Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 375 (2021) 112739
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Universal machine learning for topology optimization

Heng Chi[c], Yuyu Zhang[b], Tsz Ling Elaine Tang[c], Lucia Mirabella[c], Livio Dalloro[c], Le Song[b], Glaucio H. Paulino[a,*]

[a] *School of Civil and Environmental Engineering, Georgia Institute of Technology, 790 Atlantic Drive, Atlanta, GA, 30332, United States of America*
[b] *School of Computational Science and Engineering, Georgia Institute of Technology, 266 Ferst Drive, Atlanta, GA, 30332, United States of America*
[c] *Siemens Corporation, Corporate Technology, 755 College Rd E, Princeton, NJ, 08540, United States of America*

Received 8 March 2019; received in revised form 23 October 2019; accepted 6 November 2019
Available online 24 December 2020

Abstract

We put forward a general machine learning-based topology optimization framework, which greatly accelerates the design process of large-scale problems, without sacrifice in accuracy. The proposed framework has three distinguishing features. First, a novel online training concept is established using data from earlier iterations of the topology optimization process. Thus, the training is done during, rather than before, the topology optimization. Second, a tailored two-scale topology optimization formulation is adopted, which introduces a localized online training strategy. This training strategy can improve both the scalability and accuracy of the proposed framework. Third, an online updating scheme is synergistically incorporated, which continuously improves the prediction accuracy of the machine learning models by providing new data generated from actual physical simulations. Through numerical investigations and design examples, we demonstrate that the aforementioned framework is highly scalable and can efficiently handle design problems with a wide range of discretization levels, different load and boundary conditions, and various design considerations (e.g., the presence of non-designable regions).

1. Introduction

In this work, we create a general framework to amalgamate machine learning with topology optimization so that we can significantly accelerate the design process without sacrificing accuracy. The basic idea behind this framework is to exploit the history data of topology optimization and employ machine learning techniques to discover the underlying mapping between the design variables and their corresponding sensitivities. Our goal is that, once the machine learning model is trained, it avoids solving the state equations for structural responses in the later optimization steps. Conceptually different from any other machine learning-based topology optimization frameworks in the literature, our proposed framework has several unique features, which will be demonstrated throughout this work. First, the proposed framework is universal because it can be applied to any design domains in conjunction with any suitable machine learning models and any advanced linear solvers without any pre-collected

* Corresponding author.
E-mail address: paulino@gatech.edu (G.H. Paulino).

https://doi.org/10.1016/j.cma.2019.112739
0045-7825/

information. Unlike other offline frameworks in the literature, our framework collects the training data and trains the machine learning models online. Therefore, it does not need a separate stage for collecting samples and training of the machine learning models. Second, the proposed framework has great scalability and is capable of effectively handling design problems containing a large number of design variables.

The remainder of this paper is organized as follows. In Section 2, we summarize the relevant attempts in the literature of topology optimization and the motivation of our work. Section 3 briefly reviews the topology optimization formulation for the minimum compliance problem, and Section 4 presents the proposed machine learning-based topology optimization framework, which integrates machine learning and topology optimization through a two-scale formulation. In Section 5, we perform numerical assessments to demonstrate the scalability and accuracy of the deep learning module. Several design examples are presented in Section 6, which showcases the effectiveness of the proposed machine learning-based topology optimization framework on various discretization levels and with various choices of parameters. Finally, Section 7 contains concluding remarks and future research directions.

2. Related work

Since the seminal work of Bendsøe and Kikuchi [1], topology optimization has evolved into a powerful computational tool which has been successfully used in many industrial applications (e.g., aerospace and automotive) and widely implemented in major commercial software. For an overview of the recent developments as well the popular approaches and applications of topology optimization, the interested readers are referred to [2–5]. From a computational perspective, topology optimization is an intensive task because it typically involves hundreds of design iterations, and in order to update the current design, the structural response needs to be solved to compute the sensitivity information. To handle large-scale topology optimization problems (e.g., problems involves millions of design variable and beyond), the associated computational cost could be enormous and one typically has to resort to parallel computing [6–8], advanced iterative solvers [9,10], or multi-scale and multi-resolution approaches [11–13].

Machine learning has been developed to automatically detect patterns in data and then use those patterns to predict future data or perform various kinds of decision making with uncertainty [14,15]. For decades, machine learning has achieved huge success in many aspects. For example, machine learning systems are now widely used in many daily-life applications, such as face recognition [16], automatic vehicle licence plate recognition [17], online advertising [18], web search engine [19], and recommendation systems on e-commerce websites [20]. Since we are entering into the big data era with more and more data available, then machine learning is showing great potential in more and more applications.

Conventional machine learning methods are limited in processing data in raw form, and typically require domain expertise to design features on top of raw data to serve as additional input to machine learning systems [21–23]. On the contrary, deep learning methods can be directly fed with raw data and automatically learn the representations needed for different tasks, which creates major advances over conventional machine learning methods. Deep learning methods turn out to achieve very impressive performance in a variety of challenging problems in the artificial intelligence community, such as surpassing human-level performance on image classification [24], beating grandmaster go players [25,26], and matching human levels in automatic machine translation [27]. Deep learning has demonstrated its ability to discover intricate structures in high-dimensional data without hand-designed features, and therefore can be applied in many domains of science and engineering. Given their successful applications and capability of processing large amount of data, this work will focus on the deep learning models.

More recently, several attempts have been made to apply various machine learning techniques to accelerate the topology optimization process. Essentially, their ideas are to use machine learning models to replace the optimizer either partially or completely so that, once the machine learning models are trained, one could directly employ them to map the initial designs or partially converged designs to the final designs. For example, Ulu et al. [28] developed a data-driven approach for predicting optimized topologies under various loading cases. In this approach, the optimized topologies obtained under a wide range of loading cases are treated as training samples and a feed-forward neural network is adopted together with the Principal Component Analysis (PCA) to establish a mapping for predicting the optimized topology under a given loading scenario. Sosnovik and Oseledets [29] introduced a deep learning-based framework, which uses Convolutional Neural Networks (CNN) to predict the final optimal 2D design based on two input images: a partially converged design and its change with respect to the previous step. By doing so, the total number of optimization steps are reduced. Later, Banga et al. [30] took a similar idea and extended

the work to 3D design problems and incorporated additional inputs such as external loads and boundary conditions. More recently, Yu et al. [31] proposed a two-stage prediction procedure to produce nearly-optimal structural design without any iterations. When given a design domain, load and boundary conditions, the procedure first uses a trained CNN-based encoder and decoder to predict a low resolution structural design and, then, the predicted low-resolution design is mapped to a high resolution one using a trained Generative Adversarial Network (GAN). All the above-mentioned frameworks are proposed in the context of a density-based topology optimization approach. With the same goal, Lei et al. [32] developed a machine learning-based framework for the Moving Morphable Component (MMC) approach. The framework first uses the MMC approach to generate a set of training samples with different external load locations and subsequently applies either Supported Vector Regression (SVR) or K-Nearest Neighbors (KNN) to establish an instantaneous mapping between the design parameters and the optimized topology. Compared to the density-based and level-set approaches, the MMC approach typically contains a relatively small number of design variables, which could reduce the computational burden on the training of the machine learning models.

