Bi-fidelity modeling of uncertain and partially unknown systems using DeepONets

Subhayan De 1,2 · Matthew Reynolds 3 · Malik Hassanaly 3 · Ryan N. King 3 · Alireza Doostan 2

Received: 14 August 2022 / Accepted: 13 January 2023 / Published online: 25 March 2023
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2023

Abstract
Recent advances in modeling large-scale, complex physical systems have shifted research focuses towards data-driven techniques. However, generating datasets by simulating complex systems can require significant computational resources. Similarly, acquiring experimental datasets can prove difficult. For these systems, often computationally inexpensive, but in general inaccurate models, known as the low-fidelity models, are available. In this paper, we propose a bi-fidelity modeling approach for complex physical systems, where we model the discrepancy between the true system’s response and a low-fidelity response in the presence of a small training dataset from the true system’s response using a deep operator network, a neural network architecture suitable for approximating nonlinear operators. We apply the approach to systems that have parametric uncertainty and are partially unknown. Three numerical examples are used to show the efficacy of the proposed approach to model uncertain and partially unknown physical systems.

Keywords Bi-fidelity method · Uncertain system · Neural network · Deep operator network · Uncertainty quantification

1 Introduction
The ubiquitous presence of uncertainty often affects the modeling of a physical system using differential equations. For example, in engineering systems, the sources of uncertainty can include material properties, geometry, or loading conditions. This type of uncertainty is known as parametric uncertainty and differs from one realization to another. In contrast, during modeling, some physical phenomena are often ignored or simplified, which leads to the so called structural uncertainty [28]. Both types of uncertainties can be present in a real-life engineering system, and should be addressed when developing digital twins of the system. Models developed using polynomial chaos expansion [22,62], Gaussian process regression [20,60], and other response surfaces [9,23,25] address parametric uncertainty, which defines possible realizations of the system. However, the cost of developing these models increases significantly with the dimension of the uncertain variables [17]. In this paper, we make a distinction between these two types of uncertainties to show the broad applicability of the proposed method here.

Neural networks, commonly used in computer vision [57] and pattern recognition problems [1], have been used to model physical systems in the presence of uncertainty [11,55]. To model physical systems, Raissi et al. [51] augmented the standard data discrepancy loss with the residual of the corresponding governing equations. This approach was termed “physics-informed neural networks.” Subsequently, a significant amount of work employing this approach to address basic computational mechanics problems came into prominence in the literature, some notable ones being Karniadakis et al. [27], Cai et al. [6], and Viana and Subramaniam
[56]. Among these, several works address the parametric uncertainty in the system [21,61,63–65,68]. However, machine learning studies that address structural uncertainty in modeling are scarce. Among them, Blakseth et al. [2] used neural networks combined with governing equations to address modeling and discretization errors. Zhang et al. [67] used the physics-informed approach to quantify both the parametric and approximation uncertainty of the neural network. Duraisamy and his colleagues further trained neural networks to comprehend the discrepancies between turbulence model prediction and the available data [18,47,54]. Recently, several neural network architectures capable of approximating mesh-independent solution operators for the governing equations of physical systems have been proposed [35–39,42,44], drawing inspiration from the universal approximation theorem for operators in Chen and Chen [8]. These architectures overcome two shortcomings of the physics-informed use of neural networks to learn a system’s behavior: firstly, they do not depend on the mesh used to generate the training dataset. Secondly, they are solely data-driven and do not require the knowledge of the governing equations. Among these network architectures, the deep operator network (DeepONet) proposed in Lu et al. [42,44] uses two separate networks—one with the external source term as input and the other with the coordinates where the system response is sought as input. The former network is known as the branch network while the latter is known as the trunk network. Outputs from these two networks are then used in a linear regression model to produce the system response. Lu et al. [43] showed how the computational domain can be decomposed into many small subdomains to learn the governing equation with only one solution of the equation. While Cai et al. [7] used DeepONets for a multiphysics problem involving the modeling of field variables across multiple scales, Ranade et al. [52] and Sharma Priyadarshini et al. [53] employed DeepONets for approximating joint subgrid probability density function in turbulent combustion problems and to model non-equilibrium flow physics, respectively. Additionally, Wang et al. [59] and Wang and Perdikaris [58] used the error in satisfying the governing equations to train the DeepONets, with a variational formulation of the governing equations being applied by Goswami et al. [24] to train DeepONets to model brittle fracture. Further comparisons of DeepONet with the iterative neural operator architecture proposed in Li et al. [35] were performed in Kovachki et al. [33] and Lu et al. [45]. The approximation errors in the latter neural operator architectures were subsequently discussed in Kovachki et al. [32], Lanthaler et al. [34], Deng et al. [16], and Marcati and Schwab [46].

The fidelity of a training dataset represents the degree to which the dataset describes the behavior of the true system [40]. However, high-fidelity training datasets, a subjective measure, for a complex physical system can often be limited due to the computational burden associated with simulating such systems. Similarly, repeating experiments to obtain measurement data can become infeasible. On the other hand, a similar but computationally inexpensive model, known as low-fidelity, that captures many aspects of the true system’s behavior can be used to generate larger low-fidelity training datasets. In this paper, we propose a modeling strategy with DeepONets combining both high- and low-fidelity training datasets. In particular, the discrepancy between the response from the true physical system and the low-fidelity model is estimated using DeepONets. The assumption used here is that the modeling of the discrepancy requires only a small training dataset of the true system’s response, and as a result, bi-fidelity approaches are often advantageous with scarce training data from the high-fidelity simulations [12,13,15]. In principle, we train a DeepONet that uses the low-fidelity response as the input of the branch network to predict the discrepancy between the low- and high-fidelity data. However, we do not incorporate physics loss or error in satisfying the governing equations (as previously employed in physics informed neural networks) to circumvent potential bias incorporation in partially known engineering systems subject to the presence of structural uncertainty. We also demonstrate the effectiveness of the proposed approach using formal results from Lanthaler et al. [34] and using an elementary example.

In this paper, we employ three numerical examples to show the efficacy of the proposed bi-fidelity approach to modeling the response of an uncertain and partially unknown physical system. The first two examples consider a nonlinear oscillator and a heat transfer problem, respectively, where uncertainty is present in some of the parameters of the governing equations. The low-fidelity model used in these examples ignores the nonlinearity or some of the uncertain components of the governing equations. In the third example, we model power generated in a wind farm, with the two and three dimensional simulations respectively serving as high and low-fidelity model systems.

