train the parameters of FNO, one uses the dataset $\{u_j, w_j\}_{j=1}^N$, where $u_j \sim \mu$ is an i.i.d. sequence obtained from the probability measure $\mu$ supported on $\mathcal{U}$, and $w_j = G(u_j)$ is the target or label, possibly corrupted with noise.

*2.4.1 Applications.* FNO can be used to learn parametric solution operators for complex PDEs. For example, in Ref. [20], the authors used FNO to approximate the operator that maps the vorticity described by the Navier–Stokes equation for a viscous, incompressible fluid in vorticity form within the time interval [0, 10] s to the vorticity up to some later time $T > 10$ s. Figure 2 (presented in the original article [20] and obtained with the original code published on Github) compares the FNO prediction with the ground truth. The results show excellent agreement between the true values and predictions. Thus, FNO is a potentially excellent tool for forecasting complex engineering systems, especially in the steady state.

*2.4.2 Drawbacks.* FNO is discretization invariant, meaning that the network is insensitive to different meshes. However, the input and output functions for FNO must be in the same domain and on the same grid. Consequently, FNO may not be the best choice for constructing parametric surrogate models of complex engineering systems that we intend to use for control and optimization purposes.

## 3 Operator Learning and Uncertainty Quantification

The conventional optimization framework used for training deterministic OL does not accurately quantify uncertainty, which is crucial for building estimators or credible intervals in critical applications involving engineering systems. This problem results

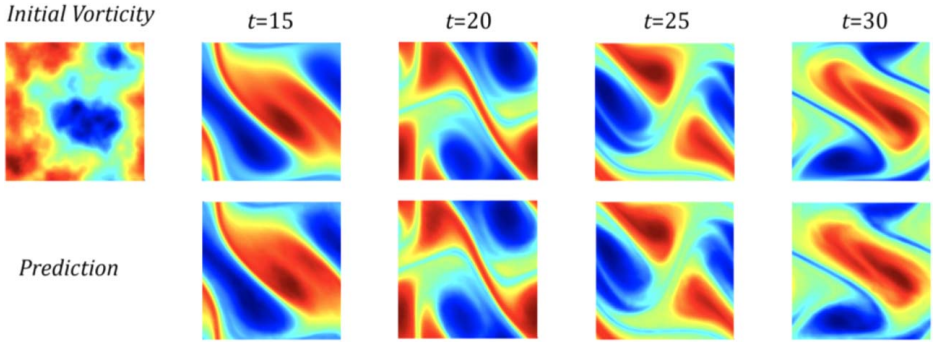


**Fig. 2 Comparison between the ground truth (top) and the FNO prediction (bottom) of the operator mapping the vorticity described by the Navier–Stokes equation for a viscous, incompressible fluid in vorticity form within the time interval [0, 10] *s*, to the vorticity up to some later time $T > 10$ *s***

in unreliable operator learning predictions and raises doubts about the credibility of operator learning for critical tasks. However, quantifying the uncertainty associated with noisy or limited data, as well as neural network overparameterization, is a challenging task. This challenge is even greater in operator learning, as it involves infinite-dimensional objects.

In Refs. [15,18], the authors developed the first Bayesian and UQ framework for OL. Specifically, to enable quantification of uncertainty for OL, the authors approximated the probability distribution $p(\tilde{G}|x_0, \tilde{u}, \lambda, t, \mathcal{D}_{\text{train}})$. Let us assume that the OL outputs are scalars and noisy: $\tilde{G} = G_d + \epsilon_u$. Thus, the output represents the response of the $i$th state component for an engineering system, $x^{(i)} = G^{(i)}$ for $i = 1, \ldots, n_x$. Note that extending the following analysis to the vector case requires using a multivariate distribution to represent the posterior. In the aforementioned equation, $G_d$ captures the deterministic portion of the OL's output, and $\epsilon_u$ represents the data uncertainty or noise model that we assume we know. It is worth noting that in practice, for engineering systems, one can estimate this noise model from data. We also define the likelihood model for the noise, e.g., a factorized Gaussian.