Several limitations exist in the aforementioned frameworks. The first limitation is related to the data collection and online-training of the machine learning models. In those frameworks, one needs to collect a large amount of training samples and training the machine learning models in a separate stage before they can make satisfactory predictions. For instance, the work of Yu et al. [31] uses designs of 100,000 completed topology optimization simulations as the training samples for the CNN and GAN, and the work by Banga et al. [30] uses a total of 6,000 samples. Collecting those training samples could be a computationally intensive task, especially for large-scale 3D problems, because each training sample corresponds to a completed topology optimization simulation. The second limitation is related to the scalability and the generality of those frameworks. Being restricted by the large amount of training samples needed and the capacity of the machine-learning models, the above-mentioned frameworks only consider either 2D or small-scale (with less than 10,000 design variables) 3D problems, and the design domains considered are all simple rectangular domains with fixed boundary conditions and concentrated load at only one location. Finally, because direct mappings are pervasively adopted, hanging and disconnected members are likely to appear in the final topology as the information about those structural artifacts is not embedded in the training samples of machine learning models.

In this work, we propose a novel machine learning-based topology optimization framework, which takes a different path from all the aforementioned frameworks. *Instead of dedicating a separated offline stage to collect training samples for the machine learning models, our idea is to propose an online training concept which exploits the data generated during the topology optimization and employs the machine learning models to extract the underlying mapping between the design variables and their corresponding sensitivities. Once established, this mapping can be subsequently used in later optimization steps to avoid solving the state equations so that the entire design process can be accelerated. By doing so, the training samples are collected simultaneously as the topology optimization proceeds. Moreover, to ensure that the proposed framework is efficient and scalable for design problems (potentially of any size), we devise a tailored two-scale topology optimization formulation, which allows for the training of machine learning models based on local features of the topology optimization. As a result, the proposed framework is capable of handling 3D large-scale design of a wide range of problem sizes while achieving significant speedup*. For example, we demonstrate that the proposed framework can achieve close to an order of magnitude speedup in a 3D design problem with more than 1 million design variables. Additionally, we introduce an online updating scheme to frequently update the machine learning model based on new data generated from physical simulations so that we constantly improve the prediction accuracy throughout the optimization and avoid structural artifacts such as hanging and disconnected members in the final design. Four design examples with varied design scenarios are presented to demonstrate the universality and potential of the proposed machine learning-based framework in practical engineering applications.

3. Topology optimization formulation

In this section, the topology optimization formulation for the classical compliance-minimization problem is briefly reviewed. Throughout, we assume that the design domain is discretized by a finite element mesh and adopt the standard density-based approach [4,33], where the material distribution is characterized by an element-wise constant function.

Our goal is to find the structural topology, which has the most stiffness under prescribed load and boundary conditions. For a given finite element mesh with N nodes and M elements, we denote $\mathbf{f} \in \mathbb{R}^{dN\times 1}$ as the

applied global force vector. We introduce $\mathbf{z}$ as the vector of design variables, whose ith component z_i is the design variable associated with the ith element. Within this setting, the topology optimization formulation for compliance-minimization problems is stated as:

$$\begin{aligned} \min_{\mathbf{z}} \quad & c(\mathbf{z}) = \mathbf{f}^\top \mathbf{u}(\mathbf{z}) \\ \text{s.t.} \quad & g_V(\mathbf{z}) = \mathbf{v}^\top \overline{\mathbf{z}} - V_{\max} \leq 0 \\ & 0 \leq z_i \leq 1 \quad \forall i \in \{1, \ldots, M\} \\ \text{with} \quad & \mathbf{K}(\mathbf{z})\mathbf{u}(\mathbf{z}) = \mathbf{f} \text{ and } \overline{\mathbf{z}} = \mathbf{P}\mathbf{z}, \end{aligned} \tag{1}$$

where $c(\mathbf{z})$ is the compliance function, $\mathbf{u}(\mathbf{z}) \in \mathbb{R}^{dN\times 1}$ is the global displacement vector, $\mathbf{v}$ is a vector whose ith component v_i is the volume of element i, and $V_{\max}$ is the maximum allowable volume imposed on the design.

To ensure the well-posedness of the formulation and impose a minimum length scale on the design, the filtered design variable $\overline{\mathbf{z}}$ is used in which $\mathbf{P}$ is the density filter matrix whose (i, j)-th component is given by

$$(\mathbf{P})_{ij} = \frac{\max\left(0, R - |\mathbf{x}_i^* - \mathbf{x}_j^*|\right)}{\sum_{k\in\mathcal{S}(j)}\left(R - |\mathbf{x}_k^* - \mathbf{x}_j^*|\right)}, \tag{2}$$

where R is the radius of the density filter, $\mathcal{S}(j)$ denotes the indices of elements whose centroids fall within radius R of the centroid of the jth element, and $\mathbf{x}_i^*$ and $\mathbf{x}_j^*$ stand for the centroid of the ith and jth elements, respectively.

The standard Solid Isotropic Microstructures with Penalization (SIMP) scheme [34–36] is adopted to penalize the intermediate densities throughout. In the SIMP scheme, the global stiffness matrix $\mathbf{K}$ is interpolated as

$$\mathbf{K}(\mathbf{z}) = \bigcup_j E_j \mathbf{k}_j^0, \tag{3}$$

where $\mathbf{k}_j^0$ is the element stiffness matrix for the jth element when the material is solid, $\bigcup$ represents the standard assembly procedure in FEM [37], and E_j is interpolated stiffness[1] of element j given by

$$E_j = E_{\min} + (1 - E_{\min})(\overline{\mathbf{z}})^p. \tag{4}$$

In the above expression, $E_{\min}$ is the Ersatz stiffness and p is the SIMP penalization parameter, which are taken to be 10^{-4} and 3, respectively, throughout this work.