The rest of the paper is organized as follows: first, we provide a brief background on DeepONets in Sect. 2. Then, we present our proposed approach for modeling the discrepancy between a low-fidelity system response and the true response for uncertain and partially unknown systems in Sect. 3. Three numerical examples involving a dynamical system, a heat transfer problem, and a wind farm application are subsequently used in Sect. 4 to showcase the efficacy of the proposed approach. Finally, we conclude the paper with a brief discussion on future applications of the proposed approach.
2 Background on DeepONets

Let us assume the governing equation of a physical system in a domain $\Omega$ is given by

$$\mathcal{L}(u)(x) = f(x), \quad x \in \Omega,$$

(1)

where $\mathcal{L}$ is a nonlinear differential operator; $x$ is the location; $u$ is the response of the system; and $f$ is the external source. For a boundary value problem, Dirichlet boundary condition $u(x) = u_D(x)$ on $x \in \partial\Omega_D$ and/or Neumann boundary condition $\mathcal{L}_N(u)(x) = u_N(x)$ on $x \in \partial\Omega_N$ are also present, where $\mathcal{L}_N$ is the differential operator for the Neumann boundary condition; $\partial\Omega_D \cup \partial\Omega_N = \partial\Omega$ with the boundary of the domain being $\partial\Omega$; and $\partial\Omega_D \cap \partial\Omega_N = \emptyset$. For an initial value problem, an initial condition $u(x_0) = u_0$ is used instead, where $x_0$ is the initial instant. The solution of (1) is given by $G(f)(x)$, where $G : \mathcal{X} \rightarrow \mathcal{Y}$ denotes the solution operator. For $\mathcal{X}$ and $\mathcal{Y}$, there exist embeddings $\mathcal{X} \hookrightarrow L^2(D)$ and $\mathcal{Y} \hookrightarrow L^2(\Omega)$.

The DeepONet architecture proposed in Lu et al. [42] approximates the solution operator $G(f)(x)$ as

$$G_{\theta}(f)(x) = c_0 + \sum_{i=1}^p c_i(f(x_1), \ldots, f(x_m); \theta_{\text{branch}})\psi_i(x; \theta_{\text{trunk}}),$$

(2)

where $c_0$ is a constant bias parameter; $c_i(\cdot)$, $i = 1, \ldots, p$, are obtained from a neural network, known as the branch network, that has the input function (e.g., external source term) $f(\cdot)$ measured at $\{x_i\}_{i=1}^m$ locations as input; $\psi_i(\cdot)$, $i = 1, \ldots, p$, are obtained from a neural network, known as the trunk network, that has the location $x$ as the input; and $\theta = [\theta_{\text{branch}}^T \theta_{\text{trunk}}^T c_0]^T$ is the parameter vector containing weights and biases $\theta_{\text{branch}}$ and $\theta_{\text{trunk}}$ from the branch and trunk networks, respectively, and $c_0$. This approximation is based on the universal approximation theorem in Chen and Chen [8] for operators. A schematic of the DeepONet architecture used in this study is shown in Fig. 1. The main advantage of using a DeepONet architecture over other standard ones, such as feedforward, recurrent, etc., is that DeepONet provides a mesh-independent approximation of the solution operator since DeepONet uses a set of parameters $\theta$ trained only once with different mesh discretizations [35,42].

Lanthaler et al. [34] showed that the DeepONet can be thought of as a composition of three components. The first component is an encoder $E$ that maps the input function space to $m$ sensor inputs $\{f(x_i)\}_{i=1}^m$ for $\{x_i\}_{i=1}^m \subset D$. The second component, the approximator $A$, models the relation between $N_f$ sensor inputs $\{f(x_i)\}_{i=1}^m$ and the coefficients $\{c_k\}_{k=1}^P$. The encoder and the approximator together comprise the branch network. The third or the final component of DeepONet is the reconstructor, which maps the output of the branch network to the response $u(x)$, thereby showing dependence on the trunk network outputs $\{\psi_i\}_{i=1}^P$. With approximate inverses of the encoder and reconstructor defined as decoder $D$ and projector $P$, respectively, three errors associated with DeepONet approximation can be defined as encoding error $E_E = \|D \circ E - G\|_{L^2(\mu)}$, approximation error $E_a = \|A - P \circ G \circ D\|_{L^2(\mu)}$, and reconstruction error $E_r = \|R \circ P - G \circ D\|_{L^2(\mu)}$, where $\mu$ is an identity matrix; $\mu$ is the measure associated with $f$; $E_a$ and $G_{\theta}$ are pushforward measures associated with the encoder and operator $G$, respectively. Figure 2 shows the three components of a DeepONet and associated errors as defined here. The total approximation error associated with a DeepONet depends on these three errors as outlined in Theorem 3.2 in Lanthaler et al. [34].

The parameter vector $\theta$ is estimated by solving the optimization problem

$$\min_\theta J(\theta) = \frac{1}{N_f N_d} \sum_{j=1}^{N_f} \sum_{i=1}^{N_d} \left| G_{\theta}(f_j)(x_i) - G(f_j)(x_i) \right|^2,$$

(3)

where in the training dataset for each of $N_f$ realizations of the external source term $\{f_j(x)\}_{j=1}^{N_f}$, $N_d$ measurements of the response at locations $\{x_i\}_{i=1}^{N_d}$, which might not be the same as the locations where the external source term is measured, are used. These sensor locations can be random or equispaced inside the domain and kept fixed during training. The optimization problem in (3) can be solved using stochastic gradient descent [4,5,14] to obtain optimal parameter values of the DeepONets. In this paper, we use a popular variant of stochastic gradient descent, known as the Adam algorithm [31], to train DeepONets. Note that in Lu et al. [42], the input function $f(x)$ is sampled from function spaces with the locations $x$ as domain and the external source $f$ as range, such as Gaussian random field and Chebyshev polynomials. However, for practical reasons, in this paper, we use probability distribution of the uncertain variables associated with the problems to sample the input function. The number of training data for DeepONets can be limited, nonetheless, if the computational cost of the simulation of the physical system or the cost of repeating the experiments
is significant. In this study, to address the challenge of training DeepONets in the presence of limited data, we propose a bi-fidelity training approach utilizing a computationally inexpensive low-fidelity model of the physical system. In the next section, we detail this proposed approach and discuss two applications to address parametric and structural uncertainties present in the system.