The value of $\tilde{G}$ for a given input and dataset is a random variable, which one can denote as $(\tilde{G}|x_0, \tilde{u}, \lambda, t, \mathcal{D}_{\text{train}})$. To obtain the density of this random variable, one must integrate out the trainable parameters $\theta$, i.e.,

$$p(\tilde{G}|x_0, \tilde{u}, \lambda, t, \mathcal{D}_{\text{train}}) = \int p(\tilde{G}|x_0, \tilde{u}, \lambda, t, \theta) p(\theta|\mathcal{D}_{\text{train}})\, \mathrm{d}\theta \tag{8}$$

The aforementioned equation requires knowledge of the posterior density $p(\theta|\mathcal{D}_{\text{train}})$. Knowing the posterior density enables quantifying the uncertainty of the OL parameters, also known as *epistemic uncertainty*. The posterior density can be obtained via the Bayes rule (see Fig. 3 for a pictorial representation):

$$P(\theta|\mathcal{D}_{\text{train}}) = \frac{P(\mathcal{D}_{\text{train}}|\theta)P(\theta)}{P(\mathcal{D}_{\text{train}})} \propto P(\mathcal{D}_{\text{train}}|\theta)P(\theta) \tag{9}$$

Here, the likelihood of the data is

$$P(\mathcal{D}_{\text{train}}|\theta) = \prod_{i=1}^{N_{\text{train}}} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left( -\frac{(G_\theta(x_{0,i}, \tilde{u}_i, \lambda_i)(t_i) - \tilde{G}_i)^2}{2\sigma^2} \right) \tag{10}$$

which we calculate using the OL forward pass and i.i.d. noisy training data, as illustrated in Fig. 3, and $P(\theta)$ is the prior of the parameters.

Acquiring the posterior density from Bayes' rule is often computationally and analytically intractable. Thus, in Ref. [18], the authors approximated the posterior distribution $P(\theta|\mathcal{D}_{\text{train}})$ using samples obtained from it. Specifically, an ensemble of $\theta$ samples, denoted as $\{\theta_k\}_{k=1}^M$, was obtained.

Subsequently, by using the ensemble, we can fit a Gaussian distribution $\mathcal{N}(\bar{\mu}, \bar{\sigma}^2)$ to the ensemble of posterior samples and use it for evaluations during control and optimization design. The mean of the fitted Gaussian is given by:

$$\mathbb{E}[\tilde{G}|x_0, \tilde{u}, \lambda, t, \mathcal{D}_{\text{train}}] = \bar{\mu} = \frac{1}{M} \sum_{j=1}^{M} G_{\hat{\theta}_j}(x_0, \tilde{u}, \lambda)(t) \tag{11}$$

where $\{G_{\hat{\theta}_j}(x_0, \tilde{u}, \lambda)(t)\}_{j=1}^M$ is the set of OL predictions, corresponding to the ensemble samples $\{\hat{\theta}_j\}_{j=1}^M$. We also quantified the *epistemic uncertainty* of the predictions by computing the variance:

$$\mathrm{Var}(\tilde{G}|x_0, \tilde{u}, \lambda, t, \mathcal{D}_{\text{train}}) \approx \bar{\sigma}^2 = \frac{1}{M} \sum_{j}^{M} (G_{\hat{\theta}_j}(x_0, \tilde{u}, \lambda)(t) - \bar{\mu})^2 \tag{12}$$

**3.1 Applications.** OL and UQ can be used to estimate credible sets and provide reliable predictions for safety-critical engineering systems, even when there are limited data available for training OL networks. For instance, one could develop robust surrogate models for power engineering systems that estimate credible sets capable of capturing the state trajectories following significant system disturbances, as described in Ref. [15] and illustrated in

