Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport

Lu Lu,[1,*] Raphaël Pestourie,[2] Steven G. Johnson,[2] and Giuseppe Romano[3]
[1]*Department of Chemical and Biomolecular Engineering, University of Pennsylvania, Philadelphia, Pennsylvania 19104, USA*
[2]*Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA*
[3]*Institute for Soldier Nanotechnologies, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA*

(Received 14 April 2022; accepted 25 May 2022; published 13 June 2022)

Deep neural operators can learn operators mapping between infinite-dimensional function spaces via deep neural networks and have become an emerging paradigm of scientific machine learning. However, training neural operators usually requires a large amount of high-fidelity data, which is often difficult to obtain in real engineering problems. Here we address this challenge by using multifidelity learning, i.e., learning from multifidelity data sets. We develop a multifidelity neural operator based on a deep operator network (DeepONet). A multifidelity DeepONet includes two standard DeepONets coupled by residual learning and input augmentation. Multifidelity DeepONet significantly reduces the required amount of high-fidelity data and achieves one order of magnitude smaller error when using the same amount of high-fidelity data. We apply a multifidelity DeepONet to learn the phonon Boltzmann transport equation (BTE), a framework to compute nanoscale heat transport. By combining a trained multifidelity DeepONet with genetic algorithm or topology optimization, we demonstrate a fast solver for the inverse design of BTE problems.

DOI: 10.1103/PhysRevResearch.4.023210

I. INTRODUCTION

Scientific machine learning (SciML) has become an emerging research field, where deep learning in the form of deep neural networks (DNNs) is used to address challenging problems in computational science and engineering [1]. In the past a few years, different SciML methods have been proposed [1]. One notable method is physics-informed neural networks (PINNs) [1–3], which have been developed to solve both forward and inverse problems of ordinary and partial differential equations (ODEs and PDEs). PINNs and the extensions [4–7] have been applied to solve diverse scientific and engineering applications [1,8–10].

More recently, learning operators mapping between infinite-dimensional function spaces via DNNs has become a new and important paradigm of SciML [11,12]. DNNs of this type are called deep neural operators. A typical application of deep neural operators for PDEs is used as a surrogate solver of the PDE, mapping from initial conditions, boundary conditions, or forcing terms to the PDE solution. Different architectures of deep neural operators have been developed, such as deep operator networks (DeepONet) [11,13], Fourier neural operators [12,13], nonlocal kernel networks [14], and several others [15–17]. Among these deep neural operators, DeepONet was the first one to be proposed (in 2019) [11] based on the universal approximation theorem (UAT) of neural networks for operators [18], which is an extension of the well-known UAT for functions. A DeepONet consists of a network (called "branch net") for encoding the input function space and another network (called "trunk net") for encoding the domain of the output functions. Many subsequent extensions and improvements have been developed, such as DeepONet with proper orthogonal decomposition (POD-DeepONet) [13], DeepONet for multiple-input operators (MIONet) [19], DeepONet for multiphysics problems via physics decomposition (DeepM&Mnet) [20,21], DeepONet with uncertainty quantification [22–24], multiscale DeepONet [25], POD-DeepONet with causality [26], and physics-informed DeepONet [27,28].

DeepONet and its extensions have demonstrated good performance in diverse applications, such as fractional-derivative operators [11], stochastic differential equations [11], high-speed boundary-layer problems [29], multiscale bubble growth dynamics [30,31], solar-thermal systems [32], aortic dissection [33], multiphysics and multiscale problems of electroconvection [20] and hypersonics [21], power grids [23], and multiscale modeling of mechanics problems [34]. It has also been theoretically proved that DeepONet may break the curse of dimensionality in some PDEs [35–37].

Despite this computational and theoretical progress, training DeepONet usually requires a large amount of data to achieve a good accuracy. However, in many engineering problems, it is often difficult to obtain necessary data of high accuracy. In these situations, it may be advantageous to complement the high-fidelity data set (small and accurate) by employing a low-fidelity data set (larger and less

*Corresponding author: lulu1@seas.upenn.edu accurate), i.e., learning from multifidelity data sets [38–40]. Multifidelity learning can be used in diverse applications and scenarios. For example, the high-fidelity data set might be obtained from simulations with fine mesh, while the low-fidelity data set is from simulations with coarse mesh. In Ref. [40], the high-fidelity data set is from experiments and three-dimensional (3D) simulations, while the low-fidelity data set is from two-dimensional (2D) simulations and analytical solutions. Multifidelity learning by neural networks has been developed for a function approximation in Refs. [39,40]. Another recent approach to data efficiency using a low-fidelity solver trains a combination of a generative neural network and a low-fidelity solver, end-to-end, to match the limited data of a high-fidelity solver [41].

In this work we aim to alleviate the challenge of large data sets required for training DeepONet. We propose new algorithms to significantly reduce the required number of high-fidelity data. Specifically, we develop a new version of DeepONet by using residual learning and input augmentation that is capable of fusing together different sets of data with different fidelity levels, making the proposed multifidelity DeepONet very efficient for learning complex systems.

As an application, we apply multifidelity DeepONet to learn heat transport in nanostructured materials, described by the phonon Boltzmann-transport equation (BTE) [42]. We first train a multifidelity DeepONet to predict the map of magnitude of thermal flux in a Si nanoporous system; then, the trained DeepONet is employed with genetic algorithms and topology optimization for inverse design of materials to maximize heat transport over prescribed points with a constrained number of pores. Our method is validated against numerical BTE simulations. We note that the same DeepONet can be used for multiple inverse design tasks without the need for retraining. Potential applications in this area include energy harvesting and heat management [43–45].

The paper is organized as follows. In Sec. II, after introducing the algorithm of DeepONet, we present the setup of multifidelity learning and a few different approaches of multifidelity DeepONet. Then we discuss how to apply the trained DeepONet for inverse design. In Sec. III we compare different approaches of multifidelity DeepONet and demonstrate the effectiveness on a Poisson equation and BTE problem. Subsequently, we solve several inverse design problems of BTE for different objective functions. Finally, we conclude the paper in Sec. IV.

II. METHODS

We first introduce DeepONet and then present different approaches of multifidelity learning for DeepONet. We also discuss how to use the learned multifidelity DeepONet for inverse design.

A. DeepONet

DeepONet was proposed in Ref. [11] for learning nonlinear operators mapping from a function to another function, e.g., from the boundary/initial condition to the PDE solution. We denote the input function by v defined on the domain $D \in \mathbb{R}^d$:

$$v : D \ni x \mapsto v(x) \in \mathbb{R},$$

and the output function by u defined on the domain $D' \in \mathbb{R}^{d'}$:

$$u : D' \ni \xi \mapsto u(\xi) \in \mathbb{R}.$$

Let $\mathcal{V}$ and $\mathcal{U}$ be the spaces of v and u, respectively, and then the operator is

$$\mathcal{G} : \mathcal{V} \ni v \mapsto u \in \mathcal{U}.$$

DeepONet is used to learn the mapping $\mathcal{G}$ from a data set of v and u. In a DeepONet there are two subnetworks: a branch net and a trunk net [Fig. 1(a)]. The branch net takes the evaluations of v at $\{x_1, x_2, \ldots, x_m\}$ as its input, while the trunk net takes ξ as its input. Then the DeepONet output is an inner product of the branch and trunk outputs

$$\mathcal{G}(v)(\xi) = \sum_{k=1}^{p} b_k(v)t_k(\xi) + b_0,$$

where b_0 is a bias.

B. Multifidelity DeepONet

DeepONet can be used to learn diverse nonlinear operators, but it usually requires a large high-fidelity data set for training, which is generated by expensive numerical simulations. Here we propose multifidelity DeepONet to reduce dramatically the required high-fidelity data set by leveraging an additional low-fidelity data set.

1. Multifidelity learning

The idea of multifidelity learning is that instead of generating a large data set of high accuracy (i.e., high fidelity), we only generate a small high-fidelity data set, but in the meanwhile, we generate another data set of low accuracy (i.e., low fidelity). The low-fidelity data set is much cheaper to generate, and thus it is easy to generate a large low-fidelity data set. In short, in multifidelity learning, we have two data sets:

(i) a high-fidelity data set of size N_H:

$$\mathcal{T}_H = \{(v_i^H, u_i^H = \mathcal{G}_H(v_i^H))\}_{i=1,2,\ldots,N_H},$$

(ii) a low-fidelity data set of size N_L:

$$\mathcal{T}_L = \{(v_i^L, u_i^L = \mathcal{G}_L(v_i^L))\}_{i=1,2,\ldots,N_L},$$

where $N_H \ll N_L$. $\mathcal{G}_H$ (i.e., the original $\mathcal{G}$) is the high-fidelity operator we aim to learn, and $\mathcal{G}_L$ is the low-fidelity operator. Then we use both data sets to learn the operator.

Because we have a large low-fidelity data set, it is straightforward to use a DeepONet (DeepONet$_\text{L}$) to learn $\mathcal{G}_L$ from $\mathcal{T}_L$. Here we develop two methods to learn $\mathcal{G}_H$, including residual learning and input augmentation.

2. Residual learning

Although the low-fidelity solution is not accurate enough, it captures the basic trend or shape of the solution. Instead of directly learning the high-fidelity output, we learn the residual between the high-fidelity and low-fidelity outputs:

$$\mathcal{R}(v)(\xi) = \mathcal{G}_H(v)(\xi) - \mathcal{G}_L(v)(\xi).$$

We use another DeepONet (DeepONet$_\text{H}$) to learn the residual operator $\mathcal{R}$. Hence, we have two DeepONets (DeepONet$_\text{L}$ and

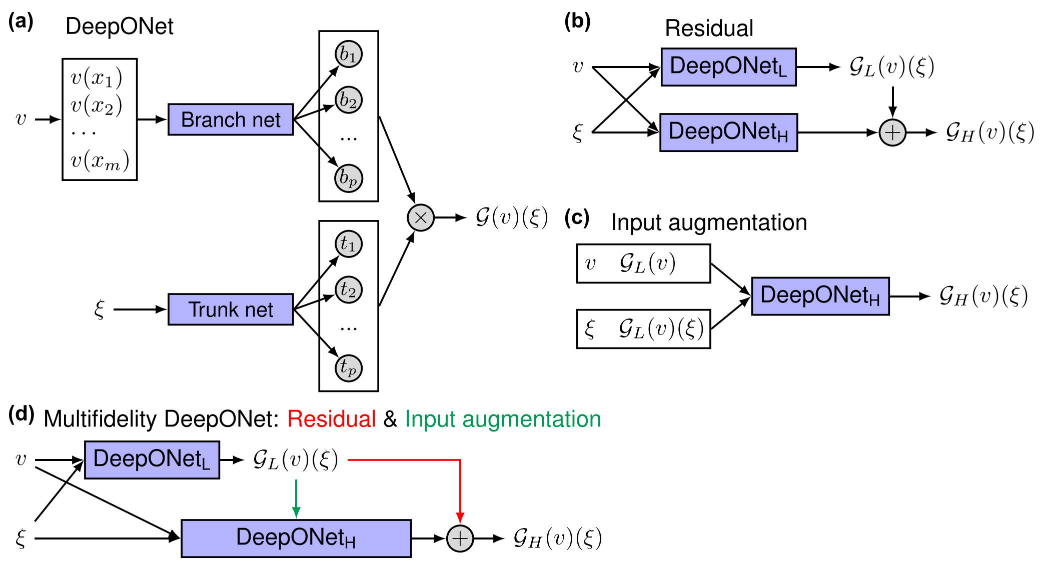

FIG. 1. Architectures of DeepONet and multifidelity DeepONet. (a) Architecture of DeepONet, adapted from [11]. (b) A DeepONet learns the residual between the high-fidelity and low-fidelity solutions. (c) The low-fidelity solution is used as the extra input of the DeepONet. (d) Multifidelity DeepONet uses the residual learning (the red line) and input augmentation (the green line).