3 Methodology

In the present work, we assume a low-fidelity model of the physical system in (1) is given by

\[ L_l(u_l)(x) = f_l(x), \quad x \in \Omega, \quad (4) \]

where \( u_l(x) \) is the low-fidelity response; \( L_l \) is the differential operator related to the low-fidelity model; and \( f_l(x) \) is the external source in the low-fidelity model. Next, we denote the discrepancy between the solution of the low-fidelity model \( u_l(x) \) and that of the physical system \( u(x) \) as

\[ u_d(x) = u(x) - u_l(x). \quad (5) \]

As it is often the case that the low-fidelity model will capture many important aspects of the physical system’s response, we assume that the training of a DeepONet for modeling \( u_d(x) \) is easier than modeling the true response \( u(x) \) and that generating the low-fidelity response \( u_l(x) \) is computationally inexpensive. We present two applications of the proposed approach next. Note that \( u_d(x) \) is estimated from the DeepONet and \( u_l(x) \) from the low-fidelity model for any new \( x \in \Omega \) to predict the response \( u(x) \). Further, this new location \( x \) need not be the same locations used during training or the places where the external source is measured due to the use of DeepONet to model \( u_d(x) \). We also do not use another DeepONet or neural network to model the low-fidelity response as the simulation of the low-fidelity model is computationally
inexpensive, and this avoids any approximation error introduced through a low-fidelity trained network.

### 3.1 Application I: systems with parametric uncertainty

The proposed approach can be used for modeling systems with parametric uncertainty, where a computationally inexpensive low-fidelity model of a similar system is available. For example, consider a system described by

\[
(L_d + L_s(\xi)) (u)(x; \xi) = f(x; \xi),
\]

(6)

where \( \xi \) is the vector of uncertain variables; \( L_d \) and \( L_s \) are deterministic (i.e., without any uncertain coefficients) and uncertain (or stochastic, i.e., containing all terms with uncertain coefficients) parts of the differential operator in (1), respectively; the response \( u \) depends on the uncertain variables \( \xi \); and the external source may depend on \( \xi \) as well. In this case, a low-fidelity model for (6) can be defined as follows

\[
L_d(u_1)(x; \xi) = f_1(x).
\]

Hence, the response of the physical system is approximated as

\[
\begin{align*}
 u(x; \xi) &= u_1(x; \xi) + u_d(x; \xi) \\
 &\approx u_1(x; \xi) + G_\theta(u_1)(x, \xi) \\
 &= u_1(x; \xi) + c_0 + \sum_{i=1}^{p} c_i(u_1(x_1; \xi), \ldots, u_1(x_m; \xi); \theta_{\text{branch}}) \psi_i(x, \xi; \theta_{\text{trunk}}),
\end{align*}
\]

(8)

where \( \{u_1(x_j; \xi)\}_{j=1}^{m} \) are used as the input of the branch network and \( (x, \xi) \) as the input of the trunk network to model the discrepancy \( u_d(x; \xi) \). A schematic of the approximation in (8) is shown in Fig. 3 with differences from the standard DeepONet schematic in Fig. 1 highlighted.

As a special case, when the operators \( L_d \) and \( L_s(\xi) \) are linear, using (5) in (6) we arrive at

\[
u_d(x; \xi) = - (L_d + L_s(\xi))^{-1} (L_s(\xi)(u_1)(x) - f_u(x; \xi)),
\]

(9)

which is modeled using a DeepONet, where \( u_1(x) \) is the solution to (7) and \( f_u(x; \xi) = f(x; \xi) - f_1(x) \). Note that we do not necessarily require the linearity of the differential operators for the proposed approach to apply. However, we make that assumption to illustrate the simplified construction of the low-fidelity model and the correction.

### 3.2 Application II: systems with structural uncertainty

In this application, we propose to use (5) to model the response of a partially unknown system. For example, let us assume we can rewrite (1) as follows

\[
(L_k + L_u)(u)(x) = f_k(x) + f_u(x),
\]

(10)

where \( L_k \) and \( L_u \) are the known and unknown parts of the differential operator, respectively; and \( f_k \) and \( f_u \) are the known and unknown parts of the external source, respectively. A low-fidelity model for this application can be defined as

\[
L_k(u_1)(x) = f_k(x).
\]

(11)

The response of the physical system is approximated next as

\[
\begin{align*}
 u(x) &= u_1(x) + u_d(x) \\
 &\approx u_1(x) + G_\theta(u_1)(x) \\
 &= u_1(x) + c_0 + \sum_{i=1}^{p} c_i(u_1(x_1), \ldots, u_1(x_m); \theta_{\text{branch}}) \psi_i(x, \theta_{\text{trunk}}),
\end{align*}
\]

(12)

where we use \( \{u_1(x_j)\}_{j=1}^{m} \) as the branch network input and \( x \) as the trunk network input. We utilize a similar schematic as shown in Fig. 3 for the approximation in (12), except we only use \( x \) as input to the trunk network.

As a special case, assuming the differential operators \( L_k \) and \( L_u \) are linear leads to the following relation for \( u_d(x) \) in (5)

\[
u_d(x) = - (L_k + L_u)^{-1} (L_u(u_1)(x) - f_u(x)).
\]

(13)

As before, we do not necessarily require linear differential operators to apply the proposed method. Note that it is straightforward to apply the proposed approach for an uncertain and partially unknown system using \( (x, \xi) \) as trunk network input, as in the previous subsection and addresses both parametric and structural uncertainties. Also, the low-fidelity model can be uncertain in these two applications, but for brevity we do not explicitly write the dependence of the low-fidelity solution on \( \xi \) in this section.

### 3.3 Choice of low-fidelity model

The choice of the low-fidelity model can have a significant effect on the total DeepONet approximation error. As discussed in Sect. 2, with a three-component decomposition of the DeepONet, Lanthaler et al. [34] showed that the total
error in a DeepONet approximation depends on errors in each of these components [34, Eq. (3.4)]. Among them, the error in the approximator depends on the neural network approximation error, such as in Yarotsky [66, Thm. 1]. On the other hand, the errors in encoder and reconstructor depend on the eigenvalues of covariance operator of the branch network inputs and \( G(f)(x, \xi) \) [34], respectively. In particular, these errors can be bounded from below using \( \sqrt{\sum \lambda_i} \), where \( \lambda_1 \geq \lambda_2 \geq \ldots \) are eigenvalues of the covariance operator of the branch network input for encoding error \( E_e \) and eigenvalues of the covariance operator of \( G(f)(x, \xi) \) for reconstruction error \( E_r \), respectively.