The sensitivity information is needed to perform the design variable update. In the following discussion, we denote by $\mathbf{G}$ and $\overline{\mathbf{G}}$ the sensitivity vectors of the compliance with respect to $\mathbf{z}$ and $\overline{\mathbf{z}}$, respectively. We first compute $\overline{\mathbf{G}}$ as

$$\overline{G}_i = \frac{\partial c}{\partial \overline{z}_i} = -p(\overline{z}_i)^{p-1}\left(\mathbf{u}_i\right)^\top \mathbf{k}_j^0 \mathbf{u}_i, \tag{5}$$

where $\mathbf{u}_i$ is the displacement vector of the ith element. Having obtained $\overline{\mathbf{G}}$, $\mathbf{G}$ is given by

$$\mathbf{G} = \mathbf{P}^\top \overline{\mathbf{G}}. \tag{6}$$

On the other hand, the sensitivity of the volume constraint function g_V is simply given by

$$\frac{\partial g_V}{\partial \mathbf{z}} = \mathbf{P}^\top \mathbf{v}. \tag{7}$$

Once the sensitivities of both the objective and constraint functions are obtained, we use the Optimality Criteria (OC) method [4] to update the design variables.[2] If we denote $\mathbf{z}^{(k)}$ and $\mathbf{G}^{(k)}$ as the design variable and sensitivity vectors at the optimization step k, respectively, the OC method updates the design variable vector for the optimization step $k + 1$ as

$$z_i^{(k+1)} = \begin{cases} \max(z_{\min}, z_i^{(k)} - m) & \text{if } z_i^{(k)}\left(B_i^{(k)}\right)^\eta \leq \max(z_{\min}, z_i^{(k)} - m) \\ \min(1, z_i^{(k)} + m) & \text{if } \min(1, z_i^{(k)} + m) \leq z_i^{(k)}\left(B_i^{(k)}\right)^\eta \\ z_i^{(k)}\left(B_i^{(k)}\right)^\eta & \text{otherwise,} \end{cases} \tag{8}$$

[1] The interpolated stiffness is normalized by the Young's modulus of the solid material.

[2] We note that our proposed machine learning-based framework also works with any gradient-based design update scheme, for instance, the popular Method of Moving Asymptotes (MMA) [38].

where, as usual, m is the move limit and η is the damping coefficient. The coefficient $B_i^{(k)}$ is given by

$$B_i^{(k)} = -\frac{G_i^{(k)}}{\left[\sum_j (\mathbf{P})_{ji} v_j\right] \Lambda}, \tag{9}$$

where $\Lambda > 0$ is the Lagrangian multiplier found using a bi-section algorithm [4] such that $g_V(\mathbf{z}^{(k+1)}) = 0$.

4. Universal machine learning for topology optimization: Concept, methodology and algorithms

We propose a general-purpose framework, which uses machine learning to considerably accelerate large-scale topology optimization without any sacrifice in accuracy. Unlike the dominating offline training strategies in the literature, where the data collection and machine learning training are performed before the actual topology optimization, this framework adopts a novel online training & updating strategy, where data collection and machine learning training happen simultaneously during the topology optimization process. In the sequel, we will first provide an overview of the framework and introduce the novel online training & updating strategy. We will then focus on the machine learning module of the framework and discuss details regarding the deep learning model that we adopt. Finally, we introduce a tailored two-scale topology optimization step-up together with a localized online training and prediction concept, and discuss how they can ensure the efficiency, accuracy, and scalability of the proposed framework.

4.1. An overview of the proposed framework and the online training & updating strategy

Topology optimization is an iterative process which often involves hundreds of steps. Every time we come up with a new design, we need to solve for the structural response of the current design to compute the sensitivity information. For large-scale topology optimization, this procedure is computationally intensive.

A large amount of history data (e.g., design variables, their corresponding sensitivities, and displacement solutions) are generated during topology optimization, of which we typically do not make full use. *Thus, our key idea is to use machine learning models to learn the underlying mapping between design variables and their corresponding sensitivities from those history data. Once the machine learning model is trained, then it can be employed in the later optimization steps to directly predict the sensitivity information of the current design without solving the state equations.*

In our proposed approach, the machine learning model is trained online in several stages: an initial online training stage and several online updating ones. To control the design iteration which each stage starts, we introduce two parameters, N_I and N_F, which are the initial online training step and online updating frequency, respectively. Additionally, to control the amount of history data used in training, we also introduce W_I and W_U as the window sizes for initial online training and online updating, respectively. As for the machine learning model, a fully-connected Deep Neural Network (DNN) is employed in this work, whose details will be provided in Section 4.2.

Initial online training. In the proposed framework, we start (topology optimization) with the standard procedure (e.g., solving the state equation and sensitivity analysis based on Eq. (6)) in the first $N_I + W_I - 1$ optimization steps, where the history data in the last W_I steps (i.e., step N_I to step $N_I + W_I - 1$) are collected for the initial online training.[3] Afterwards, starting from step $N_I + W_I$, instead of following the standard procedure, we use the trained machine learning model to directly predict the sensitivities based on the new design variables, which allows us to avoid the most computationally expensive tasks of solving the state equations and sensitivity analyses.

Online updating. To guarantee the accuracy of the predicted sensitivity in the long term (i.e., steps farther away from the initial online training), the proposed framework also introduces an online updating strategy to dynamically update the machine learning model by switching back to the standard procedure for one optimization step to generate new data. We use N_F to control the frequency of the online updating, meaning that we will perform the online updating every N_F optimization steps after the initial prediction step ($N_I + W_I$). In the online updating, the data collected from the past W_U exact evaluations are used.

[3] In our framework, we discard the data from first $N_I - 1$ because they generally have the most variations and are biased to the initial guess.

As an illustration, let us consider $N_I = 10$, $W_I = 5$, $N_F = 10$ and $W_U = 2$. With this set, we start the optimization with the standard procedure in the first 14 steps and use the data generated from step 10 to step 14 to train the machine learning model. Starting from optimization step 15, we use the trained machine learning model to predict the sensitivity. Because the online updating frequency is 10, we switch back to the standard procedure at optimization step 25 to generate new data, and use the data in optimization 14 and 25 (because the window size for online update is $W_U = 2$) as the input to update the machine learning model. We then continue to use the updated machine learning model to predict the sensitivity in the following steps and recursively update the model every 10 steps until either the convergence criteria are fulfilled or the maximum allowable step is reached.

To ensure that the proposed framework is efficient, accurate and scalable, we devise a two-scale topology optimization setup which includes a coarse-scale mesh and a fine-scale mesh. On the fine-scale mesh, we perform all the design variable update every optimization step but only solve the state equations in those steps where we need to collect data for training. On the other hand, on the coarse-scale mesh, no design variable update is performed but the state equation is solved at every optimization step based on the stiffness distribution mapped from the fine-scale mesh. The main reason for the two-scale topology optimization setup is to enable a localized training and prediction approach, which guarantees the scalability of the framework. More discussion on the two-scale topology optimization setup and localized training and prediction of the machine learning model will be discussed in Section 4.3. To conclude the overview, we summarize the entire procedure of the proposed machine learning-based topology optimization framework in Algorithm 1.

Algorithm 1: Proposed framework of universal machine learning for topology optimization.