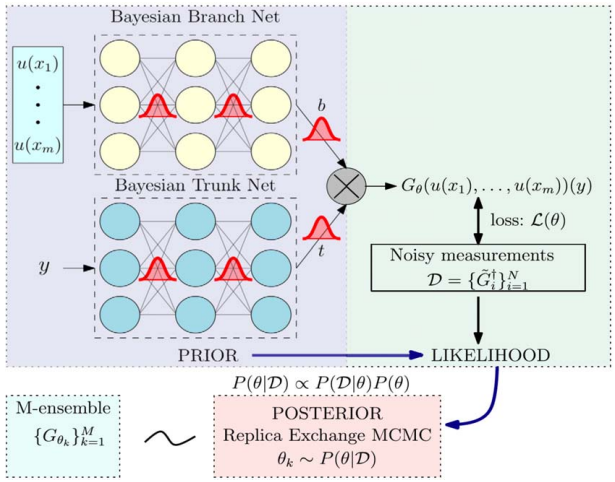


**Fig. 3 The Bayesian DeepONet framework. The Bayesian DeepONet represents the prior of the trainable parameters $\theta$. One computes the likelihood using noisy measurements $\mathcal{D} = \{\tilde{G}_i^\dagger\}_{i=1}^N$. Markov chain Monte Carlo enables sampling from the posterior $\theta_k \sim P(\theta|D)$. The sampled parameters are used to build the DeepONet *M*-ensemble $\{G_{\theta_k}\}_{k=1}^M$ for prediction and uncertainty quantification.**

Fig. 4. In this example, we learn the OL model that takes as input $u$ a discrete representation of the state variable $x$ (i.e., the voltage) within the time interval [0, 2] s, which contains the disturbance event. The OL model then predicts the state trajectory (i.e., the voltage trajectory) and a credible or 95% confidence interval within the future time interval [2, 7] s. In addition, OL and UQ can provide an effective method to estimate the number of false alarms due to violations of constraints in engineering systems.

**3.2 Drawbacks.** One should note that OL and UQ require employing the $M$-ensemble of parameters sampled from the posterior distribution. However, performing inference with so many parameters for large-scale engineering systems might become computationally expensive, which may preclude the application of OL and UQ to online tasks. Although a parallel computing framework has been effectively used for certain complex engineering tasks, such as robot motion simulation [25,26], fast motion planning [27], and multibody system dynamics [28], it can only partially reduce computational costs. Therefore, we believe that more efficient solutions, combined with parallel computing strategies, are still necessary to accurately quantify the uncertainty of complex engineering systems. In this regard, we can leverage the work that enables robot planning in the presence of uncertainties [29].

## 4 Multifidelity Operator Learning

The flexibility of operator learning is balanced by the requirement for substantial amounts of high-fidelity data. In complex engineering systems, it is often not feasible to collect large amounts of high-fidelity data due to its high cost. However, these same systems may have access to an abundance of low-fidelity or noisy data, along with knowledge of physical models of various fidelities or orders. Therefore, to apply OL to engineering systems, a combination of multifidelity data and models must be used during training and inference.

The aim of multifidelity OL is to capture the behavior of engineering systems as comprehensively as possible by utilizing a combination of multifidelity data and models. However, this approach raises several questions that need to be addressed. For example, how can multifidelity OL integrate data with the underlying physics of engineering systems?

**4.1 Fusing Data and Physics for Multifidelity Operator Learning.** In Ref. [23], the authors created a multifidelity OL framework that can combine both physics and data. Specifically, the authors employed an OL network to estimate the residual operator, which characterizes the error or residual dynamics between the true solution operator of the system $G$ and the solution operator derived from integrating the system's low-fidelity mathematical model, e.g., a linearized model.

Mathematically, we assume we know a previously derived mathematical model of the system:

$$\begin{aligned} &\frac{\mathrm{d}}{\mathrm{d}t}\tilde{x}(t) = f_{\text{approx}}(\tilde{x}(t), \tilde{u}; \lambda), \quad t \in [0, T], \\ &\tilde{x}(0) = x_0 \end{aligned} \tag{13}$$

In the aforementioned equation, $f_{\text{approx}} : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the known vector field that approximates the true vector field $f$. The corresponding solution of Eq. (13) is expressed as follows:

$$\tilde{G}_{\text{approx}} \equiv \tilde{x}_0 + \int_0^t f_{\text{approx}}(\tilde{x}(s), \tilde{u}; \lambda)\,\mathrm{d}s, \quad t \in (0, T] \tag{14}$$

$\tilde{G}_{\text{approx}}$ can be, for example, (i) a trajectory solution of an integration scheme, e.g., Runge–Kutta [30] of a linearized model, (ii) a physics-informed DeepONet [17] trained to satisfy Eq. (13), or (iii) a DeepONet (see § 2.3) trained using a dataset $\mathcal{D}$ generated by simulating Eq. (13).