In the bi-fidelity DeepONet, we are replacing the external source term with the low-fidelity response in the branch network input, and the discrepancy \( u_d(x, \xi) \) is modeled using the DeepONet. Hence, an ideal choice for the low-fidelity model that will reduce the encoding error \( E_e \) will be a low-pass filter system that will remove most of the eigenvalues for \( i > p \) of the covariance operator of the branch network input. Also, in many cases, encoding the low-fidelity model’s response will be easier than encoding the external source reducing the encoding error. Similarly, to reduce the reconstruction error \( E_r \), we can select a low-fidelity model such that the eigenvalues of covariance of \( u_d(x, \xi) \) has smaller magnitudes for \( i > p \). However, in practice, we may be limited in terms of choosing the low-fidelity model. A simplified physics or a coarser discretization is often used as the low-fidelity model in most of the multi-fidelity methods. In our numerical examples described in Sect. 4, we will show that these practical low-fidelity models can still provide significant improvements in the total DeepONet approximation error.

As an elementary example to illustrate the above discussion, we consider a duffing oscillator with governing differential equation

\[
\frac{d^2 u(t; \xi)}{dt^2} + \delta \frac{du(t; \xi)}{dt} + \alpha u(t; \xi) + \beta u^3(t; \xi) = \gamma \cos \omega t; \quad t \in [0, 10], \quad u(0) = 0, \quad u_0 = 0,
\]  

where \( \delta \) is the damping coefficient; \( \alpha \) is the linear stiffness coefficient; \( \beta \) is the nonlinear stiffness coefficient; \( \gamma \) is the amplitude of the external force; \( \omega \) is the frequency of the external force; and \( u_0 \) is the initial response of the system. We consider the two applications of bi-fidelity DeepONet discussed in Sect. 3. In the first application, the parameters \( \beta \) and \( \omega \) are assumed as uncertain with \( \beta \sim \mathcal{U}[0.5, 2.5] \) and \( \omega \sim \mathcal{U}[1.5, 2.5] \), where \( \mathcal{U}[a, b] \) denotes a uniform distribution between limits \( a \) and \( b \), and a low-fidelity model that does not include the term \( \beta u^3(t; \xi) \) is considered. Here, we use \( \alpha = 1.0, \delta = 1.0, \) and \( \gamma = 2.5 \). For this case, Fig. 4a shows the lower bound of the reconstruction error \( E_r \), computed using \( \sqrt{\sum \lambda_i} \), where \( \lambda_1 \geq \lambda_2 \geq \ldots \) are eigenvalues of the sample covariance matrix of \( u_d(x, \xi) \) estimated using 2000 realizations of the random parameters [34, Th. 3.4]. In the second application, we use a low-fidelity model that does not consider the term \( \beta u^3(t; \xi) \) and \( \delta \) is replaced with a different \( \delta_{\text{nom}} = 1.5 \). Here, we assume \( \alpha = 1.0, \delta = 1.0, \) and \( \gamma = 2.5 \) as before, but \( \beta = 1.0 \). The frequency of the external load is assumed to be uniformly distributed as in the previous case. The lower bound of the reconstruction error \( E_r \), computed similarly for this case is shown in Fig. 4b. These two plots show that for smaller \( p \) in (2), the bi-fidelity DeepONet provides significantly lower bound for the reconstruction error. For large \( p \), this lower bound for both bi-fidelity DeepONet and the DeepONet described in Sect. 2, denoted as standard in the rest of the paper, reaches a small value.
Experimental results for these two applications of bi-fidelity DeepONet are provided in the next section. Note that we can also choose different low-fidelity models that do not necessarily provide a smaller lower bound for the reconstruction error, but the approximation and encoding errors are smaller with bi-fidelity DeepONet compared to a standard DeepONet as is done in Example 4.2.

4 Numerical examples

In this section, we illustrate the proposed approach in approximating the discrepancy between the low-fidelity response and the physical system’s response with DeepONets using three numerical examples. The first example uses nonlinear ordinary differential equation for an initial value problem. In the second example, we model nonlinear heat transfer in a thin plate, where a partial differential equation for the boundary value problem is used. We consider both parametric and structural uncertainties to illustrate the applications I and II of the proposed approach from Sect. 3 in these two examples. The third example considers the flow through a wind farm with a combination of parametric and structural uncertainties, where two-dimensional and three-dimensional simulations are used as the low- and high-fidelity model of the system, respectively. In these examples, we use two separate datasets, namely a training dataset $\mathcal{D}_{tr} = \{(x_i, \xi_i), u(x_i; \xi_i), \{f^{(i)}(x_j)\}_{j=1}^{m}\}_{i=1}^{N_{tr}}$ and a validation dataset $\mathcal{D}_{val} = \{(x_i, \xi_i), u(x_i; \xi_i), \{f^{(i)}(x_j)\}_{j=1}^{m}\}_{i=1}^{N_{val}}$.

We use the training dataset $\mathcal{D}_{tr}$ to train the DeepONets, and the accuracy of the trained DeepONets is measured using the validation dataset $\mathcal{D}_{val}$, with the relative validation root mean squared error (RMSE), $\varepsilon_{val}$, defined as follows

$$\varepsilon_{val} = \frac{\|u_{\text{pred}} - u_{\text{val}}\|_2}{\|u_{\text{val}}\|_2},$$

where $\|\cdot\|_2$ is the $\ell_2$-norm of its argument, $u_{\text{pred}}$ is the vector of predicted response using DeepONets, and $u_{\text{val}}$ is from the validation dataset $\mathcal{D}_{val}$. Note that we select the number of hidden layers in the trunk and branch networks and the number of neurons per hidden layer using an iterative procedure, where we increase the number of neurons per hidden layer gradually up to an upper limit (100 in this study) and then add hidden layers to the network until we observe no reduction of $\varepsilon_{val}$. For the number of terms $p$ in (8) and (12), we increase $p$ in multiples of 10 to a maximum to observe any reduction of $\varepsilon_{val}$. We also found the ELU activation function [10] to produce the best results for the numerical examples in this section compared to other common activation functions, such as rectified linear unit (ReLU), leaky ReLU, logistic sigmoid, hyperbolic tangent, etc.

4.1 Example 1: duffing oscillator

In this example, we use a duffing oscillator described by the governing differential equation (14), which we repeat here.
for continuity,
\[
\frac{d^2 u(t; \xi)}{dt^2} + \delta \frac{du(t; \xi)}{dt} + \alpha u(t; \xi) + \beta u^3(t; \xi) = \gamma \cos \omega t; \quad t \in [0, 10], \quad u(0) = 0,
\] (16)

where \( \delta \) is the damping coefficient; \( \alpha \) is the linear stiffness coefficient; \( \beta \) is the nonlinear stiffness coefficient; \( \gamma \) is the amplitude of the external force; \( \omega \) is the frequency of the external force; and \( u_0 \) is the initial response of the system. We use a neural network that has three hidden layers with 40 neurons each as the branch network and a neural network that has two hidden layers with 40 neurons each as the trunk network. We use ELU as the activation function in these networks [10]. The number of terms in the summation in (2) is fixed at \( p = 20 \) following the procedure discussed in the previous section.