DeepONet$_\mathrm{H}$) [Fig. 1(b)]. DeepONet$_\mathrm{L}$ is trained using $\mathcal{T}_L$, and DeepONet$_\mathrm{H}$ is trained using $\mathcal{T}_H$. During the preparation of this paper, a new paper [46] appeared, which also proposed a similar approach of residual learning for multifidelity DeepONets.

3. *Input augmentation*

For a same input function v, the low-fidelity solution $\mathcal{G}_L(v)$ and high-fidelity solution $\mathcal{G}_H(v)$ are usually highly correlated. In our residual learning above, therefore, we are exploiting the property that $\mathcal{G}_H(v)$ and $\mathcal{G}_L(v)$ exhibit similar behaviors, which should make their difference $\mathcal{R}$ easy to learn. We can also learn this correlation directly from data. Specifically, here we use the low-fidelity prediction as an additional input of DeepONet$_\mathrm{H}$. We can append the low-fidelity prediction to the input of the branch net and/or the trunk net [Fig. 1(c)]. We denote the operator learned by this DeepONet$_\mathrm{H}$ by $\mathcal{Q}$.

a. Approach I. We append the low-fidelity prediction $\mathcal{G}_L(v)$ (the entire function) to the branch net inputs:

$$\mathcal{G}_H(v)(\xi) = \mathcal{Q}(v, \mathcal{G}_L(v))(\xi),$$

i.e., the branch net has two functions v and $\mathcal{G}_L(v)$ is the input. In this study we directly concatenate the discretized v and $\mathcal{G}_L(v)$, i.e., the pointwise evaluations of v and $\mathcal{G}_L(v)$ are stacked together as the input of the branch net.

b. Approach II. Instead of using the entire function $\mathcal{G}_L(v)$, we can also use only the evaluation of $\mathcal{G}_L(v)$ at the location ξ to be predicted, i.e., $\mathcal{G}_L(v)(\xi)$. In this case we append $\mathcal{G}_L(v)(\xi)$ in the trunk-net inputs:

$$\mathcal{G}_H(v)(\xi) = \mathcal{Q}(v)(\xi, \mathcal{G}_L(v)(\xi)).$$

In our numerical experiments with the Poisson equation in Sec. III A, we find that Approach II achieves better accuracy than Approach I. One possible reason is that the input dimension of the DeepONet$_\mathrm{H}$ in Approach I is much higher than that in Approach II, and thus the DeepONet$_\mathrm{H}$ in Approach I is more difficult to train.

4. *Residual learning and input augmentation*

We also combine the methods of residual learning and input augmentation Approach II [Fig. 1(d)], where we have two DeepONets: DeepONet$_\mathrm{L}$ and DeepONet$_\mathrm{H}$. DeepONet$_\mathrm{L}$ learns the low-fidelity operator $\mathcal{G}_L$, and DeepONet$_\mathrm{H}$ learns the residual operator $\mathcal{R}$. DeepONet$_\mathrm{H}$ also uses $\mathcal{G}_L(v)(\xi)$ as one of its inputs of the trunk net. The final prediction of the high-fidelity solution is

$$\mathcal{G}_H(v)(\xi) = \underbrace{\mathcal{G}_L(v)(\xi)}_{\mathrm{DeepONet_L}} + \underbrace{\mathcal{R}(v)(\xi, \overbrace{\mathcal{G}_L(v)(\xi)}^{\mathrm{DeepONet_L}})}_{\mathrm{DeepONet_H}}. \qquad (1)$$

We can decompose the error of the multifidelity DeepONet $\mathcal{E}_{\mathcal{G}_H}$ by the triangle inequality as

$$\mathcal{E}_{\mathcal{G}_H} \leqslant \mathcal{E}_{\mathcal{G}_L} + \mathcal{E}_{\mathcal{R}}, \qquad (2)$$

where $\mathcal{E}_{\mathcal{G}_L}$ and $\mathcal{E}_{\mathcal{R}}$ are the errors of DeepONet$_\mathrm{L}$ and DeepONet$_\mathrm{H}$, respectively. Hence, in order to have an accurate multifidelity DeepONet surrogate for $\mathcal{G}_H$, both DeepONet$_\mathrm{L}$ and DeepONet$_\mathrm{H}$ should be trained well. We also note that the computational cost of training a multifidelity DeepONet is

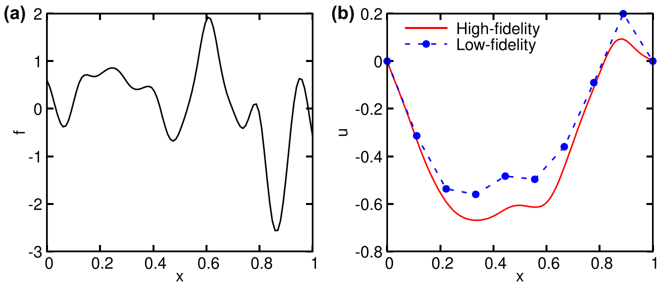

FIG. 2. Examples of the Poisson equation. (a) An example of randomly sampled f. (b) The corresponding low- and high-fidelity solutions.

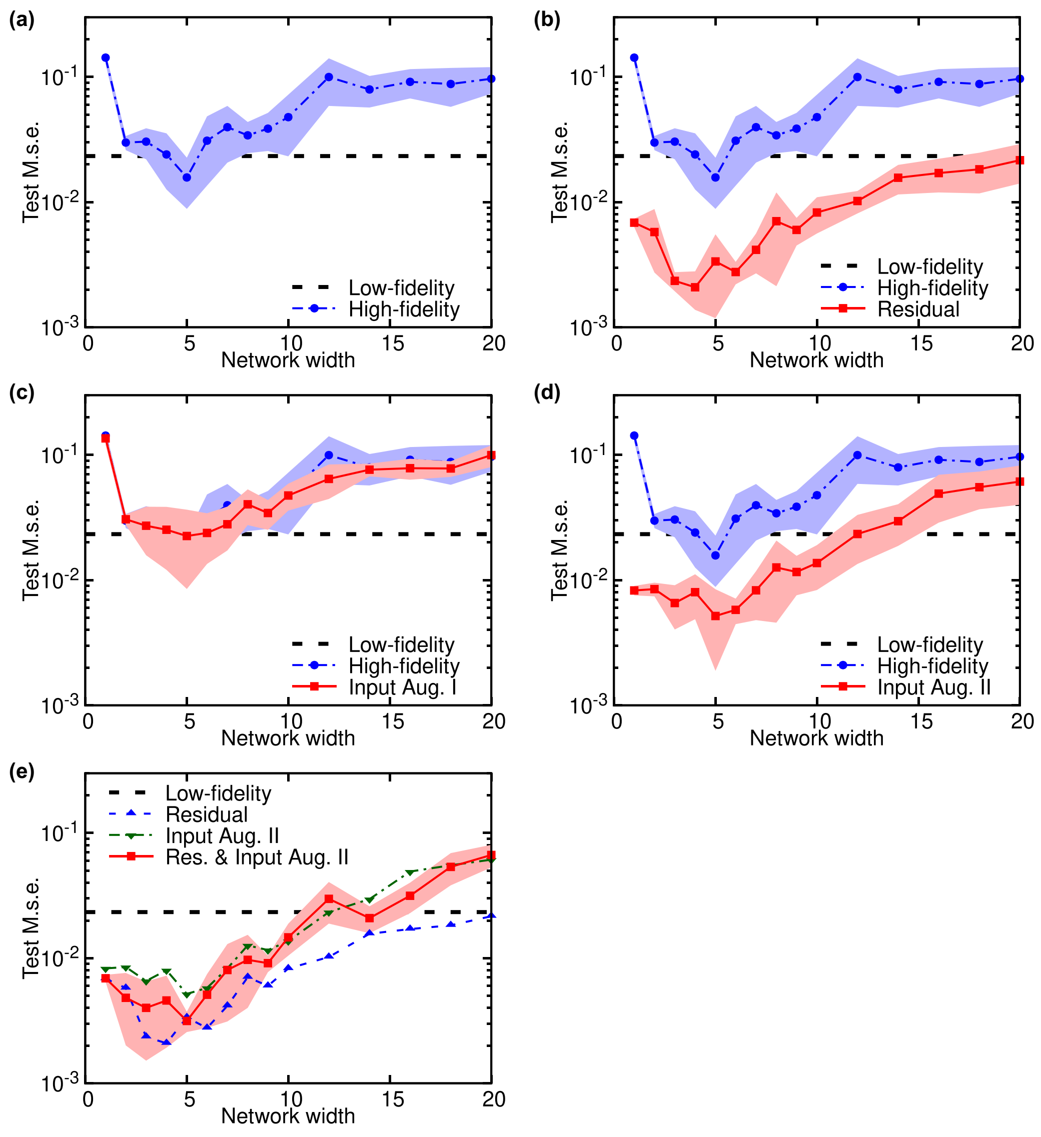

FIG. 3. Comparison of different multifidelity methods for DeepONet on the Poisson equation. (a) DeepONet trained only with an extremely small high-fidelity data set. (b) Multifidelity DeepONet with residual learning. (c) Multifidelity DeepONet with input augmentation Approach I. (d) Multifidelity DeepONet with input augmentation Approach II. (e) Multifidelity DeepONet with residual learning and input augmentation Approach II. The shaded regions represent the one standard deviation of ten runs with a randomly generated data set and random network initialization.

much higher than the cost of training a single DeepONoet only with the high-fidelity data set, as (1) a multifidelity DeepONet includes two DeepONet to be trained and (2) the low-fidelity data set is usually much larger than the high-fidelity data set. Once the multifidelity DeepONet is trained, it can be used repeatedly for inference of different inputs.

C. Inverse design

Once we have a surrogate model of the operator $\mathcal{G}$, we can predict $\mathcal{G}(v)$ for any v within a fraction of a second for most systems in practice, while the exact inference time depends on the specific network size and hardware configuration being used. One application is inverse design, which is formulated as an optimization problem:

$$\max_v \mathcal{J}(u; v),$$

subject to

$$u = \mathcal{G}(v).$$

We have the flexibility to choose the optimization algorithm.