One can obtain the residual OL as follows:

$$\begin{aligned} G &\equiv \tilde{x}(t) \\ &= \tilde{x}_0 + \int_0^t f(\tilde{x}(s), \tilde{u}, \lambda)\,\mathrm{d}s \\ &= \tilde{G}_{\text{approx}}(\tilde{x}_0, \tilde{u}, \lambda)(t) + ''\text{residual dynamics}'' \end{aligned} \tag{15}$$

The *residual operator* $G_\epsilon$ is then:

$$G_\epsilon := [\tilde{G} - \tilde{G}_{\text{approx}}](x_0, \tilde{u}, \lambda)(t), \quad t \in (0, T] \tag{16}$$

To approximate the residual operator $G_\epsilon$, we designed a residual OL network (residual DeepONet) $G^\epsilon_\theta$ with trainable parameters $\theta \in \mathbb{R}^p$. $G^\epsilon_\theta$ has the same architecture and is trained in the same way as the DeepONet described earlier. As a simple example of the effectiveness of fusing data and physics, we learn the residual operator of a nonlinear pendulum with known linear dynamics. Figure 5 depicts an oscillatory trajectory obtained using the residual OL network.

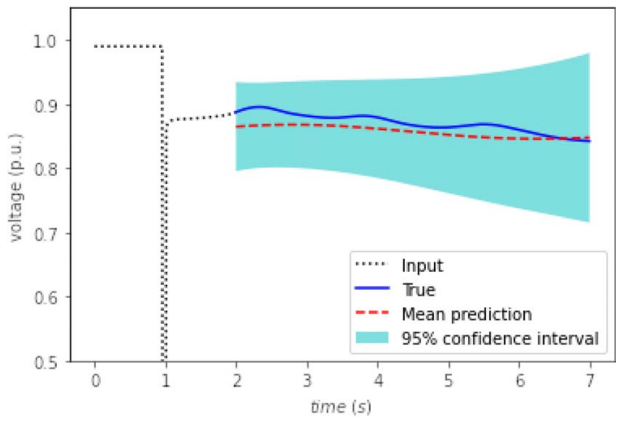


**Fig. 4 OL and UQ prediction and confidence interval for a post-disturbance state trajectory. Specifically, one learns the OL model that takes as input $u$ a discrete representation of the state variable $x$ (i.e., the voltage) within the time interval [0, 2] s, which contains the disturbance event. The OL model then predicts the state trajectory (i.e., the voltage trajectory) and a credible or 95% confidence interval within the future time interval [2, 7] s.**

**4.2 Applications.** These findings indicate a promising path for multifidelity OL by leveraging the fusion of data and physics information with varying degrees of accuracy. This scenario is common in high-impact engineering fields, such as materials engineering and mechanical systems, where physical models may exist, but generating a large volume of precise data from numerical or physical experiments can be prohibitively expensive.

**4.3 Drawbacks.** While incorporating physical model information can guide the optimization landscape for OL training by serving as a prior, it can also slow down the optimization process. Therefore, it is crucial to examine the potential downsides of merging data and physics for complex, large-scale engineering systems in OL. On the other hand, using data of varying levels of accuracy necessitates treating each level separately to prevent high-fidelity data from being diluted by low-fidelity data. Thus, the design of neural network architectures and training protocols to identify correlations between the predictions made by a high-fidelity model (e.g., a full-order model) and a low-fidelity model (e.g., a reduced-order model) remains an open research question. Specifically, the goal is to understand how the low-fidelity model can accurately predict the behavior of the high-fidelity model, and how the two models can be combined to produce more accurate