### 4.1.1 Application I: parametric uncertainty

For the oscillator defined in (16), we use \( \alpha = 1.0, \delta = 1.0, \) and \( \gamma = 2.5. \) The uncertain variables are assumed as \( \xi = (\beta, \omega) \) with \( \beta \sim \mathcal{U}(0.5, 2.5) \) and \( \omega \sim \mathcal{U}(1.5, 2.5), \) where \( \mathcal{U}(a, b) \) denotes a uniform distribution between limits \( a \) and \( b. \) We use a training dataset \( \mathcal{D}_t \) with \( N_t = 200 \) and a validation dataset \( \mathcal{D}_{val} \) with \( N_{val} = 1000 \) realizations of the uncertain variables uniformly sampled from their corresponding distributions, respectively, where the response is measured at 100 equidistant points along \( t. \) A low-fidelity model of the system can be specified without the nonlinear stiffness, as follows

\[
\frac{d^2 u_l(t; \xi)}{dt^2} + \delta_{nom} \frac{du_l(t; \xi)}{dt} + \alpha u_l(t; \xi) = \gamma \cos \omega t; \quad t \in [0, 10], \quad u_l(0) = 0,
\] (17)

where \( u_l(t; \xi) \) is the low-fidelity response. Note that uncertainty in the low-fidelity model only comes through the uncertain variable \( \omega. \) Figure 5a depicts a representative realization of \( u(t; \xi) \), the low-fidelity response \( u_l(t; \xi) \), and the discrepancy \( u_d(t; \xi) \). Note that in this example the cost of generating the low- and high-fidelity datasets is insignificant.

The proposed use of DeepONet to model the discrepancy \( u_d(t; \xi) \) between the low-fidelity and the true responses produces \( 4.72 \times 10^{-3} \) as the validation error \( \varepsilon_{val} \) with the validation dataset \( \mathcal{D}_{val}. \) In contrast, the standard DeepONet results in a validation error of \( 1.11 \times 10^{-2}. \) Figure 5b compares the histograms of the relative validation RMSE of these two approaches. Hence, in this case, the proposed approach improves the validation error by more than a factor of two, as modeling of the discrepancy \( u_d(t; \xi) \) proved to be easier using the same DeepONet architecture than modeling the response \( u(t; \xi). \) We also note that due to the absence of the parameter \( \beta \) in (17), the low-fidelity response misses some important characteristics of the true response. So, we expect the improvement compared to high-fidelity only approach will be larger if the dataset from low-fidelity model is able to capture more aspects of the true response.

### 4.1.2 Application II: partially unknown system (structural uncertainty)

In this case, we assume \( \alpha = 1.0, \delta = 1.0, \) and \( \gamma = 2.5 \) as before, but \( \beta = 1.0. \) The frequency of the external load is assumed to be uniformly distributed as before. We use a training dataset \( \mathcal{D}_t \) with \( N_t = 200 \), where we also add 5\% additive Gaussian white noise to the dataset, and a validation dataset \( \mathcal{D}_{val} \) with \( N_{val} = 1000. \) Note that other error distributions can also be assumed for the dataset. The low-fidelity model is assumed as

\[
\frac{d^2 u_l(t; \xi)}{dt^2} + \delta_{nom} \frac{du_l(t; \xi)}{dt} + \alpha u_l(t; \xi) = \gamma \cos \omega t; \quad t \in [0, 10], \quad u_l(0) = 0,
\] (18)

where the nominal value of the damping coefficient is \( \delta_{nom} = 1.5. \) Hence, the unknown dynamics not modeled in the low-fidelity model can be given by

\[
\mathcal{L}_u u(t; \xi) = (\delta - \delta_{nom}) \frac{du(t; \xi)}{dt} + \beta u^3(t; \xi).
\] (19)

Fig. 6a shows a comparison of the response from the low-fidelity model \( u_l(t; \xi) \), true response \( u(t; \xi) \), and their difference \( u_d(t; \xi) \). When DeepONet is used to model the discrepancy \( u_d(t; \xi) \) between the true response and the low-fidelity model response, it produces a validation error \( \varepsilon_{val} = 9.98 \times 10^{-3}. \) Note that the standard use of DeepONet produces an \( \varepsilon_{val} = 2.22 \times 10^{-2} \) with the same training and validation datasets. Figure 6a shows a comparison of histograms from the proposed use of DeepONet to model \( u_d(t; \xi) \) and the standard use of DeepONet as described in Sect. 2. The figure shows that, in this case as well, modeling \( u_d(t, \xi) \) as a function of the low-fidelity response \( u_l(t, \xi) \) using DeepONet can achieve better accuracy compared to the standard use. Hence, this example shows the effectiveness of the proposed approach to model the discrepancy using DeepONet for uncertain and partially unknown dynamical systems, which reduces the validation error by a factor of more than two.

### 4.2 Example 2: nonlinear steady-state heat transfer in a thin plate

In our second example, we use a two-dimensional steady-state heat transfer problem in a thin L-shaped plate with dimensions shown in Fig. 7a. The left boundary is assumed
Fig. 5 The use of DeepONets to model the response of an uncertain Duffing oscillator. Here, a standard use of DeepONet as described in Sect. 2 is denoted as ‘Standard’, and the proposed use of DeepONet to model the discrepancy between the low-fidelity model and true system is denoted as ‘Bi-fidelity’ (a) Comparison of the low-fidelity response $u_1(t; \xi)$, correction $u_d(t; \xi)$ and true response $u(t; \xi)$ for one realization of the uncertain variables (b) Relative validation error in predicting the response $u(t; \xi)$ using DeepONets.