1. Genetic algorithm

We can use a derivative-free optimization algorithm such as genetic algorithm (GA). In this study we consider the canonical genetic algorithm. The algorithm starts from a population of randomly generated candidate solutions (called individuals), and then the population is evolved toward better solutions. The population in each iteration is called a

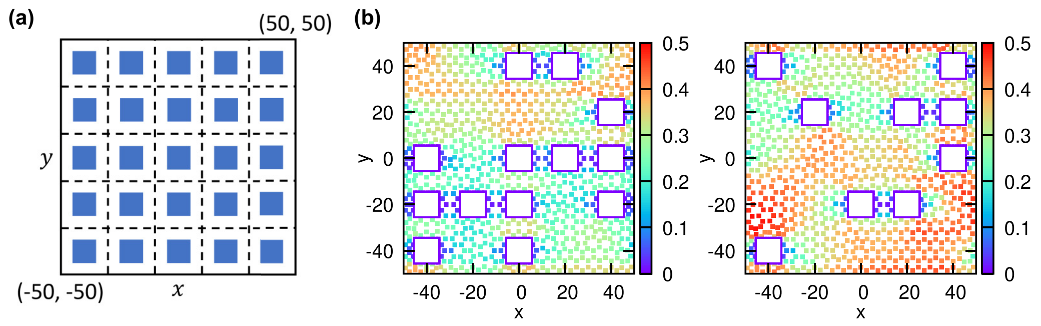

FIG. 4. BTE problem setup and two examples. (a) The computational domain and possible pore locations. (b) Two examples of the pore locations and the corresponding flux. The dots represent the locations of the mesh nodes.

generation. In each generation the individuals are selected and modified via the operations of mutation and crossover to form a new generation in the next iteration. The individuals with higher values of the objective function have a higher chance to be selected. We terminate the algorithm when a maximum number of generations has been produced. For more details of the algorithm, see Ref. [47]. We implement the algorithm using the library DEAP [48]. In this work we choose the population size as 100.

2. Gradient-based topology optimization

Because $\mathcal{G}$ is approximated by DeepONets in Eq. (1), we can compute the derivative of $\frac{\partial u(\xi)}{\partial v(x_i)}$ as

$$\frac{\partial \mathcal{G}_H(v)(\xi)}{\partial v(x_i)} = \frac{\partial \mathcal{R}(v)(\xi, \mathcal{G}_L(v)(\xi))}{\partial v(x_i)} + \frac{\partial \mathcal{G}_L(v)(\xi)}{\partial v(x_i)}\left(1 + \frac{\partial \mathcal{R}(v)(\xi, \mathcal{G}_L(v)(\xi))}{\partial \mathcal{G}_L(v)(\xi)}\right),$$

where the three partial derivatives on the right-hand side are computed by automatic differentiation (AD; also called "backpropagation"). Then we can compute $\frac{\partial \mathcal{J}}{\partial v(x_i)}$ by the chain rule and use gradient-based topology optimization (TO) methods.

In TO, each pixel of the domain is a degree of freedom, where the material property is iteratively updated using a gradient-based optimization algorithm, such as the conservative convex separable approximations algorithm [49]. In contrast to genetic algorithm, TO uses a continuous relaxation optimization scheme, which takes full advantage of the extension from binary inputs to real inputs of neural networks. Even though the data only has binary inputs, the neural network defines a differentiable continuous function which can return function evaluations and gradient evaluations anywhere between the binary inputs. Despite the fact that this continuation is artificial, i.e., nonbinary values do not correspond to any physical materials, we can use this framework to perform optimization as long as the end design only contains binary values.

In order to converge to a binary-input result, TO uses a smoothed approximation of the Heaviside function as a thresholding function, which pushes the design values towards binary values [50]:

$$\tilde{\xi}(\xi) = \frac{\tanh \beta\eta + \tanh \beta(\xi - \eta)}{\tanh \beta\eta + \tanh \beta(1 - \eta)}, \qquad (3)$$

where η is a hyperparameter to be fixed in [0,1], and $\beta \in [1, \infty]$ controls how binary the resulting design will be. The larger the β, the stiffer the optimization becomes. In TO, β starts at 1, and is successively doubled after each round of optimization, using the optimal design of the last round of optimization as a starting point to the next, until the optimum converges to binary values. Although the thresholding function is usually sufficient to converge to binary design, it is not guaranteed. In the case when the thresholding function is not sufficient to converge to a binary design, a penalty function $\bar{\xi}(\tilde{\xi}) = \tilde{\xi}(1 - \tilde{\xi})$ is added to the objective function to force the design to be binary.

III. RESULTS

We apply the proposed multifidelity DeepONet to learn the Poisson equation and Boltzmann transport equation. We then use the learned multifidelity DeepONet for inverse design of thermal transport in nanostructures. All the multifidelity DeepONet codes in this study are implemented by using the library DeepXDE [3] and will be deposited in GitHub at [51].

A. Poisson equation

We first consider a one-dimensional (1D) Poisson equation to compare the different multifidelity methods proposed

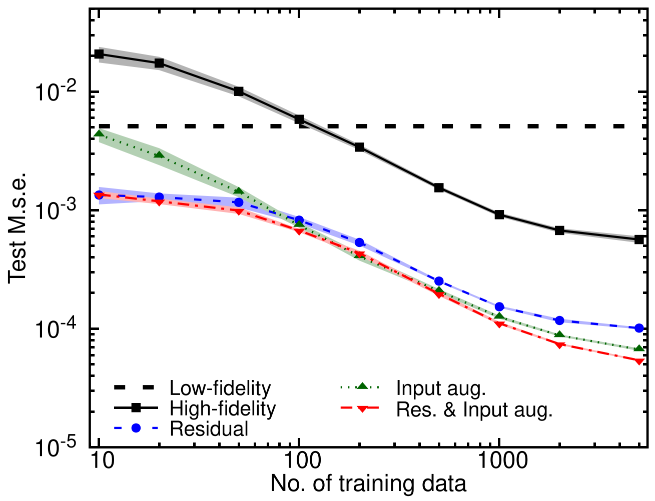

FIG. 5. Comparison of different multifidelity methods for DeepONet on BTE.

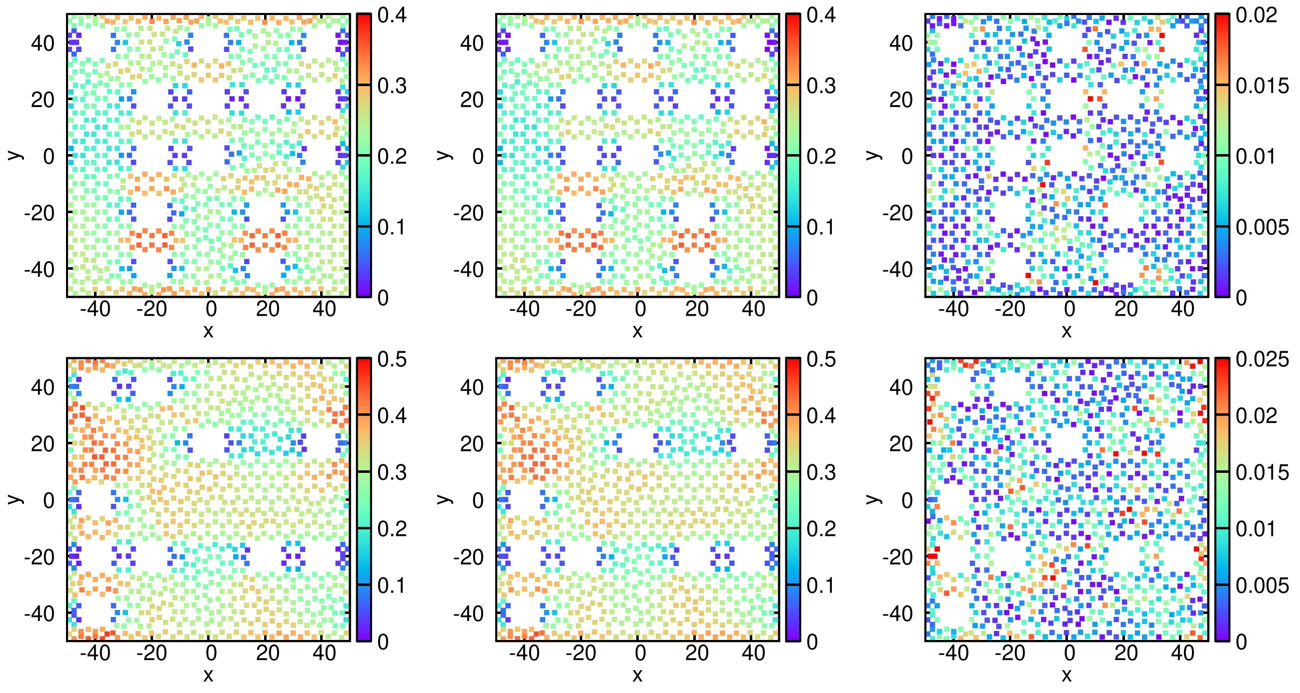

FIG. 6. Two examples of the predictions and pointwise absolute errors of multifidelity DeepONet for BTE. The results in the same row is for the same geometry of pore locations. (Left) Reference solutions. (Middle) Predictions of multifidelity DeepONet. (Right) Pointwise absolute errors.

in Sec. II B, and demonstrate the effectiveness of multifidelity DeepONet.

1. Problem setup

The 1D Poisson equation is described by

$$\frac{d^2u}{dx^2} = 20f(x), \quad x \in [0, 1],$$

with the Dirichlet boundary condition $u(0) = u(1) = 0$. We aim to learn the operator from the forcing term $f(x)$ to the PDE solution:

$$\mathcal{G} : f \mapsto u.$$

We sample f from a Gaussian random field (GRF) with mean zero:

$$f \sim \mathcal{GP}(0, k_l(x_1, x_2)),$$

where the covariance kernel $k_l(x_1, x_2) = \exp[-|x_1 - x_2|^2/(2l^2)]$ is the Gaussian kernel with a length-scale parameter $l = 0.05$. An example of randomly sampled f is shown in Fig. 2(a).

We generate the high-fidelity and low-fidelity data sets by solving the Poisson equation via the finite difference method with different mesh size Δx. For the high-fidelity solutions, we use $\Delta x = 1/99$, and for the low-fidelity solutions, $\Delta x = 1/9$. The low- and high-fidelity solutions for the random sample f above are shown in Fig. 2(b), which shows that the low-fidelity solution has a large error. When testing on a large data set, the mean squared error (mse) of the low-fidelity solver is 0.0233 (the black dashed lines in Fig. 3).