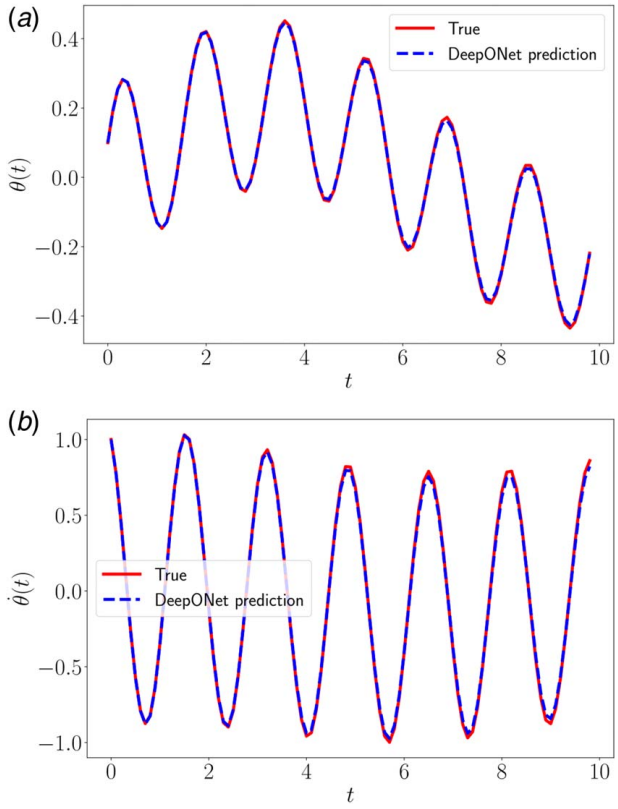


**Fig. 5 Comparison of residual DeepONet prediction with the actual trajectories of the pendulum system's state $x = (\theta(t), \dot{\theta}(t))^T$ ((*a*) is angle $\theta(t)$ and (*b*) is velocity $\dot{\theta}(t)$) response to an oscillatory control input $u(t)$**

OL output. This is an active area of research in the fields of operator learning, uncertainty quantification, and surrogate modeling.

**4.4 Other Potential Applications of Operator Learning.** Designing effective and robust data-driven control and optimization strategies for engineering dynamical systems is still a challenging task when using deep learning-based solutions. Two major challenges are as follows: (i) data distributional drift, which renders traditional supervised learning strategies ineffective; and (ii) adversarial designs, which are risky designs that could jeopardize the operation of the engineering systems. The multifidelity Bayesian OL strategies described in this position paper have the potential to provide effective and efficient solutions to such challenges. This is because the OL and UQ framework can handle distributional drift and prevent adversarial designs. Thus, the methods discussed in this position paper may solve control and optimization problems of the form:

$$\max \int_0^t J(\hat{x}(s), u(s))\, \mathrm{d}s \tag{17}$$

where $\hat{x}(t)$ is the forecast generated by a Bayesian, multifidelity OL-based surrogate model for an engineering system.

## 5 Challenges

While multiple researchers have proposed operator learning frameworks for scientific and engineering applications, several challenges must be addressed before operator learning can be effectively applied to complex engineering systems. In this section, we will discuss these main challenges and propose possible solutions and open problems centered on multifidelity operator learning with uncertainty quantification capabilities.

**5.1 Scalability.** Large-scale systems, such as power grids, fluids, and materials engineering, are common in the field of engineering. Therefore, it is crucial to ensure that OL techniques can scale with the size of the system. However, the original universal approximation theorem for nonlinear operators only covers the single-input single-output case, leaving us with a lack of theoretical support for OL when dealing with large-scale systems. A few studies, such as Jin et al. [31] and Sun et al. [32], have attempted to develop strategies to alleviate the scalability problem. For example, in Ref. [31], the authors used tensor products to extend the original OL framework presented in the study by Lu et al. [12] to the multi-input single-output setting, providing extensive theoretical support. However, since it has only one output, it cannot provide the theoretical support for most engineering systems.

On the other hand, in Ref. [32], the authors combined OL with graph neural networks to tackle forecasting and prognosis problems for large-scale networked systems, such as power and traffic engineering systems. The empirical results show that by exploiting the underlying graph structure of the system, OL learning can effectively predict the infinite-dimensional response of such systems. However, this work lacks the necessary theoretical support for scalable multi-input multi-output OL. Therefore, it is worthwhile to study such a problem so that OL can become an effective tool for engineering systems.