Fig. 6 The use of DeepONets to model the response of a partially unknown Duffing oscillator in Example 1. (a) Comparison of the low-fidelity response $u_1(t; \xi)$, correction $u_d(t; \xi)$ and true response $u(t; \xi)$ for one realization of the uncertain variables (b) Relative validation error in predicting the response $u(t; \xi)$ using DeepONets.

to have a constant temperature of 100 K. The steady-state temperature at any location inside the plate, $T(x)$, is given by

$$-h_t \nabla (k(x) \nabla T(x)) + 2Q_c(x) + 2Q_r(x) = Q_i(x),$$

where $\nabla := \frac{\partial}{\partial x_1} + \frac{\partial}{\partial x_2}$; $h_t$ is the thickness of the plate; $k(x)$ is the thermal conductivity of the plate; $Q_c(x)$ is the amount of heat transferred through one face of the plate per unit area using convection; $Q_r(x)$ is the amount of heat transferred through one face of the plate per unit area using radiation; and $Q_i(x)$ is the amount of heat added to the plate per unit area from an internal heat source. Heat transferred by convection can be given by Newton’s law of cooling as

$$Q_c(x) = h_c(T(x) - T_a),$$

where $h_c$ is the specified convection coefficient and $T_a$ is the ambient temperature. For heat transferred due to radiation, the Stefan-Boltzmann law can be used,

$$Q_r(x) = \epsilon \sigma (T^4(x) - T_a^4),$$

where $\epsilon$ is the emissivity of the plate surface and $\sigma$ is the Stefan-Boltzmann constant of the plate. Given (21) and (22), we rewrite (20) as

$$-h_t \nabla (k(x) \nabla T(x)) + 2h_c T(x) + 2\epsilon \sigma T^4(x) = 2h_c T_a + 2\epsilon \sigma T_a^4 + Q_i(x).$$

In this example, we set $h_t = 0.01$ m; $h_c = 1$ W/(m²K); $\epsilon = 0.025$; and $\sigma = 5.670373 \times 10^{-8}$ W/(m²K⁴) (similar to copper) to generate the datasets. We also assume the internal heat source to be an uncertain squared exponential source centered at the (0,0) corner given by

$$Q_i(x) = Q_0(1 + 0.1 \xi_0) \exp \left( -\sum_{i=1}^{2} \xi_i^2 \right),$$

where $\xi_0$ is the uncertain variable.
where \( Q_0 = 5 \times 10^3 \, \text{Wm}^{-2} \) and \( \xi_Q \) is a standard Gaussian random variable. We use the finite element method with the mesh shown in Fig. 7a to solve (23). Figure 7b shows a realization of the steady-state temperature in the plate. In this example, we use a neural network that has three hidden layers with 40 neurons each and ELU activation as the branch network, and a neural network that has two hidden layers with 40 neurons each and ELU activation as the trunk network. The number of terms in the summation in (2) is fixed at \( p = 20 \) that produces the smallest validation error.

### 4.2.1 Application I: uncertain thermal conductivity

In our first application, we model the thermal conductivity as

\[
k(x) = k_0 + \exp(z(x)),
\]

(25)

where \( k_0 = 400 \, \text{W(m·K)}^{-1} \) and \( z(x) \) is a zero mean Gaussian random field with a square exponential covariance function \( K(x, x') \)

\[
K(x, x') = \exp\left(-\frac{1}{2} \sum_{i=1}^{2} \frac{x_i - x_i'}{l_i}^2 \right),
\]

(26)

where \( l_i, \, i = 1, 2, \) are length scale parameters, which we choose as 0.5 for both dimensions. A Karhunen-Loève (KL) expansion truncated at the 25th term that captures 97% of the energy (defined as the sum of all eigenvalues) is used to generate realizations of the random field \( z(x) \). For the low-fidelity model, we consider

\[-h_r k_0 \nabla^2 T_l(x) + 2h_c T_l(x) + 2\varepsilon \sigma T_l^4(x) = 2h_c T_a + Q_l^{\text{nom}}(x),
\]

(27)

where \( T_l \) is the temperature given by the low-fidelity model; and \( Q_l^{\text{nom}}(x) = Q_0 \exp\left(-\sum_{i=1}^{2} \xi_{Q_i} x_i^2 \right) \). Hence, the low-fidelity model is deterministic and we only need to solve it once. Here, the input to the trunk network of the bi-fidelity DeepONet that models the discrepancy (see Fig. 3) contains 25 standard normal random variables used in the KL expansion of \( z(x) \). \( \xi_Q \) in (24), and the location \( x \) for predicting the temperature. We use \( N_t = 30 \) realizations of the uncertain variables in the training dataset \( D_t \) and \( N_{val} = 1000 \) realizations for the validation dataset \( D_{val} \). For each of these realizations, we measure the temperature of the plate at 200 locations marked using circles and crosses in Fig. 8a. During training, we use the temperature at 100 of these locations marked with circles and validate the DeepONets by predicting the temperature at the remaining 100 locations marked with crosses. We compare the result to the standard use of DeepONet as discussed in Sect. 2, where a sample of \( Q_l(x) \) and the right hand side of (23), measured at the training locations, marked with circles in Fig. 8a, are used as input to the branch network.

Using the bi-fidelity DeepONet, the validation error \( \varepsilon_{val} \), as defined in (15), is \( 6.72 \times 10^{-3} \). However, the validation error \( \varepsilon_{val} \) of DeepONet is \( 8.70 \times 10^{-2} \). Figure 8b shows the histograms comparing the errors from these two approaches, denoted as ‘standard’ and ‘bi-fidelity’, respectively. The validation error \( \varepsilon_{val} \) and the histograms show that the proposed approach improves the accuracy of the predictions by almost an order of magnitude. This is primarily due to the fact that modeling of the temperature discrepancy can be performed relatively accurately using DeepONets in the presence of a small training dataset.

### 4.2.2 Application II: partially unknown system (structural uncertainty)

Next, we assume that the low-fidelity model is unaware of the radiation component in (23), i.e.,

\[-h_r k_0 \nabla^2 T_l(x) + 2h_c T_l(x) = 2h_c T_a + Q_l^{\text{nom}}(x),
\]

(28)

where \( Q_l^{\text{nom}}(x) = Q_0 \exp\left(-\sum_{i=1}^{2} x_i^2 \right) \) is used. Hence, the low-fidelity model is deterministic and we only need to solve it once. Here, we use \( x \) and \( \xi_Q \) as the input to the trunk network. The training dataset \( D_t \) and the validation dataset \( D_{val} \) consist of \( N_t = 3 \) and \( N_{val} = 1000 \) data points, respectively, similar to the previous subsection. For each of these realizations, we measure the temperature of the plate at 200 locations marked using circles and crosses in Fig. 9a. During training, we use the temperature at 20 of these locations marked with circles and validate the DeepONets by predicting the temperature at the remaining 180 locations marked with crosses. When using DeepONet for modeling the discrepancy between the low-fidelity model and the true system, the validation error \( \varepsilon_{val} \) is \( 5.32 \times 10^{-3} \). The standard use of DeepONet, however, gives \( \varepsilon_{val} = 1.46 \times 10^{-2} \), one order of magnitude larger than the proposed bi-fidelity DeepONet. Figure 9b compares the histograms of the errors from these two approaches, illustrating the advantage of the bi-fidelity approach.