Input: $\mathbf{z}^{(0)}$, $V_{\max}$, $\mathbf{f}^C$, $\mathbf{f}$, R, Tol, $Iter_{\max}$, N_I, N_F, W_I and W_U;
Form filter matrix $\mathbf{P}$ based on (2);
for $k = 0$ **to** $Iter_{\max}$ **do**
  Filter design variables: $\overline{\mathbf{z}}^{(k)} = \mathbf{P}\mathbf{z}^{(k)}$;
  Assemble the global stiffness matrix $\mathbf{K}^C$ on the coarse-scale mesh based on (14);
  Solve the state equation on coarse-scale mesh: $\mathbf{u}^C = (\mathbf{K}^C)^{-1}\mathbf{f}^C$;
  Evaluate the strain vector $\boldsymbol{\varepsilon}^{C,(k)}$ on the coarse-scale mesh based on $\mathbf{u}^C$;
  if $k < N_I + W_I$ **or** $\mathrm{mod}(\max(k - N_I - W_I, 1), N_F) = 0$ **then**
    Assemble the global stiffness matrix $\mathbf{K}$ on the fine-scale mesh based on (3);
    Solve the state equation on fine-scale mesh: $\mathbf{u} = \mathbf{K}^{-1}\mathbf{f}$;
    Evaluate $\overline{\mathbf{G}}^{(k)}$ based on (5);
    Store the history data $\overline{\mathbf{z}}^{(k)}$, $\overline{\mathbf{G}}^{(k)}$ and $\boldsymbol{\varepsilon}^{C,(k)}$;
    if $k = N_I + W_I - 1$ **then**
      Initial training of the machine learning model using last W_I step of collected data;
    else if $\mathrm{mod}(\max(k - N_I - W_I, 1), N_F) = 0$ **then**
      Online update of the machine learning model using last W_U step of collected data;
    end
    Compute the sensitivity vector as $\mathbf{G}^{(k)} = \mathbf{P}^\top \overline{\mathbf{G}}^{(k)}$;
    Update $\mathbf{z}^{(k+1)}$ using $\mathbf{G}^{(k)}$ based on (8);
  else
    Use the machine learning model to predict $\widetilde{\overline{\mathbf{G}}}^{(k)}$ based on the input $\overline{\mathbf{z}}^{(k)}$ and $\boldsymbol{\varepsilon}^C$;
    Compute the predicted sensitivity as $\widetilde{\mathbf{G}}^{(k)} = \mathbf{P}^\top \widetilde{\overline{\mathbf{G}}}^{(k)}$;
    Update $\mathbf{z}^{(k+1)}$ using $\widetilde{\mathbf{G}}^{(k)}$ based on (8);
  end
end
if $||z^{(k+1)} - z^{(k)}||_\infty \leq Tol$ **then**
  Output: optimization converged and plot final design;
end

4.2. The machine learning model: Deep Neural Networks (DNNs)

In this work, we employ the fully-connected Deep Neural Networks (DNNs) as the universal function approximator for predicting the sensitivities of the objective function because of (1) its powerful capability of processing high-dimensional inputs and outputs without domain expertise; (2) its powerful approximation property of nonlinear mapping between high-dimensional spaces [39–41]; and (3) its success in achieving the state-of-the-art performance in various machine learning tasks processing raw data in forms of vision, audio, and natural language. Notice that our proposed framework is independent of any specific implementation of the machine learning module. Thus, other machine learning and deep machine learning models, such as Support Vector Machine (SVM) [42,43], Convolutional Neural Networks (CNNs) [44,45] and Residual Networks (ResNets) [46], and hybridized models (e.g., Principal Component Analysis (PCA) [47,48] and DNN), can be directly applied in the proposed framework.

The fully-connected DNN consists of one input layer, multiple hidden layers, and one output layer. Each hidden layer has a set of neurons, each of which takes an input value and performs a non-linear activation to generate its output value. The number of hidden layers is a hyper-parameter and can be tuned according to the trade-off between the computational complexity and model accuracy. Let us denote N_h as the total number of hidden layers in DNN. During prediction, each hidden layer takes the output of the last layer as input, and performs feed-forward computation as follows

$$\mathbf{h}_i = \sigma(\mathbf{W}_i \mathbf{h}_{i-1} + b_i), i = \{1, \ldots, N_h\} \tag{10}$$

where $\mathbf{h}_i$ is the output of the ith hidden layer; $\mathbf{W}_i$ and b_i are respectively the weight vector and the bias of the ith layer that can be randomly initialized and then optimized during model training; and $\sigma(\cdot)$ is a non-linear activation function. By convention, we designate $\mathbf{h}_0$ as the input of the input layer. The output layer is obtained by applying a linear transformation of the output of the last hidden layer as

$$\mathbf{y} = \mathbf{W}_{\text{out}} \mathbf{h}_N, \tag{11}$$

where $\mathbf{W}_{\text{out}}$ is also a weight matrix, which will also be learned according to the training data. The architecture of our DNN model is illustrated in Fig. 1. In this work, the output $\mathbf{y}$ is the chosen as the sensitivity of the compliance with respect to the filtered design variables.

In this work, we choose to use the Parametric Rectified Linear Unit (PReLU) as the activation function, which generalizes the traditional rectified unit and is shown to achieve impressive performance on image classification tasks [24]. The PReLU activation function is defined as follows:

$$\sigma(x) = \max(0, x) + a \min(0, x), \tag{12}$$

where a is a learnable parameter, and x is the input of each neuron in the DNN. To train the DNN model, we collect the training data from exact evaluations in the topology optimization as the supervision signal. During training, we use Adam optimizer [49], which is a widely used algorithm for stochastic gradient-based optimization. In the initial training, we randomly initialize all the learnable parameters in the DNN and, in each subsequent online updating, we take the optimized parameters from the last training step as the initial guess and continue to update them based on the new training data received.

4.3. Two-scale topology optimization and local online training & prediction

As mentioned at the beginning of this section, a key concept of the proposed machine learning-based framework is to make use of the history data in topology optimization to train a machine learning model, which can make a direct prediction of the sensitivity information based on the design variables. However, it is common nowadays for topology optimization problems to have millions of (or even more) design variables and thus building a scalable online machine learning model for those problems is by no means an easy task. More specifically, the challenges arise from three aspects. First, as the number of design variables increases, the dimensions of the input and output grow accordingly, making it much more difficult to learn the mapping from input to output due to limited machine learning model capacity. Second, although the increased learning difficulty may be alleviated by providing more supervisions, i.e., collecting "exact" gradients in more optimization steps for model training, it can significantly affect the speedup performance for large-scale problems because evaluating the "exact" gradients in topology