**5.2 Long-Term Predictions.** OL provides an effective method for predicting the solution operator of time-dependent engineering problems. However, as the time horizon increases, training OL frameworks becomes increasingly challenging, which can affect prediction accuracy due to the accumulation of errors. Moreover, traditional supervised training may require an excessive amount of data, making it prohibitive. Thus, developing an effective OL method that can approximate the dynamic response of nonautonomous systems for long-term horizons is a crucial challenge in OL for engineering systems.

Attempts have been made to find solutions to the problem of long-term prediction. For instance, in Ref. [33], the authors combined DeepONets with recurrent neural networks to demonstrate that their method can handle the long-term prediction of time-dependent PDEs. However, this approach lacks theoretical support, and it is unclear whether it will work for nonautonomous systems. In Refs. [14,23], the authors proposed DeepONets for the long-term prediction of nonautonomous systems [14] and power grid devices that interact with simulators [23]. In these works, the error accumulation of long-term prediction was studied, reinforcement learning-inspired training techniques were proposed when data are scarce, and a multifidelity framework was enabled. However, theoretical evidence for the scalability and robustness of the proposed approach is still lacking. Therefore, more ideas are needed to tackle the long-term prediction of engineering systems. One possibility is to fuse UQ scalable models with multifidelity OL, which may lead to significant results.

Finally, it is worth noting that FNO may provide inaccurate predictions as the size of the time horizon increases. Additionally, the architecture of FNO only allows for the use of recurrent solutions with fixed time-steps. This limitation could hinder the use of FNO for predicting stiff engineering systems, such as power grids.

**5.3 Privacy.** Developing effective OL frameworks for complex engineering systems poses important challenges related to privacy and data sharing. For instance, power grid utilities in the electrical market may be unwilling to transmit their data to a centralized location for training due to concerns about revealing information to competitors or cybersecurity risks. To address this issue, federated learning is an emerging technique that allows for collaborative training of a shared model while keeping the data decentralized. This technique provides potential privacy protection for engineering users.

Moya and Lin studied the federated training of DeepONets [34]. The findings of this article suggest that federated learning can be an effective way to train models that approximate nonlinear operators using decentralized and heterogeneous data. Currently, there are no known privacy-protecting OL methods that have been successfully applied to complex engineering systems. Therefore, federated and distributed training of multifidelity, Bayesian, OL models could potentially be used to design secure and effective tools for these systems.

Finally, it should be noted that the current versions of DeepONet and FNO require the training data to be transferred to a centralized location. As a result, neither DeepONet nor FNO can protect the data of engineering users.

## 6 Conclusion

This position paper provides a comprehensive overview of operator learning and its potential applications in complex engineering tasks. We discuss the three main criteria for mesh-free operator learning frameworks, which can be a powerful tool for developing flexible solutions to complex engineering problems. The three main criteria are as follows:

(1) The OL framework must be *discretization invariant*, which is important in situations where input functions may have irregular sampling domains.
(2) The OL framework must be *prediction free*, particularly relevant in areas that require predictions at different sampling times.
(3) The OL framework must be *domain independent*, recognizing that the input functions and the solution output function may belong to different domains.

It is worth noting that the two most well-known operator learning frameworks described in this position paper, DeepONet and FNO, are not mesh free. Developing mesh-free operator learning frameworks that can handle multiple inputs and outputs, are theoretically sound, and are scalable remains an open research problem.

We also discussed three variations of OL frameworks: deterministic OL, which models nonautonomous systems; OL with uncertainty quantification capabilities; and multifidelity OL. Each variation can be built upon mesh-free and nonmesh-free OL frameworks, as demonstrated in the Bayesian and multifidelity DeepONet frameworks proposed in the literature and listed in this position paper.

For each variation, we explored drawbacks and potential applications to engineering systems and provided a detailed explanation. We also described how multifidelity OL approaches with UQ capabilities can be leveraged to design, optimize, and control engineering systems. Finally, we outlined potential challenges for OL within the engineering domain.