2. High-fidelity only

Instead of achieving high accuracy, we aim to compare the performance of different approaches of multifidelity DeepONet, and thus we generate an extremely small high-fidelity data set. In the high-fidelity data set we have 500 different samples of f, but for each f we do not have the full field observation of the corresponding solution u, and instead, we only know the value of $u(x)$ at one location x randomly sampled in [0, 1].

Because the data set is small, we choose both the trunk net and the branch net as shallow fully connected neural networks with the SELU activation function [52]. All DeepONets are trained with an Adam optimizer [53] with a learning rate 10^{-4} for 50 000 epochs. We test DeepONets with different width, and the DeepONet with the width 5 has the smallest test mse of 0.0157 ± 0.0069 [Fig. 3(a)]. The mean and standard deviation of the error are obtained by ten runs with a randomly generated data set and random network initialization. Larger DeepONets have the issue of overfitting. Because an extremely small data set has been used, the best DeepONet has similar accuracy as the low-fidelity solution.

3. Multifidelity

We use the same setup (the same high-fidelity data set and the same hyperparameters) to test different approaches of multifidelity DeepONet. Because a multifidelity DeepONet has a low-fidelity DeepONet (DeepONet$_\mathrm{L}$) and a high-fidelity DeepONet (DeepONet$_\mathrm{H}$), in order to remove the interference of DeepONet$_\mathrm{L}$, we directly use the low-fidelity solver to replace DeepONet$_\mathrm{L}$ in Figs. 1(b), 1(c), and 1(d), i.e., we assume that the network DeepONet$_\mathrm{L}$ is perfect. Because the low-fidelity solver can only predict the solution on the ten

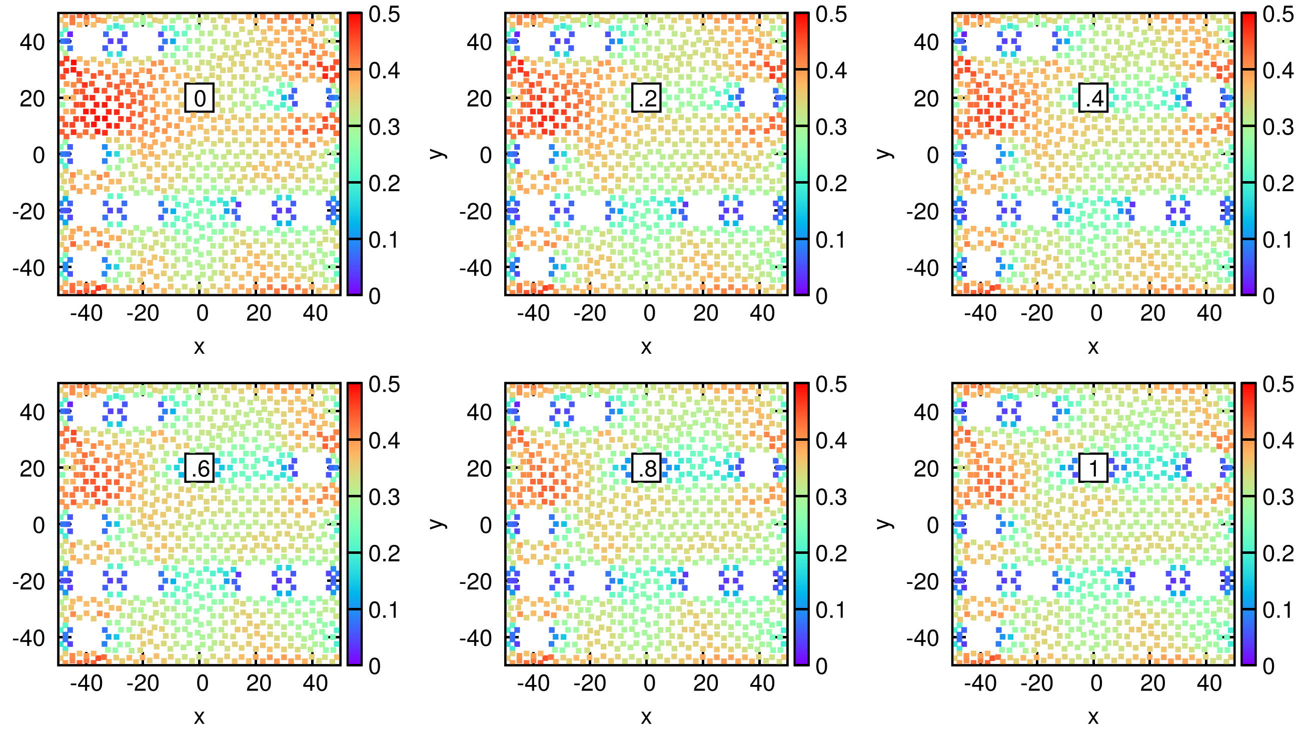

FIG. 7. Examples of predictions for $v_{2,3} \in [0, 1]$ by the multifidelity DeepONet. $v_{2,3}$ is equal to 0 (not pore), 0.2, 0.4, 0.6, 0.8, and 1 (pore).

locations as shown in Fig. 2(b), we perform a linear interpolation to compute the low-fidelity solution for other locations [e.g., the dashed line in Fig. 2(b)].

a. Residual learning. We first test the residual learning in Sec. II B 2. By learning the residual, the error of DeepONet is always about one order of magnitude smaller than the DeepONet with high-fidelity only [Fig. 3(b)] no matter what the network width is. The smallest error (0.0021 ± 0.0007) is obtained for the width 4.

b. Input augmentation. Similarly, we also test the two approaches of input augmentation in Sec. II B 3. We show that Approach I, i.e., using the entire low-fidelity solution as the branch net input, does not have any improvement [Fig. 3(c)]. For Approach II of only using the low-fidelity solution of the target point as the trunk net input, the smallest test error is 0.0052 ± 0.0033, also obtained at the width 5. Input augmentation Approach II is uniformly better than high-fidelity only [Fig. 3(d)].

c. Residual and input augmentation (Approach I). We further combine the residual learning and the input augmentation (Approach II). When the width is 5, the smallest test mse is 0.0031 ± 0.0006 [Fig. 3(e)].

We show that in this Poisson equation, residual learning and input augmentation Approach II improve the accuracy of DeepONet, but there is no improvement by combing them. However, as we show in the next section, by using both approaches together, we can achieve better accuracy.

B. Boltzmann transport equation

When the characteristic length of a semiconductor material approaches the mean-free-path (MFP) of heat carriers, i.e., phonons, Fourier's law breaks down [42]. Here we learn the nondiffusive heat transport with the steady-state phonon Boltzmann transport equation (BTE) by multifidelity DeepONet.

1. Problem setup

Under the relaxation time approximation, BTE is given by [42]

$$-\mathbf{v}_\mu \cdot \nabla f_\mu(\mathbf{r}) = \frac{f_\mu(\mathbf{r}) - f_\mu^0(\mathbf{r})}{\tau_\mu}, \qquad (4)$$

where $\mathbf{v}_\mu$ and τ_μ are the phonon group velocity and intrinsic scattering time, respectively. The label μ indicates both the phonon wave vector and polarization. The unknown of Eq. (4) is the nonequilibrium phonon distribution $f_\mu(\mathbf{r})$. Both the group velocities and the scattering times, computed with density functional theory (see Ref. [54] for its application to phonon transport), are obtained by AlmaBTE [55] using a wave-vector discretization of $24 \times 24 \times 24$. The equilibrium distribution $f_\mu^0(\mathbf{r})$, under the assumption of small temperature variation, is given by

$$f_\mu^0(\mathbf{r}) = \sum_{\mu'} \alpha_{\mu'} f_{\mu'}(\mathbf{r}),$$

where $\alpha_{\mu'} = C_{\mu'}/\tau_{\mu'}(\sum_{\mu''} C_{\mu''}/\tau_{\mu''})^{-1}$; the terms C_μ and τ_μ are the mode-resolved heat capacity and scattering time, respectively [55].

We consider BTE defined on a rectangular domain (in nm) $\Omega = [-50, 50] \times [-50, 50]$ with periodic boundary conditions in both x and y directions, and a temperature difference of 1 K is applied along the x axis. Inside the domain we have a five by five equispaced array of small squares [the blue squares in Fig. 4(a)]. Each square is of size (10, 10) and can be a pore

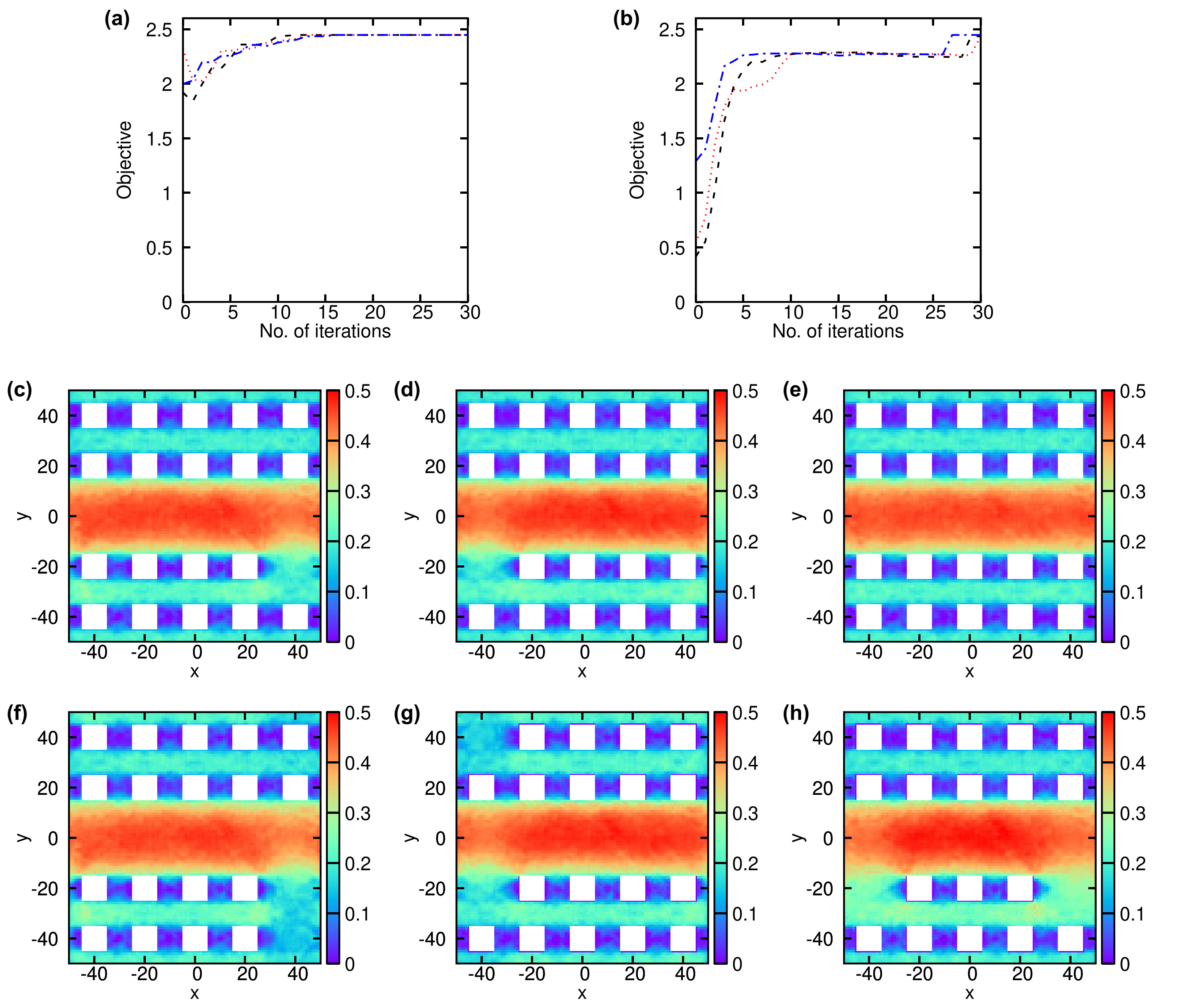

FIG. 8. Maximize the normalized heat flux at the center via GA and TO in Sec. III C 2. (a) The optimization process of GA. (b) The optimization process of TO. (c)–(h) The best six designs.

without any material. Along the wall of the pores we adopt the totally diffuse boundary conditions [56].