### 4.3 Example 3: wind farm

In our third example, we model the power generated in a wind farm and consider uncertain wind speed, inflow direction, and yaw angle. Estimating the expected power with respect to the joint distribution of wind speed and direction is a common calculation in determining the annual energy production of a wind plant; however, the number of function evaluations required often precludes the use of high-fidelity models in
industry [30]. Deliberately yawing wind turbine rotors out of perpendicular to incoming wind is an emerging flow control technique called wake steering, which can improve the overall plant output by directing wind turbine wakes away from downwind turbines [19]. The wake steering optimization problem is fraught with uncertainty due to misalignment in the nacelle-mounted wind vanes located just downwind of the rotor [50]. We consider two layouts of a wind farm with the number of turbines six and 64, respectively. For the probability distribution of uncertain wind speed and direction, we use 10 min-average measurement data from the Princess Amalia wind farm in the Netherlands. The joint probability density function of the wind speed and direction estimated using these measurements is shown in Fig. 10.

For each realization of the wind speed, direction, and yaw angle, we use the WindSE toolkit [29], a Python package with FEniCS [41] as the backend solver, to generate the training and validation datasets. Two-dimensional (2D) Reynolds Averaged Navier–Stokes (RANS) simulations using the finite element mesh shown in Fig. 12a are used to generate the low-
fidelity data. Three-dimensional (3D) simulations using the mesh shown in Fig. 12b are used to estimate the power generated from the true system. For the two- and three-dimensional simulations, a mixing length model [49][48, 7.1.7] is used for closing the turbulent stress term, with a uniform mixing length \( l_m = 15 \text{ m} \). In the three-dimensional simulations, a velocity log-profile is used for the atmospheric boundary layer with von Kármán constant \( \kappa = 0.4 \). The log profile is such that the hub-height velocity is the value of the sample drawn from the pdf shown in Fig. 10. In the two-dimensional simulations, the governing equations follow the 2D approximation of Boersma et al. [3] to approximate the flow field at hub-height, which is sufficient for estimating the power output.

We use a training \( D_t \) and validation \( D_{\text{val}} \) datasets consisting of \( N_t = 100 \) and \( N_{\text{val}} = 1000 \) realizations of the uncertain variables and power generated. The realizations of wind speed and wind direction are generated using the histograms from the dataset for the Princess Amalia wind farm in the Netherlands. We use a neural network with three hidden layers and 100 neurons each with ELU activation as the branch network and another neural network with two hidden layers and 100 neurons each with ELU activation as the trunk network in this example. In (2), we truncate the summation at \( p = 60 \) that produces the smallest validation error. Results for two layouts with six and 64 turbines are discussed next.

4.3.1 Layout 1: six turbines

First, we model the power generated in a wind farm with six turbines arranged in a \( 3 \times 2 \) grid as shown in Fig. 11a. The wind turbines are placed 474 m apart in the \( x_1 \) direction and 387 m apart in the \( x_2 \) direction. We use the NREL 5 MW reference turbine [26], which has a 126 m rotor diameter and 90 m hub-height. Figure 11b shows a single turbine with yaw angle between the wind direction and the turbine axis, which is used as an input parameter in this example.

We consider three cases here. In Case I, we only consider wind speed and direction as uncertain variables and assume yaw angle to be zero for all the turbines. In Case II, we assume the yaw angle to be uncertain and uniformly distributed between \([-\pi/2, \pi/2]\), but the same for all the turbines. In the final Case III, we assume each of the six turbines has uncertain yaw angles that are independent and identically distributed uniformly between \([-\pi/2, \pi/2]\).

We compare the results with a standard DeepONet that uses wind speed, direction, and yaw angle (for the last two cases) as branch network input and the turbine location as the trunk network input. Note that one 2D simulation of this layout takes 4.69 s (average of 1000 runs) compared to 3.54 min (average of 1000 runs) for one 3D simulation on the National Renewable Energy Laboratory’s Eagle high-performance computing system (https://www.nrel.gov/hpc/eagle-system.html). Therefore, for a fair comparison, we train this DeepONet with a training dataset \( D_t \) consisting of 103 realizations of the uncertain variables and corresponding power generated (i.e., with similar computational cost of generating the training dataset compared to the bi-fidelity training dataset used in the proposed approach).

We list the results in Table 1, which shows that the proposed use of DeepONet to model the discrepancy between the power outputs from 2D and 3D simulations provides significant improvement in the validation error. Also, the standard use of DeepONet shows improvement when we add yaw angle as another random input. This is primarily due to the fact that yaw angle has a significant effect on the power generated, and the standard DeepONet is able to model that relation with increasing connection due to the addition of a new input node in the network. A similar but slight improvement in the proposed approach can also be noticed in Cases II and III in Tables 2 and 3, respectively.

4.3.2 Layout 2: 64 turbines

In this layout, we consider 64 turbines in total arranged in a \( 8 \times 8 \) grid that is more characteristic of the size of a utility-scale wind plant. Other specifications of the turbine layout and the input parameters are the same as before. Figure 13 shows the two- and three-dimensional meshes used to generate the training and validation datasets. The results from the proposed bi-fidelity training of DeepONets are compared with a DeepONet (denoted as ‘standard’) trained only using a high-fidelity dataset with wind speed, direction, and yaw angle as branch network inputs and the turbine location as the trunk network input. Note that one 2D simulation of this layout takes 3.68 min. (average of 300 runs) compared to 1.15 hrs. (average of 300 runs) for one 3D simulation on the National Renewable Energy Laboratory’s Eagle.
### Table 1  
Network inputs and validation results for case I studied in Example 3 for Layout 1 with six turbines

| Case | Method       | Network input Branch                           | $\xi_{\text{val}}$  |
|------|--------------|-----------------------------------------------|---------------------|
| II   | Standard     | Wind speed, wind direction                    | $1.49 \times 10^{-1}$ |
|      | Bi-fidelity  | Turbine location                              |                     |
|      |              | **Power generated from 2D simulations**       |                     |
|      |              | Wind speed, wind direction, turbine location  | $1.03 \times 10^{-2}$ |