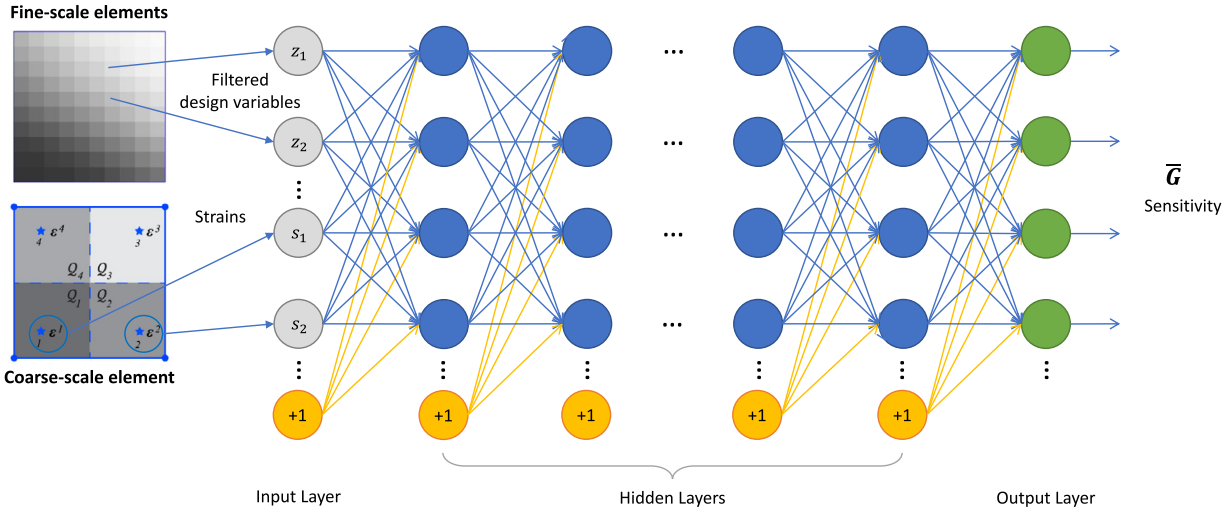

Fig. 1. Architecture of the fully-connected Deep Neural Networks.

Table 1
Comparison of the number of parameters to learn and the GPU memory requirement between a single-scale and a two-scale scheme for online training. The DNN for both cases is assumed to have 4 hidden layers with each hidden layer having 1,000 neurons.

Scheme		86K DVs	250K DVs	1.5M DVs
Single-scale	# parameters	175M	867M	3B
	GPU memory	3.9GB	–	–
Two-scale	# parameters	3.3M	3.3M	3.3M
	GPU memory	0.7GB	0.7GB	0.9GB

optimization is expensive. Third, the size of the machine learning model grows according to the dimensions of the input and output, which costs more GPU memory for training and prediction; and the maximum GPU memory available in existing hardware can significantly limit the scale of the optimization problem that can be handled. To illustrate the last point, let us consider a topology optimization problem discretized by three meshes, each has $86K$, $250K$, and $1.5M$ elements respectively. Let us also assume that the DNN we use has 4 hidden layers with 1,000 neurons in each layer. We record the total number of model parameters and the corresponding GPU memory cost in Table 1. For a single-scale scheme, the number of neurons in the input and output layers of the DNN is equal to the total number of elements in the mesh. Thus, as we increase problem size, the number of parameters in the DNN increases drastically and so does the GPU memory required. For meshes with 250K and 1.5M elements, we will run out of memory on a 12GB GPU. In order to address this fundamental challenge related to the scalability of the framework, this section proposes a two-scale topology optimization scheme together with local online training and updating to make better use of the history data in topology optimization.

4.3.1. Two-scale topology optimization setup

According to Eqs. (5) and (6), the sensitivity of each element depends on both the design and state variables of that element. However, the information about the state variables of each element is not available unless we solve the state equation. In order to provide sufficient information to the machine learning model and, at the same time, to avoid the most time-consuming step of solving the state equation, we introduce a topology optimization formulation with two discretization levels: a coarse-scale mesh and a fine-scale mesh. As we mentioned, the design variables (and the corresponding filtered design variables) live on the fine-scale discretization and they will be updated every optimization step. However, on the fine-scale mesh, the state equations are only solved in those optimization steps when we collect training data for the machine learning model. On the contrary, on the coarse-scale mesh, no optimization is performed but the state equation is solved at every optimization step to provide

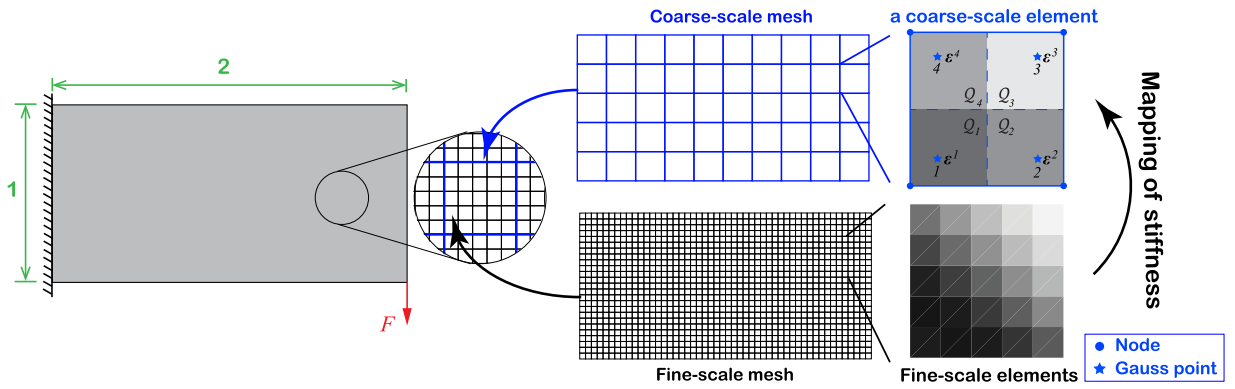

Fig. 2. A 2D illustration of the setup of the two mesh scales and the mapping from the fine-scale mesh to the coarse-scale mesh for a cantilever beam design problem.

information about the state variables to the machine learning model. Assuming that the number of elements of the coarse-scale mesh is much smaller than that of the fine-scale mesh, the time spent in solving the state equation on the coarse-scale mesh will be negligible.

The setup of the coarse-scale and fine-scale meshes is illustrated in Fig. 2. We note that although the illustration is in 2D, the numerical examples in this work also consider 3D problems. On both coarse-scale and fine-scale meshes, we adopt regular hexahedral (brick) finite elements with linear displacement interpolations and assume that the fine-scale mesh is fully embedded in the coarse-scale mesh. Under this assumption and because of the regularity of the two meshes, every element in the coarse-scale mesh contains the same number of elements in the fine-scale mesh. Thus, we introduce a parameter called block size N_B to quantify how many fine-scale elements are contained on each side of the coarse-scale element. For example, the illustration in Fig. 2 has a block size of $N_B = 5$, meaning every element in the coarse-scale mesh constrains $5 \times 5 = 25$ fine-scale elements.