We solve Eq. (4) by the finite-volume method implemented in the free/open-source software OpenBTE [57], which adopts the source iteration scheme [56], i.e.,

$$\tau_\mu \mathbf{v}_\mu \cdot \nabla f_\mu^{(n)}(\mathbf{r}) + f_\mu^{(n)}(\mathbf{r}) = \sum_{\mu'} a_{\mu'} f_{\mu'}^{(n-1)}(\mathbf{r}). \qquad (5)$$

Here we are interested in the heat flux $\mathbf{J} = \frac{1}{\mathcal{V}}\sum_\mu C_\mu \mathbf{v}_\mu f_\mu$ ($\mathcal{V}$ being a normalization volume), and aim to learn the flux for different locations of pores. We generate the data set by randomly sampling the locations of pores. For a different number and locations of the pores, we have different BTE solutions. Each data point in the data set is a pair of the pore locations and the corresponding solution of flux in the mesh nodes. There are in total $2^{25} \approx 3.4 \times 10^7$ possibilities, and two examples of the random pore locations and the corresponding flux is shown in Fig. 4(b). Different geometry has different mesh, and the mesh nodes of the two examples are the dots in Fig. 4(b). On average, each mesh has 902 nodes. The high-fidelity solutions are obtained by Eq. (5) solved for five iterations, while the low-fidelity solutions are obtained with only two iterations.

In DeepONet, the branch net input $v = (v_{1,1}, v_{1,2}, \dots, v_{5,5})$ represents the locations of pores. If $v_{i,j} = 1$, then it is a pore, otherwise it is not a pore. We note that when training DeepONet, we scale the x and y coordinates from $[-50, 50]$ to $[-1, 1]$ and also normalize the network output to zero mean and unit variance.

2. Effectiveness of multifidelity learning

We use both the low-fidelity solver and DeepONet only trained with high-fidelity data as the two baselines. Compared to the high-fidelity solution, the low-fidelity solution has mse of 5.09×10^{-3} (the horizontal dashed line in Fig. 5). We train a DeepONet with only high-fidelity solutions, where the

TABLE I. Maximize the normalized heat flux at the center via GA and TO in Sect. III C 2. The objective values of the found designs (Fig. 8) computed by the multifidelity DeepONet and the numerical solver. Bold font indicates the highest objective value.

Design	Multifidelity DeepONet	Numerical solver	$\frac{\lvert\mathcal{J}-\mathcal{J}_{\text{opt}}\rvert}{\mathcal{J}_{\text{opt}}}$
Fig. 8(c)	2.447	2.306	1.58%
Fig. 8(d)	2.440	2.292	2.18%
Fig. 8(e)	2.435	**2.343**	Optimal
Fig. 8(f)	2.425	2.233	4.69%
Fig. 8(g)	2.414	2.235	4.61%
Fig. 8(h)	2.412	2.211	5.63%

branch net and the trunk net use ReLU activation function and have four and five layers, respectively. DeepONet is trained with an Adam optimizer [53] with a learning rate 10^{-4} for 100 000 epochs. The DeepONet needs about 150 training data to achieve similar accuracy of the low-fidelity solver (the solid black line in Fig. 5).

We then compare different approaches of multifidelity DeepONet. As discussed in Sec. III A 3, for better comparison, here we still use the low-fidelity solver to replace DeepONet$_L$. In the same setup, both residual learning and input augmentation Approach II improve the accuracy by one order of magnitude (Fig. 5). When the data set is very small (< 100), residual learning is better than input augmentation, otherwise input augmentation becomes better. When we use both approaches together, we have even better accuracy.

3. *Multifidelity DeepONet*

We have demonstrated the effectiveness of multifidelity DeepONet. As we mentioned above, we directly used the exact low-fidelity solver, and here we first train a

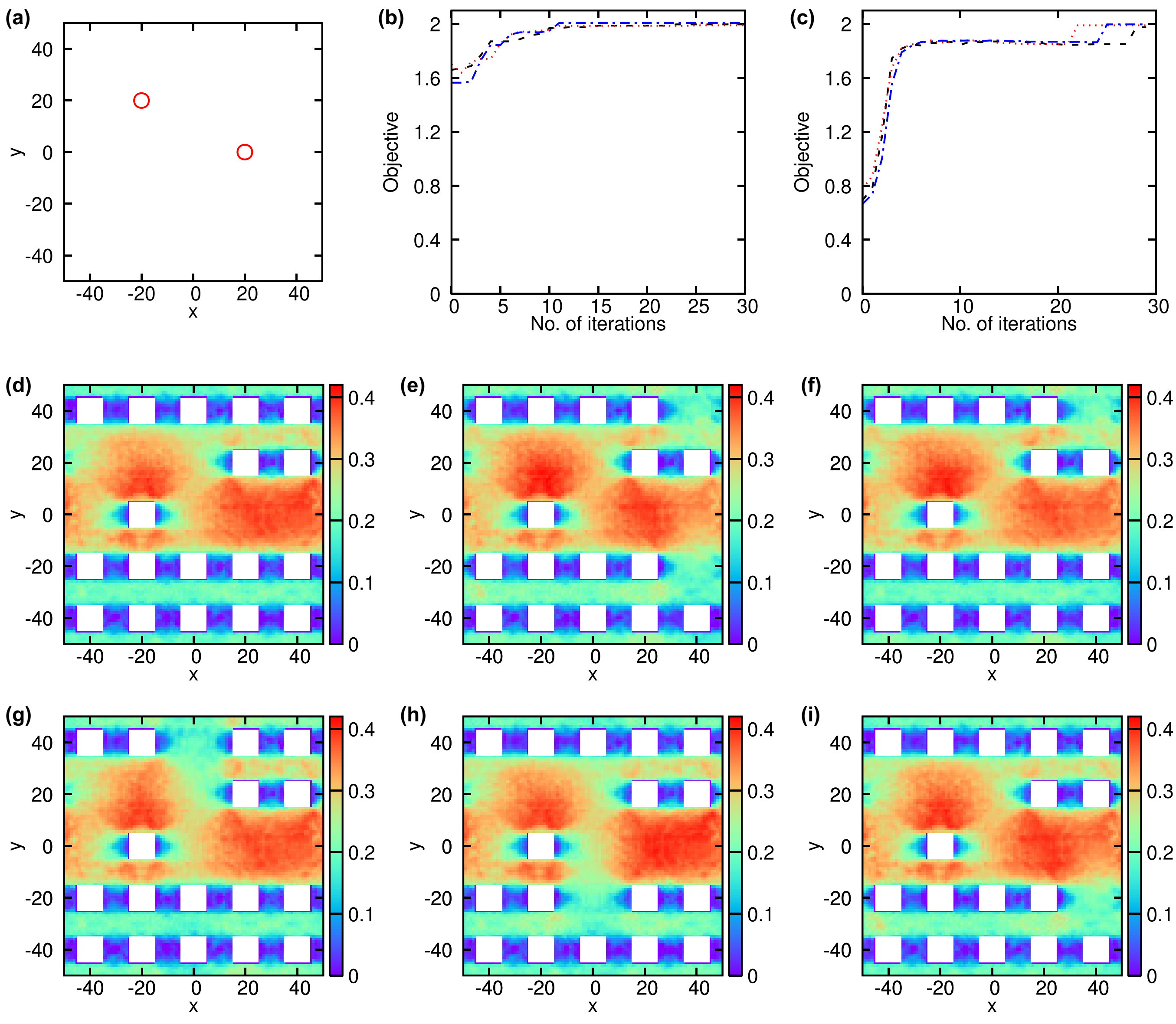

FIG. 9. Maximize the normalized heat flux on multiple points in Sec. III C 3. (a) The locations of the points to be optimized. (b) The optimization process of GA. (c) The optimization process of TO. (d)–(i) The best six designs.

TABLE II. Maximize the normalized heat flux on multiple points in Sec. III C 3. The objective values of the designs (Fig. 9) computed by the multifidelity DeepONet and the numerical solver. Bold font indicates the highest objective value.

Design	Multifidelity DeepONet	Numerical solver	$\frac{\lvert\mathcal{J}-\mathcal{J}_{\text{opt}}\rvert}{\mathcal{J}_{\text{opt}}}$
Fig. 9(d)	2.008	**1.987**	Optimal
Fig. 9(e)	1.996	1.928	2.97%
Fig. 9(f)	1.993	1.941	2.32%
Fig. 9(g)	1.990	1.961	1.31%
Fig. 9(h)	1.990	1.951	1.81%
Fig. 9(i)	1.990	1.964	1.16%

low-fidelity DeepONet (DeepONet$_\text{L}$) and then combine it with DeepONet$_\text{H}$ to have the multifidelity DeepONet.

a. Low-fidelity DeepONet. DeepONet$_\text{L}$ is trained with a data set of size 10 000 solutions. We perform a hyperparameter tuning and choose the branch net as three layers and the trunk net as five layers. Both the branch net and the trunk net use ReLU activation function and have width 512. After training with an Adam optimizer [53] with a learning rate 10^{-4} for 500 000 epochs, the testing mse is $4.61 \pm 0.21 \times 10^{-5}$, and the L^2 relative error is $1.91 \pm 0.02\%$.