In conclusion, the development of mesh-free, scalable, multifidelity, and Bayesian operator learning frameworks remains one of the main goals in operator learning research. These frameworks have the potential to provide transformative solutions to various engineering domains, such as power engineering, materials engineering, robotics, and fluid mechanics.

## Acknowledgment

The authors gratefully acknowledge the support of the National Science Foundation (DMS-1555072, DMS-2053746, and DMS-2134209), Brookhaven National Laboratory Subcontract 382247, and U.S. Department of Energy (DOE) Office of Science Advanced Scientific Computing Research program DE-SC0021142 and DE-SC0023161.

## Conflict of Interest

There are no conflicts of interest.

## Data Availability Statement

The datasets generated and supporting the findings of this article are obtainable from the corresponding author upon reasonable request.

## References

[1] Efendiev, Y., Leung, W. T., Lin, G., and Zhang, Z., 2022, "Efficient Hybrid Explicit-Implicit Learning for Multiscale Problems," J. Comput. Phys., **467**, p. 111326.
[2] Qin, T., Chen, Z., Jakeman, J. D., and Xiu, D., 2021, "Data-Driven Learning of Nonautonomous Systems," SIAM J. Sci. Comput., **43**(3), pp. A1607–A1624.
[3] Brunton, S. L., Proctor, J. L., and Kutz, J. N., 2016, "Discovering Governing Equations From Data by Sparse Identification of Nonlinear Dynamical Systems," Proc. Natl. Acad. Sci. U. S. A., **113**(15), pp. 3932–3937.
[4] Brunton, S. L., Proctor, J. L., and Kutz, J. N., 2016, "Sparse Identification of Nonlinear Dynamics With Control (sindyc)," IFAC-PapersOnLine, **49**(18), pp. 710–715.
[5] Schaeffer, H., 2017, "Learning Partial Differential Equations Via Data Discovery and Sparse Optimization," Proc. R. Soc. A: Math., Phys. Eng. Sci., **473**(2197), p. 20160446.
[6] Raissi, M., Perdikaris, P., and Karniadakis, G. E., 2019, "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations," J. Comput. Phys., **378**, pp. 686–707.
[7] Moya, C., and Lin, G., 2023, "DAE-PINN: A Physics-Informed Neural Network Model for Simulating Differential Algebraic Equations With Application to Power Networks," Neural Computi. Appl., **35**(5), pp. 1–16.
[8] Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L., 2021, "Physics-Informed Machine Learning," Nat. Rev. Phys., **3**(6), pp. 422–440.
[9] Qin, T., Wu, K., and Xiu, D., 2019, "Data Driven Governing Equations Approximation Using Deep Neural Networks," J. Comput. Phys., **395**, pp. 620–635.
[10] Raissi, M., Perdikaris, P., and Karniadakis, G. E., 2018, "Multistep Neural Networks for Data-Driven Discovery of Nonlinear Dynamical Systems," arXiv preprint arXiv:1801.01236.
[11] Misyris, G. S., Venzke, A., and Chatzivasileiadis, S., 2020, "Physics-Informed Neural Networks for Power Systems," 2020 IEEE Power & Energy Society General Meeting (PESGM), Montreal, QC, Canada, Aug. 2–6, IEEE, pp. 1–5.
[12] Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E., 2021, "Learning Nonlinear Operators Via Deeponet Based on the Universal Approximation Theorem of Operators," Nat. Mach. Intell., **3**(3), pp. 218–229.
[13] Chen, T., and Chen, H., 1995, "Universal Approximation to Nonlinear Operators by Neural Networks With Arbitrary Activation Functions and Its Application to Dynamical Systems," IEEE Trans. Neural Netw., **6**(4), pp. 911–917.
[14] Li, G., Moya, C., and Zhang, Z., 2022, "On Learning the Dynamical Response of Nonlinear Control Systems With Deep Operator Networks," arXiv preprint arXiv:2206.06536.
[15] Moya, C., Zhang, S., Yue, M., and Lin, G., 2023, "Deeponet-Grid-UQ: A Trustworthy Deep Operator Framework for Predicting the Power Grid's Post-Fault Trajectories," Neurocomputing, **535**, pp. 166–182.
[16] Cai, S., Wang, Z., Lu, L., Zaki, T. A., and Karniadakis, G. E., 2021, "Deepm&Mnet: Inferring the Electroconvection Multiphysics Fields Based on Operator Approximation by Neural Networks," J. Comput. Phys., **436**, p. 110296.
[17] Wang, S., and Perdikaris, P., 2021, "Long-Time Integration of Parametric Evolution Equations With Physics-Informed Deeponets," J. Comput. Phys., **475**, p. 111855.
[18] Lin, G., Moya, C., and Zhang, Z., 2021, "Accelerated Replica Exchange Stochastic Gradient Langevin Diffusion Enhanced Bayesian Deeponet for Solving Noisy Parametric PEDS," arXiv preprint arXiv:2111.02484.
[19] Yang, Y., Kissas, G., and Perdikaris, P., 2022, "Scalable Uncertainty Quantification for Deep Operator Networks Using Randomized Priors," arXiv preprint arXiv:2203.03048.
[20] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A., 2020, "Fourier Neural Operator for Parametric Partial Differential Equations," arXiv preprint arXiv:2010.08895.
[21] Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A., 2021, "Neural Operator: Learning Maps Between Function Spaces," arXiv preprint arXiv:2108.08481.
[22] Zhang, Z., Leung, W. T., and Schaeffer, H., 2022, "Belnet: Basis Enhanced Learning, a Mesh-Free Neural Operator," arXiv preprint arXiv:2212.07336.
[23] Moya, C., Lin, G., Zhao, T., and Yue, M., 2023, "On Approximating the Dynamic Response of Synchronous Generators via Operator Learning: A Step Towards Building Deep Operator-Based Power Grid Simulators," arXiv preprint arXiv:2301.12538.
[24] Cui, W., Yang, W., and Zhang, B., 2023, "A Frequency Domain Approach to Predict Power System Transients," IEEE Trans. Power Syst., pp. 1–13.