Because the design update is only performed on the fine-scale mesh, one needs to map the stiffness distribution of the fine-scale mesh to the coarse-scale mesh at every optimization step. Taking inspiration from the multi-resolution topology optimization frameworks [12,50], we define the mapping in the following manner. For a given coarse-scale finite element with a total of n_G integration points, as illustrated in Fig. 2, we divide it into a total of n_G sub-regions and each sub-region is associated with one of its integration point. In this work, n_G is equal to 4 in 2D and 8 in 3D. In addition, for the coarse-scale finite element k, we denote $\mathcal{Q}_j^k$ ($j = 1, \ldots, 4$ in 2D or $j = 1, \ldots, 8$ in 3D) as its sub-region associated with the jth integration point. Under this convention, the mapped stiffness at the jth integration point of coarse-scale element k, which is denoted as $E_j^{C,k}$, is computed as the weighted average of the interpolated stiffness of all the fine-scale elements that fall within in the sub-region $\mathcal{Q}_j^k$, namely,

$$E_j^{C,k} = \frac{1}{\sum_{i \in \mathcal{Q}_j^k} (w_i^{\mathcal{Q}_j^k})} \sum_{i \in \mathcal{Q}_j^k} (w_i^{\mathcal{Q}_j^k})\, E_i, \tag{13}$$

where recall from (4) that E_i is the interpolated stiffness of element i in the fine-scale mesh, and $w_i^{\mathcal{Q}_j^k}$ is the weight assigned to E_i in sub-region $\mathcal{Q}_j^k$. If element i in the fine-scale mesh falls completely in $\mathcal{Q}_j^k$, then the weight $w_i^{\mathcal{Q}_j^k}$ is taken to be 1. Otherwise, if element i falls into a total of n sub-regions, we then take its weight as $w_i^{\mathcal{Q}_j^k} = \frac{1}{n}$ for all sub-regions $\mathcal{Q}_j^k$. With the stiffness mapping and assuming that all coarse-scale elements are identical, the global stiffness matrix $\mathbf{K}^C$ on the coarse-scale mesh is computed as

$$\mathbf{K}^C = \bigcup_k \Big[\sum_{j=1}^{n_G} E_j^{C,k} (\mathbf{B}_j^C)^\top \mathbf{D}_0 \mathbf{B}_j^C \, J_j^C \Big], \tag{14}$$

where $\mathbf{D}_0$ is the constitutive matrix of the solid phase, and $\mathbf{B}_j^C$ and J_j^C are the strain–displacement matrix and the Jacobian of iso-parametric mapping at the jth integration point of a coarse-scale element, respectively. The nodal

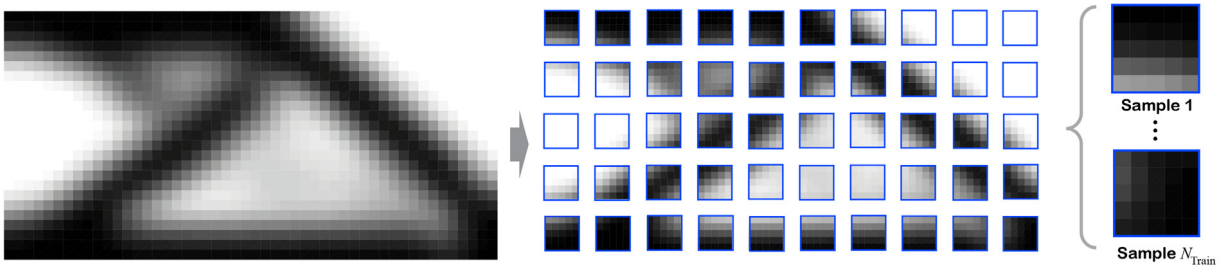

Fig. 3. A 2D illustration of the setup of the two mesh scales for a cantilever beam design problem. The block size N_B in this illustration is 5.

displacement vector of the coarse-scale mesh, denoted as $\mathbf{u}^C$, can then be obtained by solving the state equation as

$$\mathbf{u}^C = (\mathbf{K}^C)^{-1}\mathbf{f}^C, \tag{15}$$

where $\mathbf{f}^C$ is the applied force vector on the coarse-scale mesh. Finally, we also remark that other schemes of mapping the stiffness between the two meshes can also be adopted in our proposed approach, for example, the ones used in References [12,50].

4.3.2. Local online training & update approach

Having introduced the machine learning model (i.e., fully-connected DNN) and the two-scale topology optimization setup, we are now ready to propose the local online training & updating approach which capitalizes on the main features of the two-scale topology optimization formulation. In the proposed approach, instead of treating each global design as an individual training sample, we view each element in the coarse-scale mesh together with its enclosed fine-scale elements as an independent training instance. An illustration is shown in Fig. 3. The next question is what will be the proper inputs from the coarse- and fine-scale meshes. In this work, we construct a proper set of input by examining the dependence of sensitivity and the availability of information in the two mesh levels.

Training data from fine-scale mesh. Naturally, the choice of input from the fine-scale mesh is the design variables (or closely-related variables) in each training instance. In this work, we choose the filtered design variables as the input data from the fine-scale mesh because they possess smoother distribution than the design variables due to the effect of the density filter. Accordingly, we choose the output data to be the sensitivities of the objective function with respect to the filtered design variables within each instance. We will denote $\widetilde{\overline{\mathbf{G}}}$ as a prediction of $\overline{\mathbf{G}}$ by the deep learning model. Once we have $\widetilde{\overline{\mathbf{G}}}$ at hand, the prediction for $\mathbf{G}$, denoted as $\widetilde{\mathbf{G}}$, can be efficiently obtained through $\widetilde{\mathbf{G}} = \mathbf{P}^\top \widetilde{\overline{\mathbf{G}}}$.

Training data from coarse-scale mesh. Unlike the fine-scale mesh, we know the structural responses on the coarse-scale mesh at every optimization step. Thus, the input data from the coarse-mesh will be taken as the state variables on the coarse-scale mesh. Because we have access to all the information about the state variables, including the displacement, strain, and stress fields, the question then becomes: what state variable should we select as the input data to the deep learning model from the coarse-scale mesh so that we could get the most accurate prediction. The most natural choice according to the expression (5) is the nodal displacement vector of each coarse-scale element. However, as we will show soon, the choice of the nodal displacement vector of each coarse-scale element as the input to deep learning model will result in an unsatisfactory level of accuracy in prediction. Instead, this work proposes to use the strain vectors at all the integration points of each coarse-scale element as input. For the kth coarse-scale element, we use $\boldsymbol{\varepsilon}_k^C$ to denote a vector collecting the strain vectors at all the integration points of that element, namely, $\boldsymbol{\varepsilon}_k^C = [\boldsymbol{\varepsilon}_1^{C,k}, \ldots, \boldsymbol{\varepsilon}_{n_G}^{C,k}]^\top$, where $\boldsymbol{\varepsilon}_j^{C,k} = [\boldsymbol{\varepsilon}_{xx,j}^{C,k}, \boldsymbol{\varepsilon}_{yy,j}^{C,k}, \boldsymbol{\varepsilon}_{zz,j}^{C,k}, \gamma_{xy,j}^{C,k}, \gamma_{xz,j}^{C,k}, \gamma_{yz,j}^{C,k}]^\top$ is the strain vector obtained at the jth integration point of the kth coarse-scale element. The strain vector $\boldsymbol{\varepsilon}_j^{C,k}$ can be computed from the nodal displacement vector $\mathbf{u}_k^C$ of element k following the standard finite element procedure using the values of the gradients of the shape functions at the jth integration point of that element.