The flux satisfies the periodic boundary conditions in x and y directions, and as proposed in Ref. [13], we can enforce the periodic BCs in DeepONet by adding a Fourier feature layer. Specifically, we use the basis functions of the Fourier series on a 2D square as the first layer of the trunk net. By enforcing PBC, test mse decreases to $3.90 \pm 0.15 \times 10^{-5}$, and L^2 relative error becomes $1.72 \pm 0.02\%$. The training of DeepONet$_\text{L}$ took 5.6 h by using a NVIDIA A40 GPU.

b. Multifidelity DeepONet. To train the high-fidelity DeepONet, we only use a data set of size 1000. We use the same setup for DeepONet$_\text{H}$ as the DeepONet$_\text{L}$, except that the branch net is chosen as a shallow network. The training of DeepONet$_\text{H}$ took 3.6 h by using a NVIDIA A40 GPU. Then the testing mse of the multifidelity DeepONet is $8.89 \pm 0.07 \times 10^{-5}$, and the L^2 relative error is $3.34 \pm 0.01\%$. Furthermore, to verify Eq. (2), we replace DeepONet$_\text{L}$ with the exact low-fidelity solver, then the testing mse of DeepONet$_\text{H}$ is $6.00 \pm 0.04 \times 10^{-5}$ and L^2 relative error is $2.72 \pm 0.01\%$. Hence we have

$$\underbrace{3.34 \pm 0.01\%}_{\mathcal{E}_{\mathcal{G}_H}} < \underbrace{1.72 \pm 0.02\%}_{\mathcal{E}_{\mathcal{G}_L}} + \underbrace{2.72 \pm 0.01\%}_{\mathcal{E}_{\mathcal{R}}}.$$

TABLE III. Maximize the normalized heat flux on multiple points with a constraint on the number of pores in Sec. III C 4. The objective values of the designs in Fig. 10 computed by the multifidelity DeepONet and the numerical solver.

N	Multifidelity DeepONet	Numerical solver
No constraint	2.008	1.987
11	1.879	1.857
9	1.820	1.763
7	1.678	1.639

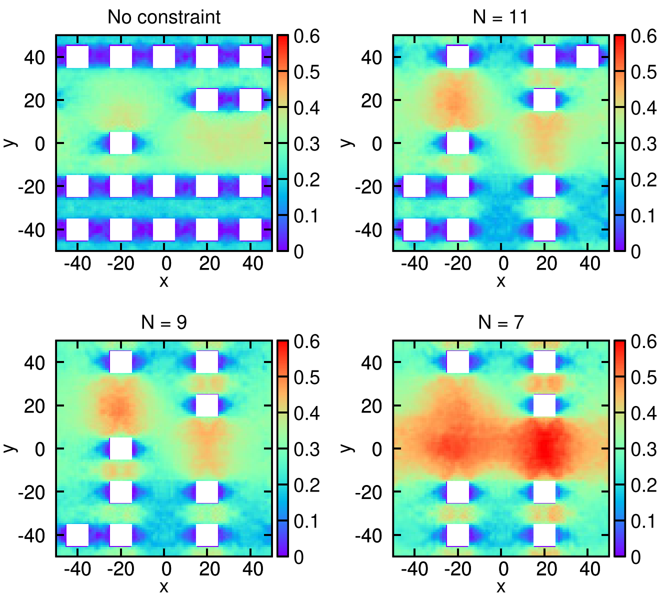

FIG. 10. Maximize the normalized heat flux on multiple points with a constraint on the number of pores in Sec. III C 4. The best designs for different values of N.

Two examples of the predictions and pointwise absolute errors of multifidelity DeepONet are shown in Fig. 6.

C. Boltzmann transport equation for inverse design

We have trained a multifidelity DeepONet as a surrogate model of BTE. Next we will use it for inverse design of thermal transport by optimizing an objective function $\mathcal{J}$:

$$\max_v \mathcal{J}(u).$$

Because $v_{i,j}$ is a binary value, this is a discrete optimization problem, and thus we can use algorithms like the genetic algorithm. Here we also consider to optimize the continuous relaxation via topology optimization. In this section we consider several different objective functions, and we note that the same surrogate model allows for using different objective functions and constraints without retraining.

1. Rationality of multifidelity DeepONet for continuous optimization algorithms

In topology optimization we treat v_{ij} as a continuous variable defined on [0, 1]. Our surrogate model has the capability to predict for any value of $v_{i,j} \in \mathbb{R}$, but we only trained the network with $v_{i,j} \in \{0, 1\}$, and thus there is no guarantee that the trained network can predict reasonable values for $v_{i,j} \in (0, 1)$. Hence, in order to use it, we need to check whether the prediction of the trained multifidelity DeepONet for $v_{i,j} \in (0, 1)$ is reasonable.

As an example, we take a random geometry shown in Fig. 7, and the value of $v_{2,3}$ changes from 0 to 1 gradually. When $v_{2,3}$ is closer to 1, the flux nearby becomes closer to 0. Hence, DeepONet makes reasonable predictions for intermediate value of v_{ij}. However, we note that in general there is no guarantee for the prediction, because we do not have training data in this domain.

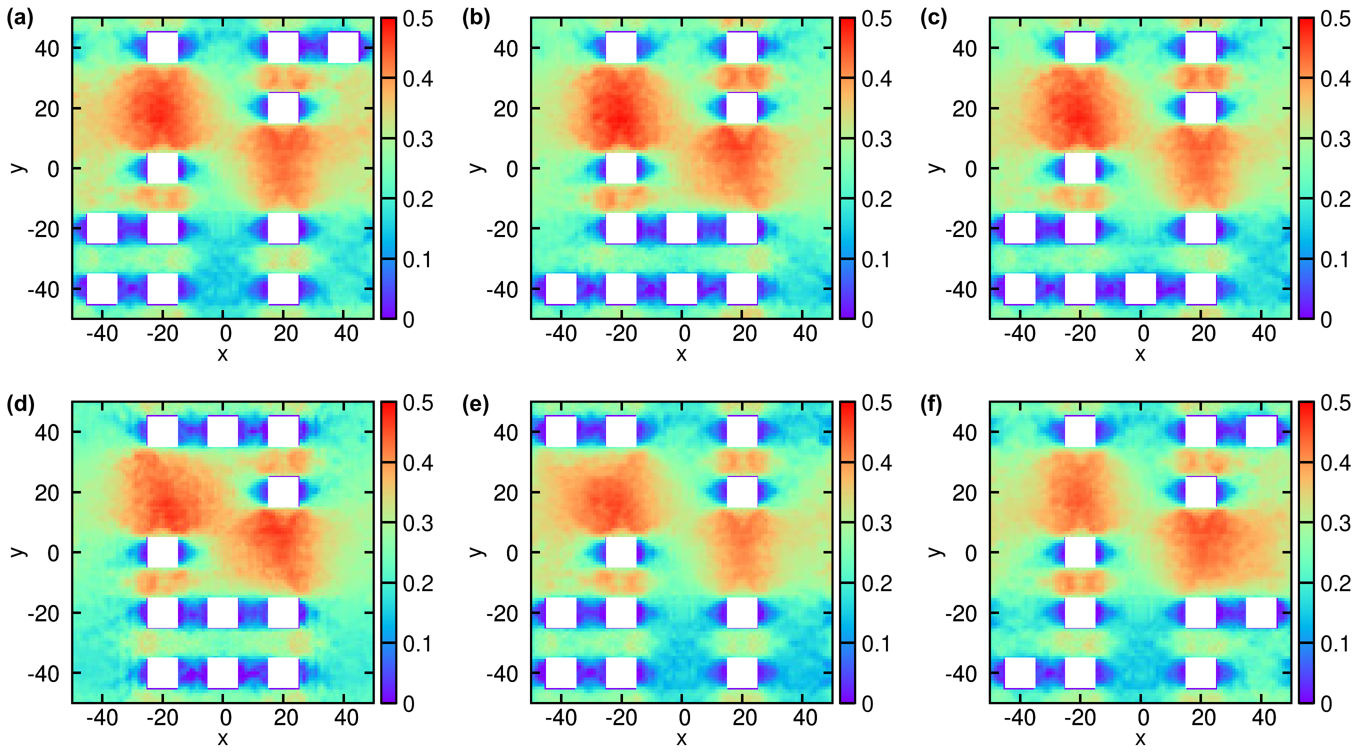

FIG. 11. Maximize the normalized heat flux on multiple points with a constraint on the number of pores in Sec. III C 4. The best six designs for $N = 11$.

2. Maximizing normalized heat flux at a given location

In the first inverse design problem, we consider to maximize the normalized heat flux at a given grid location. Here we consider the center location, i.e.,

$$\mathcal{J}(u) = \frac{u(0,0)}{\bar{u}}, \tag{6}$$

where

$$\bar{u} = \frac{1}{\text{Area}(\Omega)} \int_{\Omega} u(x,y)dxdy \tag{7}$$

is the average value of u. To compute the integral in $\bar{u}$, we use the midpoint rule as

$$\bar{u} \approx \frac{1}{N^2} \sum_{i=1}^{N^2} u(x_i, y_i),$$

where $\{(x_i, y_i)\}_{i=1}^{N^2}$ are the midpoints of an equispaced mesh of N by N, and we choose $N = 100$. We note that we can also use other numerical integration methods such as Gaussian quadrature, which has a high convergence rate for smooth functions, but in this problem, Gaussian quadrature has a slow convergence, because we use ReLU activation and the network has a bad smoothness.

We applied both GA and TO described in Sec. II C to solve this problem. Three examples of the values of the objective function during the iterative process of GA and TO are shown in Figs. 8(a) and 8(b), respectively, and they achieved similar final objective values of ~2.4. The abrupt jumps in the optimization curves for TO correspond to the final evaluation of the fully binarized structure, which happens once the optimization has converged.

Because the learned multifidelity DeepONet has an error of 3.34% as shown in Sec. III B 3, in general the best design based on the multifidelity DeepONet may not be the best design according to the high-fidelity numerical solver. Hence, instead of only using the best design from GA or TO, we need to output several top candidates and then verify them using the numerical solver. Here we consider the top six candidate designs [Figs. 8(c) to 8(h)].

For each design in Fig. 8, the objective values computed by the multifidelity DeepONet are listed in Table I. We also verify the objective values by using the high-fidelity numerical solver. Among these designs, the best one is the design in Fig. 8(e) with the optimal objective $\mathcal{J}_{\text{opt}} \approx 2.343$. We also compute the relative difference of other designs compared to this best design $\frac{|\mathcal{J}-\mathcal{J}_{\text{opt}}|}{\mathcal{J}_{\text{opt}}}$. The best design [Fig. 8(e)] is the third best based on the multifidelity DeepONet, but the best and

TABLE IV. Maximize the normalized heat flux on multiple points with a constraint on the number of pores in Sec. III C 4. The objective values of the designs (Fig. 11) computed by the multifidelity DeepONet and the numerical solver. Bold font indicates the highest objective value.