[25] Kasahara, H., Fujii, H., and Iwata, M., 1987, "Parallel Processing of Robot Motion Simulation," IFAC Proceedings Vol., **20**(5), pp. 329–336.
[26] Choi, H., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., Hearl, F., Hodgins, J., Jain, A., Leve, F., and Li, C., 2021, "On the Use of Simulation in Robotics: Opportunities, Challenges, and Suggestions for Moving Forward," Proc. Natl. Acad. Sci. U. S. A., **118**(1), p. e1907856118.
[27] Henrich, D., 1997, "Fast Motion Planning by Parallel Processing—A Review," J. Intell. Rob. Syst., **20**(1), pp. 45–69.
[28] Negrut, D., Serban, R., Mazhar, H., and Heyn, T., 2014, "Parallel Computing in Multibody System Dynamics: Why, When, and How," J. Comput. Nonlinear. Dyn., **9**(4), p. 041007.
[29] Coelho, P., and Nunes, U., 2005, "Path-Following Control of Mobile Robots in Presence of Uncertainties," IEEE Trans. Rob., **21**(2), pp. 252–261.
[30] Iserles, A., 2009, *A First Course in the Numerical Analysis of Differential Equations*, Vol. 44, Cambridge University Press, Cambridge, UK.
[31] Jin, P., Meng, S., and Lu, L., 2022, "Mionet: Learning Multiple-Input Operators Via Tensor Product," SIAM J. Sci. Comput., **44**(6), pp. A3490–A3514.
[32] Sun, Y., Moya, C., Lin, G., and Yue, M., 2022, "Deepgraphonet: A Deep Graph Operator Network to Learn and Zero-Shot Transfer the Dynamic Response of Networked Systems," arXiv preprint arXiv:2209.10622.
[33] Michałowska, K., Goswami, S., Karniadakis, G. E., and Riemer-Sørensen, S., 2023, "Neural Operator Learning for Long-Time Integration in Dynamical Systems With Recurrent Neural Networks," arXiv preprint arXiv:2303.02243.
[34] Moya, C., and Lin, G., 2022, "Fed-deeponet: Stochastic Gradient-Based Federated Training of Deep Operator Networks," Algorithms, **15**(9), p. 325.