**Histograms of validation error in Branch Trunk predicting the generated power**

### Table 2  
Network inputs and validation results for case II studied in Example 3 for Layout 1 with six turbines

| Case | Method       | Network input Branch                           | $\xi_{\text{val}}$  |
|------|--------------|-----------------------------------------------|---------------------|
| II   | Standard     | Wind speed, wind direction, yaw angle        | $4.98 \times 10^{-2}$ |
|      | Bi-fidelity  | Turbine location                              |                     |
|      |              | **Power generated from 2D simulations**       |                     |
|      |              | Wind speed, wind direction, yaw angle, turbine location | $8.55 \times 10^{-3}$ |

**Histograms of validation error in Branch Trunk predicting the generated power**

### Table 3  
Network inputs and validation results for case III studied in Example 3 for Layout 1 with six turbines

| Case | Method       | Network input Branch                           | $\xi_{\text{val}}$  |
|------|--------------|-----------------------------------------------|---------------------|
| III  | Standard     | Wind speed, wind direction, individual yaw angle | $2.97 \times 10^{-2}$ |
|      | Bi-fidelity  | Turbine location                              |                     |
|      |              | **Power generated from 2D simulations**       |                     |
|      |              | Wind speed, wind direction, individual yaw angle | $9.06 \times 10^{-3}$ |
|      |              | turbine location                              |                     |
5 Conclusions

In this paper, we propose a bi-fidelity approach for modeling uncertain and partially unknown system responses using DeepONets. In this approach, we utilize a similar and computationally less expensive model of the physical system as a low-fidelity model. The difference between the response from the low-fidelity model and the true system is subsequently modeled using DeepONets. As is often the case, with low-fidelity model we can capture the important char-
characteristics of the system’s behavior. Therefore, we make a reasonable assumption in this study that the modeling of the discrepancy between the low-fidelity model and the system’s response can be done with a smaller high-fidelity dataset than a standard DeepONet to model the system’s response in full to achieve similar validation error. In particular, we use DeepONets with low-fidelity model response, the locations, and the realizations of the uncertain variables as inputs. We apply the proposed approach to problems with parametric as well as with structural uncertainty, where some of the physics is unknown or not modeled. First, a nonlinear oscillator’s response is modeled using this proposed approach, followed by the modeling of nonlinear heat transfer in a thin plate. Both examples show that the proposed approach provides an improvement in the validation error by almost an order of magnitude. Next, we model a wind farm with six wind turbines, where power generated from two-dimensional simulations is used as the low-fidelity response and the true system’s response is obtained using three-dimensional simulations. Again, the proposed approach provides significant improvement in validation error, showcasing the efficacy of the proposed approach. These three case studies demonstrate that the bi-fidelity DeepONet approach is a powerful tool in situations with limited high-fidelity data. In the future, we plan to develop a multi-fidelity extension of this approach to determine the effect of regularization in the loss function and modeling of complex multiphysics phenomena.

Acknowledgements This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. This work was supported by funding from DOE’s Advanced Scientific Computing Research (ASCR) program. The research was performed using computational resources sponsored by the Department of Energy’s Office of Energy Efficiency and Renewable Energy and located at the National Renewable Energy Laboratory. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes. The work of AD was partially supported by the AFOSR grant FA9550-20-1-0138.

References

1. Basu JK, Bhattacharyya D, Kim TH (2010) Use of artificial neural network in pattern recognition. Int J Softw Eng Appl 4(2):23–34
2. Blakseth SS, Rasheed A, Kvamsdal T et al (2021) Deep neural network enabled corrective source term approach to hybrid analysis and modeling. arXiv preprint arXiv:2105.11521
3. Boersma S, Doekemeijer B, Vai M et al (2018) A control-oriented dynamic wind farm model: WFsim. Wind Energy Sci 3(1):75–95
4. Bottou L (2010) Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT’2010. Springer, p 177–186
5. Bottou L (2012) Stochastic gradient descent tricks. In: Neural networks: tricks of the trade. Springer, p 421–436
6. Cai S, Mao Z, Wang Z et al (2021a) Physics-informed neural networks (PINNs) for fluid mechanics: a review. arXiv preprint arXiv:2105.09506
7. Cai S, Wang Z, Lu L et al (2021) DeepM&M:net: inferring the electroconvection multiphysics fields based on operator approximation by neural networks. J Comput Phys 436(110):296
8. Chen T, Chen H (1995) Universal approximation of nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Trans Neural Netw 6(4):911–917
9. Chi G, Hu S, Yang Y et al (2012) Response surface methodology with prediction uncertainty: a multi-objective optimisation approach. Chem Eng Res Des 90(9):1235–1244
10. Clevert DA, Unterthiner T, Hochreiter S (2015) Fast and accurate deep network learning by exponential linear units (ELUs). arXiv preprint arXiv:1511.07289
11. De S (2021) Uncertainty quantification of locally nonlinear dynamical systems using neural networks. J Comput Civ Eng 35(4):04021009
12. De S, Doostan A (2022) Neural network training using ℓ1-regularization and bi-fidelity data. J Comput Phys 458:111010
13. De S, Britton J, Reynolds M et al (2020) On transfer learning of neural networks using bi-fidelity data for uncertainty propagation. Int J Uncertain Quantif 10(6):543–573
14. De S, Hampton J, Maute K et al (2020) Topology optimization under uncertainty using a stochastic gradient-based approach. Struct Multidiscip Optim 62(5):2255–2278
15. De S, Maute K, Doostan A (2020) Bi-fidelity stochastic gradient descent for structural optimization under uncertainty. Comput Mech 66(4):745–771
16. Deng B, Shin Y, Lu L et al (2021) Convergence rate of DeepONets for learning operators arising from advection-diffusion equations. arXiv preprint arXiv:2102.10621
17. Doostan A, Owhadi H (2011) A non-adapted sparse approximation of PDEs with stochastic inputs. J Comput Phys 230(8):3015–3034
18. Duraisamy K, Iaccarino G, Xiao H (2019) Turbulence modeling in the age of data. Annu Rev Fluid Mech 51:357–377
19. Fleming P, Annoni J, Shah JJ et al (2017) Field test of wake steering at an offshore wind farm. Wind Energy Sci 2(1):229–239
20. Forresters A, Söbester A, Keane AJ (2007) Multi-fidelity optimization via surrogate modelling. Proc R Soc Math Phys Eng Sci 463(2088):3251–3269
21. Geneva N, Zabaros N (2020) Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. J Comput Phys 403(109):056
22. Ghanem RG, Spanos PD (2003) Stochastic finite elements: a spectral approach. Courier Corporation, North Chelmsford
67. Zhang D, Lu L, Guo L et al (2019) Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. J Comput Phys 397(108):850

68. Zhu Y, Zabaras N, Koutsourelakis PS et al (2019) Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. J Comput Phys 394:56-81

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.