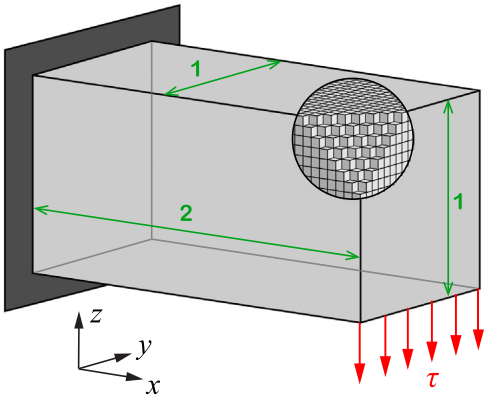

Fig. 4. An illustration of the design domain, load and boundary condition of the cantilever example. The training and prediction data is collected on a mesh with 16,000 elements and 18,081 nodes.

Here, we use a simple numerical experiment to demonstrate the improved accuracy of prediction achieved by choosing the strain vector $\boldsymbol{\varepsilon}_k^C$ instead of the nodal displacement vector $\mathbf{u}_k^C$ as input. Afterwards, an explanation will be provided about the reasons behind this improved prediction accuracy.

In this simple experiment, we consider a cantilever beam design domain of dimensions $2 \times 1 \times 1$ as shown in Fig. 4. The design domain is fixed on its face $x = 0$ and subjected to a distributed load $\tau = 1$ in the negative z direction at the lower edge of the face, $x = 2$. The volume fraction and the filter radius of this problem are taken to be $V_{\max} = 12\%$ and $R = 0.08$, respectively. In this experiment, we want to evaluate whether the fully-connected DNN is able to learn the exact expression to compute the sensitivity $\overline{\mathbf{G}}$ if it is given all the necessary input, i.e., both design and state variables of each element. To that end, we consider a limit case where the coarse-scale and fine-scale meshes are identical (i.e., $N_B = 1$) and each has 16,000 elements and 18,081 nodes, as shown in Fig. 4. In this case, the input data to the fully-connected DNN in our localized training strategy reduces to the filtered design variable and nodal displacement vector or the strain vector of each element, and based on the analytic expression (5), those input data provide sufficient information to compute the sensitivity of each element. We also separate the topology optimization and machine learning modules. We first collect the data for both training and prediction from a completed topology optimization procedure with 200 optimization steps, and use the data collected from optimization steps 11 to 15 to train a fully-connected DNN (the data from the first 10 steps are discarded). We then use the trained fully-connected DNN to predict the sensitivity $\overline{G}_i$ in each element of the mesh from steps 16 to 200. No online update is performed here. To quantify the accuracy of the prediction, we introduce the angle of deviation between the exact sensitivity $\overline{\mathbf{G}}$ and the predicted sensitivity $\widetilde{\overline{\mathbf{G}}}$, as

$$\theta_{\text{error}} = \arccos\Big(\frac{\overline{\mathbf{G}}^{\top}\widetilde{\overline{\mathbf{G}}}}{\|\overline{\mathbf{G}}\|\,\|\widetilde{\overline{\mathbf{G}}}\|}\Big). \tag{16}$$

In terms of the hyper-parameters of the fully-connected DNN, we fix the total number of hidden layers to be $N_h = 4$ and vary the total number of neurons in each hidden layer as 100, 500 and 1,000. In the training stage of the DNN, the maximum iteration is set as 1,000 and the batch size is selected as 20,000. To stabilize the training process, a decaying learning rate schedule is defined: the initial learning rate is taken as $\ell_r = 0.0005$ and will be reduced every 1,000 iterations by a factor of 0.5.

In Fig. 5, we plot the angle of deviation θ_{error} as function of the optimization step obtained by the DNN trained using two different sets of input data. In Fig. 5(a), the DNN is trained using the nodal displacement and filtered design variable of each element as input; and, in Fig. 5(b), the DNN is trained using the element-level strain vector and the filtered design variable of each element as input. By comparing the two figures, we can conclude that the DNN trained with the element-level strain vector and the filtered design variable of each element as input exhibits significantly improved the accuracy in prediction as compared to the one trained using the nodal displacement vector and the filtered design variables. Moreover, we also notice that, increasing the number of neurons in each hidden layer can monotonically improve the prediction accuracy of the DNN trained using strain vectors and filtered design

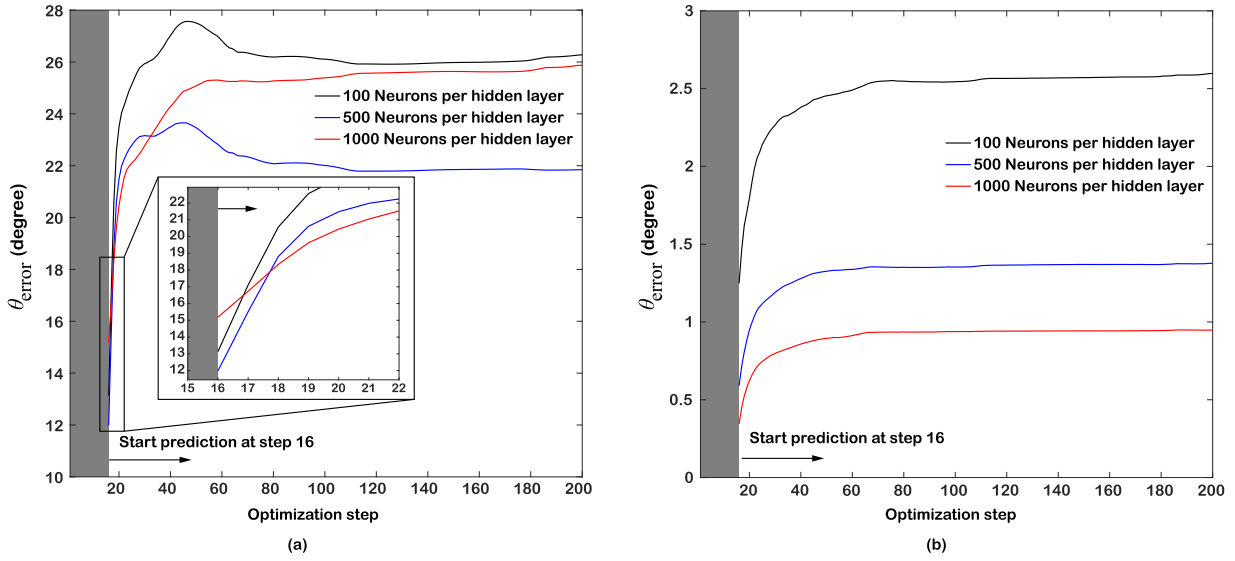

Fig. 5. The angle of deviation θ_{error} as function of the optimization steps obtained by (a) the DNN trained using the nodal displacement and filtered design variable of each element as input, and (b) the DNN trained using the element-level strain vector and the filtered design variable of each element as input.

variables but cannot improve the accuracy of the DNN trained using the nodal displacement vectors and filtered design variables. The results of this simple experiment indicate that the choice of input data can greatly influence the accuracy and prediction performance of the fully-connected DNN.