Design	Multifidelity DeepONet	Numerical solver	$\frac{\lvert\mathcal{J}-\mathcal{J}_{\text{opt}}\rvert}{\mathcal{J}_{\text{opt}}}$
Fig. 11(a)	1.879	**1.857**	Optimal
Fig. 11(b)	1.879	1.815	2.26%
Fig. 11(c)	1.873	1.840	0.92%
Fig. 11(d)	1.872	1.815	2.26%
Fig. 11(e)	1.869	1.826	1.67%
Fig. 11(f)	1.866	1.826	1.67%

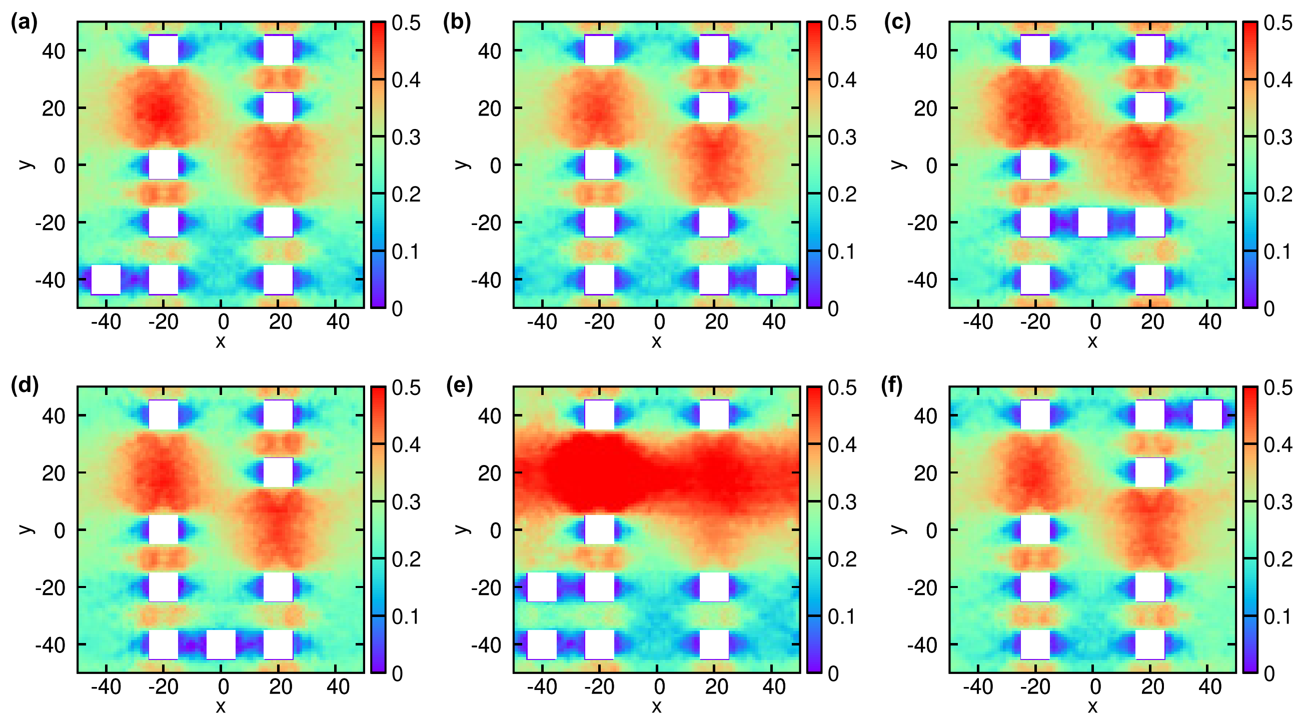

FIG. 12. Maximize the normalized heat flux on multiple points with a constraint on the number of pores in Sec. III C 4. The best 6 designs for $N = 9$.

second best designs [Figs. 8(c) and 8(d)] are only 1.58% and 2.18% worse, respectively, which is consistent with the error of the multifidelity DeepONet.

3. Maximizing normalized heat flux on multiple points

We have considered the objective for a single location, and here we consider the objective for multiple points. Specifically, we consider two point locations: $\mathbf{x}_1 = (-20, 20)$ and $\mathbf{x}_2 = (20, 0)$ [the red circles in Fig. 9(a)], and the objective function is

$$\mathcal{J} = \frac{1}{2}\sum_{i=1}^{2}\frac{u(\mathbf{x}_i)}{\bar{u}},$$

where $\bar{u}$ is defined in Eq. (7).

GA and TO achieved similar objective values of ~ 2 [Figs. 9(b) and 9(c)], and we show the top six candidate designs [Figs. 9(d) to 9(i)]. For each design, the objective values computed by the multifidelity DeepONet and the numerical solver are listed in Table II. Among these designs, the best design [Fig. 9(d)] found by the multifidelity DeepONet is indeed the best with the optimal objective $\mathcal{J}_{\text{opt}} \approx 1.987$. We also compute the relative difference of other designs compared to this design, and most of the relative differences are within 3% (Table II).

TABLE V. Maximize the normalized heat flux on multiple points with a constraint on the number of pores in Sec. III C 4. The objective values of the designs (Fig. 12) computed by the multifidelity DeepONet and the numerical solver. Bold font indicates the highest objective value.

Design	Multifidelity DeepONet	Numerical solver	$\frac{\lvert\mathcal{J}-\mathcal{J}_{\text{opt}}\rvert}{\mathcal{J}_{\text{opt}}}$
Fig. 12(a)	1.820	**1.763**	Optimal
Fig. 12(b)	1.810	1.760	0.17%
Fig. 12(c)	1.798	1.747	0.91%
Fig. 12(d)	1.798	1.759	0.23%
Fig. 12(e)	1.797	1.711	2.95%
Fig. 12(f)	1.788	1.758	0.28%

4. Maximizing normalized heat flux on multiple points with a constraint on the number of pores

In Sec. III C 3, the best design is the one in Fig. 9(d), and the results show that if we add additional pores, then the objective becomes smaller. Here we aim to investigate the best design given an extra constraint on the number of pores. Specifically, we require that

$$\text{No. of pores} \leqslant N$$

for a given integer N.

To consider this constraint, for GA, we simply return 0 as the objective if the design does not satisfy the constraint. For TO, the implementation is less straightforward, because the objective function needs to stay differentiable. We discuss how to implement the constraint in TO as follows. We implemented the constraint by adding a differentiable nonlinear constraint on the sum of the projected density. Note that since the constraint is on the projected function using Eq. (3), the density is not fully binarized. In practice we constrained the sum to be less than $N - 0.5$, which resulted in designs with less than N pores most of the time.

Here we consider three cases ($N = 11$, $N = 9$, and $N = 7$) and follow the same procedure as in Sec. III C 3 to find the best design for each case. For $N = 11$, the best six designs

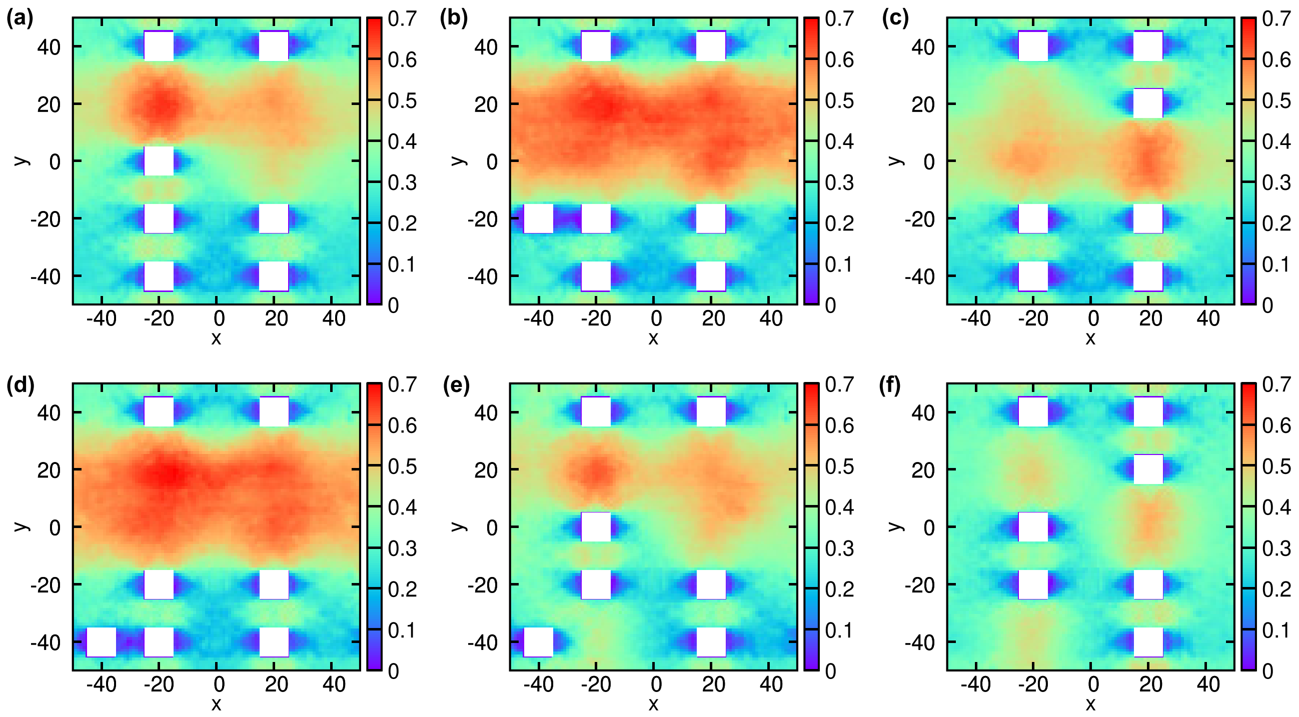

FIG. 13. Maximize the normalized heat flux on multiple points with a constraint on the number of pores in Sec. III C 4. The best six designs for $N = 7$.

and the corresponding objective values are in Fig. 11 and Table IV, respectively. For $N = 9$, the best six designs and the corresponding objective values are in Fig. 12 and Table V, respectively. For $N = 7$, the best six designs and the corresponding objective values are in Fig. 13 and Table VI, respectively.

The best designs for all the cases are listed in Fig. 10. It is interesting that the pore locations of the best design for $N = 11$ is a subset of the pore locations for the case without any constraint. This is also true for the pair of $N = 11$ and $N = 9$, and the pair of $N = 9$ and $N = 7$. For the value of the objective function, a stronger constraint on the number of pores makes the objective value smaller (Table III).

IV. CONCLUSION

In this study we developed a version of DeepONet for learning efficiently from multifidelity data sets. A multifidelity DeepONet includes two independent DeepONets coupled by residual learning and input augmentation. We demonstrated that, compared to single-fidelity DeepONet, multifidelity DeepONet achieves one order of magnitude smaller error when using the same amount of high-fidelity data. By combining multifidelity DeepONet with genetic algorithms or topology optimization, we can reuse the trained network for fast inverse design of multiple objective functions.

We have introduced two approaches of input augmentation, where Approach I uses the entire function of $\mathcal{G}_L(v)$, and Approach II uses only the low-fidelity prediction at one point $\mathcal{G}_L(v)(\xi)$. In this study we find that Approach II is more accurate than Approach I, but there are also other possibilities that might be fruitful to explore: for example, using the low-fidelity solution in a local domain $[\xi - \Delta\xi, \xi + \Delta\xi]$. Moreover, the two sub-DeepONets used in the multifidelity DeepONet are "vanilla" DeepONets. Several extensions of DeepONet have been developed very recently, as discussed in the Introduction, and these extensions may further improve the performance of multifidelity DeepONet.

TABLE VI. Maximize the normalized heat flux on multiple points with a constraint on the number of pores in Sec. III C 4. The objective values of the designs (Fig. 13) computed by the multifidelity DeepONet and the numerical solver. Bold font indicates the highest objective value.

Design	Multifidelity DeepONet	Numerical solver	$\frac{\lvert\mathcal{J}-\mathcal{J}_{\text{opt}}\rvert}{\mathcal{J}_{\text{opt}}}$
Fig. 13(a)	1.708	1.633	0.366%
Fig. 13(b)	1.680	1.572	4.087%
Fig. 13(c)	1.678	**1.639**	Optimal
Fig. 13(d)	1.677	1.568	4.331%
Fig. 13(e)	1.677	1.556	5.064%
Fig. 13(f)	1.666	1.572	4.087%

ACKNOWLEDGMENTS

This work was partially supported by MIT-IBM Watson AI Laboratory (Challenge No. 2415). R.P. is also supported in part by the U.S. Army Research Office through the Institute for Soldier Nanotechnologies at MIT under Awards No. W911NF-18-2-0048 and No. W911NF-13-D-0001.

APPENDIX: BEST DESIGNS IN SEC. III C 4

For $N = 11$, the best 6 designs and the corresponding objective values are in Fig. 11 and Table IV, respectively. For

$N = 9$, the best 6 designs and the corresponding objective values are in Fig. 12 and Table V, respectively. For $N = 7$, the best 6 designs and the corresponding objective values are in Fig. 13 and Table VI, respectively.

[1] G. Em Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, Physics-informed machine learning, Nat. Rev. Phys. **3**, 422 (2021).

[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. **378**, 686 (2019).

[3] L. Lu, X. Meng, Z. Mao, and G. Em Karniadakis, Deepxde: A deep learning library for solving differential equations, SIAM Rev. **63**, 208 (2021).

[4] G. Pang, L. Lu, and G. Em Karniadakis, FPINNs: Fractional physics-informed neural networks, SIAM J. Sci. Comput. **41**, A2603 (2019).

[5] D. Zhang, L. Lu, L. Guo, and G. Em Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, J. Comput. Phys. **397**, 108850 (2019).

[6] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, Physics-informed neural networks with hard constraints for inverse design, SIAM J. Sci. Comput. **43**, B1105 (2021).

[7] J. Yu, L. Lu, X. Meng, and G. Em Karniadakis, Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems, Computer Methods in Applied Mechanics and Engineering **393**, 114823 (2022).

[8] Y. Chen, L. Lu, G. Em Karniadakis, and L. D. Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express **28**, 11618 (2020).

[9] A. Yazdani, L. Lu, M. Raissi, and G. Em Karniadakis, Systems biology informed deep learning for inferring parameters and hidden dynamics, PLoS Comput. Biol. **16**, e1007575 (2020).

[10] M. Daneker, Z. Zhang, G. Em Karniadakis, and L. Lu, Systems biology: Identifiability analysis and parameter identification via systems-biology informed neural networks, arXiv:2202.01723.

[11] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. Em Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. **3**, 218 (2021).

[12] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv:2010.08895.

[13] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. Em Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data, Computer Methods in Applied Mechanics and Engineering **393**, 114778 (2022).

[14] H. You, Y. Yu, M. D'Elia, T. Gao, and S. Silling, Nonlocal kernel network (NKN): A stable and resolution-independent deep neural network, arXiv:2201.02217.

[15] N. Trask, R. G. Patel, B. J. Gross, and P. J. Atzberger, Gmls-nets: A framework for learning from unstructured data, arXiv:1909.05371.

[16] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, arXiv:2003.03485.

[17] R. G. Patel, N. A. Trask, M. A. Wood, and E. C. Cyr, A physics-informed operator regression framework for extracting data-driven continuum models, Comput. Methods Appl. Mech. Eng. **373**, 113500 (2021).

[18] T. Chen and H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Networks **6**, 911 (1995).

[19] P. Jin, S. Meng, and L. Lu, Mionet: Learning multiple-input operators via tensor product, arXiv:2202.06137.

[20] S. Cai, Z. Wang, L. Lu, T. A. Zaki, and G. Em Karniadakis, DeepM&MNet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks, J. Comput. Phys. **436**, 110296 (2021).

[21] Z. Mao, L. Lu, O. Marxen, T. A. Zaki, and G. Em Karniadakis, DeepM&MNet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators, J. Comput. Phys. **447**, 110698 (2021).

[22] G. Lin, C. Moya, and Z. Zhang, Accelerated replica exchange stochastic gradient langevin diffusion enhanced Bayesian DeepONet for solving noisy parametric PDEs, arXiv:2111.02484.

[23] C. Moya, S. Zhang, M. Yue, and G. Lin, Deeponet-grid-uq: A trustworthy deep operator framework for predicting the power grid's post-fault trajectories, arXiv:2202.07176.

[24] Y. Yang, G. Kissas, and P. Perdikaris, Scalable uncertainty quantification for deep operator networks using randomized priors, arXiv:2203.03048.

[25] L. Liu and W. Cai, Multiscale deeponet for nonlinear operators in oscillatory function spaces for building seismic wave responses, arXiv:2111.04860.

[26] L. Liu and W. Cai, Deeppropnet—A recursive deep propagator neural network for learning evolution PDE operators, arXiv:2202.13429.

[27] S. Goswami, M. Yin, Y. Yu, and G. Em Karniadakis, A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials, Comput. Methods Appl. Mech. Eng. **391**, 114587 (2022).

[28] S. Wang, H. Wang, and P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets, Sci. Adv. **7**, eabi8605 (2021).

[29] P. C. D. Leoni, L. Lu, C. Meneveau, G. Karniadakis, and T. A. Zaki, DeepONet prediction of linear instability waves in high-speed boundary layers, arXiv:2105.08697.

[30] C. Lin, Z. Li, L. Lu, S. Cai, M. Maxey, and G. Em Karniadakis, Operator learning for predicting multiscale bubble growth dynamics, J. Chem. Phys. **154**, 104118 (2021).

[31] C. Lin, M. Maxey, Z. Li, and G. Em Karniadakis, A seamless multiscale operator neural network for inferring bubble dynamics, J. Fluid Mech. **929**, A18 (2021).

[32] J. D. Osorio, Z. Wang, G. Karniadakis, S. Cai, C. Chryssostomidis, M. Panwar, and R. Hovsapian, Forecasting solar-thermal systems performance under transient operation using a data-driven machine learning approach based on the deep operator network architecture, Energy Convers. Manage. **252**, 115063 (2022).

[33] M. Yin, E. Ban, B. V. Rego, E. Zhang, C. Cavinato, J. D. Humphrey, and G. Em Karniadakis, Simulating progressive intramural damage leading to aortic dissection using DeepONet: An operator–regression neural network, J. R. Soc., Interface. **19**, 20210670 (2022).

[34] M. Yin, E. Zhang, Y. Yu, and G. Em Karniadakis, Interfacing finite elements with deep neural operators for fast multiscale modeling of mechanics problems, arXiv:2203.00003.

[35] S. Lanthaler, S. Mishra, and G. Em Karniadakis, Error estimates for DeepOnets: A deep learning framework in infinite dimensions, arXiv:2102.09618.

[36] B. Deng, Y. Shin, L. Lu, Z. Zhang, and G. Em Karniadakis, Convergence rate of DeepONets for learning operators arising from advection-diffusion equations, arXiv:2102.10621.

[37] C. Marcati and C. Schwab, Exponential convergence of deep operator networks for elliptic partial differential equations, arXiv:2112.08125.

[38] M. G. Fernández-Godino, C. Park, N.-H. Kim, and R. T. Haftka, Review of multi-fidelity models, AIAA **57**, 2039 (2019).

[39] X. Meng and G. Em Karniadakis, A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems, J. Comput. Phys. **401**, 109020 (2020).

[40] L. Lu, M. Dao, P. Kumar, U. Ramamurty, G. Em Karniadakis, and S. Suresh, Extraction of mechanical properties of materials through deep learning from instrumented indentation, Proc. Natl. Acad. Sci. **117**, 7052 (2020).

[41] R. Pestourie, Y. Mroueh, C. Rackauckas, P. Das, and S. G. Johnson, Physics-enhanced deep surrogates for PDEs, arXiv:2111.05841.

[42] J. M. Ziman, *Electrons and Phonons: The Theory of Transport Phenomena in Solids* (Oxford University Press, Oxford, 2001).

[43] N. Li, J. Ren, L. Wang, G. Zhang, P. Hänggi, and B. Li, Colloquium: Phononics: Manipulating heat flow with electronic analogs and beyond, Rev. Mod. Phys. **84**, 1045 (2012).

[44] S. Narayana and Y. Sato, Heat Flux Manipulation with Engineered Thermal Materials, Phys. Rev. Lett. **108**, 214303 (2012).

[45] M. Kadic, T. Bückmann, R. Schittny, and M. Wegener, Metamaterials beyond electromagnetism, Rep. Prog. Phys. **76**, 126501 (2013).

[46] S. De, M. Hassanaly, M. Reynolds, R. N. King, and A. Doostan, Bi-fidelity modeling of uncertain and partially unknown systems using DeepONets, arXiv:2204.00997.

[47] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary Computation 1: Basic Algorithms and Operators* (CRC, Boca Raton, FL, 2018).

[48] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, DEAP: Evolutionary algorithms made easy, J. Mach. Learning Res. **13**, 2171 (2012).

[49] K. Svanberg, A class of globally convergent optimization methods based on conservative convex separable approximations, SIAM J. Optim. **12**, 555 (2002).

[50] R. E. Christiansen and O. Sigmund, Inverse design in photonics by topology optimization: Tutorial, J. Opt. Soc. Am. B **38**, 496 (2021).

[51] https://github.com/lu-group/multifidelity-deeponet.

[52] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, Self-normalizing neural networks, in *Advances in Neural Information Processing Systems*, edited by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017), pp. 972–981.

[53] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980.

[54] L. Lindsay, A. Katre, A. Cepellotti, and N. Mingo, Perspective on *ab initio* phonon thermal transport, J. Appl. Phys. **126**, 050902 (2019).

[55] J. Carrete, B. Vermeersch, A. Katre, A. van Roekeghem, T. Wang, G. K. H. Madsen, and N. Mingo, ALMABTE: A solver of the space–time dependent Boltzmann transport equation for phonons in structured materials, Comput. Phys. Commun. **220**, 351 (2017).

[56] G. Romano, Efficient calculations of the mode-resolved ab-initio thermal conductivity in nanostructures, arXiv:2105.08181.

[57] G. Romano, Openbte: A solver for ab-initio phonon transport in multidimensional structures, arXiv:2106.02764.