Here, we provide an explanation of why such different predication accuracy is obtained by training the DNN with different sets of input data. We recall that from (5) that $\overline{\mathbf{G}}$ is given by

$$\overline{G}_i = -p(\overline{z}_i)^{p-1}(\mathbf{u}_i)^\top \mathbf{k}^0 \mathbf{u}_i, \tag{17}$$

where $\mathbf{k}^0$ is the local stiffness associated with solid materials that is identical for each element in the present work. Alternatively, we can also express $\overline{\mathbf{G}}$ in terms of element-level strain vectors,

$$\overline{G}_i = -p(\overline{z}_i)^{p-1} \sum_{j=1}^{n_G} \left[(\boldsymbol{\varepsilon}_j^i)^\top \mathbf{D}^0 \boldsymbol{\varepsilon}_j^i \right], \tag{18}$$

where $\boldsymbol{\varepsilon}_j^i$ is the strain vector at the jth integration point of element i, and $\mathbf{D}^0$ is the modulus matrix of the solid material. According to expressions (17) and (18), $\overline{G}_i$ depends on the nodal displacement vector $\mathbf{u}_i$ through a quadratic form determined by matrix $\mathbf{k}^0$, while its dependence on the element-level strain vector is a quadratic form determined by matrix $\mathbf{D}^0$. When training the DNN, we are essentially trying to learn the coefficients or, equivalently, the eigenvalues and their associated eigenvectors, of matrices $\mathbf{k}^0$ or $\mathbf{D}^0$. Let us examine matrix $\mathbf{k}^0$ first. The matrix $\mathbf{k}^0$ is positive semi-definite with 6 zero eigenvalues representing the rigid body motions. Therefore,we can express the quadratic form $(\mathbf{u}_i)^\top \mathbf{k}^0 \mathbf{u}_i$ in (17) as

$$(\mathbf{u}_i)^\top \mathbf{k}^0 \mathbf{u}_i = \sum_{k=1}^{n-6} \lambda_k \left[(\mathbf{q}_k)^\top \mathbf{u}_i \right]^2 + \sum_{k=n-5}^{n} 0 \left[(\mathbf{q}_k)^\top \mathbf{u}_i \right]^2, \tag{19}$$

where n is the total number of displacement DOFs in the element; $\lambda_1, \ldots, \lambda_{n-6}$ are the positive eigenvalues of $\mathbf{k}^0$ with $\mathbf{q}_1, \ldots, \mathbf{q}_{n-6}$ being their corresponding eigenvectors; and $\mathbf{q}_{n-5}, \ldots, \mathbf{q}_n$ are the eigenvectors associated with the 6 zero eigenvalues. According to the above expressions, due to the presence of zero eigenvalues, it is impossible to correctly learn vectors $\mathbf{q}_{n-5}, \ldots, \mathbf{q}_n$ from the training data. As a result, learning all the coefficients of the matrix $\mathbf{k}^0$ with the nodal displacement vector becomes an ill-posed task. Unlike $\mathbf{k}^0$, the $\mathbf{D}^0$ matrix is strictly positive definite and, thus, learning all its coefficients is a well-posed task. Based on the above analysis, we remark that,

another potential solution could be still using the displacement vector from the coarse-scale mesh but adding a small regularization term (e.g. Tikhonov regularization [51–53]) to the coarse-scale stiffness matrix in order to avoid the singular displacement modes.

In summary, an illustration of the inputs to the DNN in the proposed local online training & updating approach is shown in Fig. 1. Compared to the global approach, this local online training & updating approach can address the fundamental challenges early in this subsection. First, by adopting the local approach, we can significantly increase the total number as well as the diversity of the training samples. From a machine learning perspective, more diverse training samples could potentially lead to more accurate predictions. Second, the local approach allows us to bound the size as well as the GPU memory requirement of the fully-connected DNN for problems of any size. To illustrate the advantages of the local online training & updating approach enabled by the two-scale topology optimization, let us consider the same example at the beginning of Section 4.3 with a two-scale setup of block size $N_B = 5$. For all the three problem sizes considered in this example, the number of neurons in both input and output layers of the DNN are always equal to the total number of fine-scale elements enclosed in each coarse-scale element plus the strain information of every integration point of each coarse-scale element, which is $125 + 48 = 173$ neurons in the present case. As a result, with the two-scale scheme, the total number of parameters in the DNN and the GPU memory required for training stay almost constant, even when we drastically increase the problem size, as shown in Table 1.

4.4. Improve training efficiency by randomly dropping out void training samples

With the local online training & updating approach, many void samples exist in the training data set. A void sample is referred to as a training instance whose enclosed filtered DVs are all zero. The void sample does not contain as much useful information as the non-void ones, and thus we propose to randomly drop out a portion of them from our training set to improve the online training efficiency without affecting accuracy. This is determined by the drop out rate P_{drop}, which is the probability of removing any void instance. For example, if we choose $P_{\text{drop}} = 0.9$, each void training instance has a 90% chance of being removed from the training set. As will be demonstrated in the design examples, this proposed strategy of dropping out void training samples can greatly improve the efficiency of the training of the deep learning model without sacrificing its accuracy.

5. Numerical assessment

Before performing design examples that integrate the machine learning and the two-scale topology optimization modules, this section performs thorough numerical assessments to demonstrate the effectiveness of the localized training strategy and the online update scheme. In addition, different choices of parameters (e.g., N_I, N_F, N_B) are studied and their influence on the accuracy of the sensitivity prediction are discussed.

We consider the same cantilever design problem shown in Fig. 4 with the same separated training and prediction sequence: we first perform the entire optimization for 200 steps to collect all data in the optimization history and, subsequently, use a subset of the collected data determined by parameters N_I, N_F, W_I, W_U to train/update the deep learning model. The rest of the data collected after the initial prediction ($N_I + W_I$) is used to evaluate the deviation angle θ_{error} defined in (16). We consider three fine-scale meshes and, unless otherwise stated, the block size N_B is taken to be 5 for all of them. The mesh statistics of the fine-scale meshes and their corresponding coarse-scale meshes are provided in Table 2. Throughout this section, we consider a fully-connected DNN with 4 hidden layers and each hidden layer consists of 1,000 neurons. To train that DNN, the maximum number of iterations is set to be 2,000 and the batch size is selected as 1,000. The decaying learning rate schedule is adopted in both initial training and online update stages — the initial learning rate is taken as $\ell_r = 0.0005$ and will be reduced every 500 iterations by a factor of 0.5.

5.1. Scalability and accuracy of the localized training

First, we demonstrate the scalability and accuracy of the local training strategy by comparing its performance with a global training strategy, in which we have only one training sample at each optimization step. For each training sample in the global training strategy, the input data is the global filtered design variable vector $\overline{\mathbf{z}}$ at a given optimization step and the output data is their corresponding global sensitivity vector $\overline{\mathbf{G}}$. There is no strain vector

Table 2
Statistics of the fine-scale meshes considered in the numerical assessments.

Meshes	# of fine-scale El.	# of fine-scale Nd.	# of coarse-scale El. ($NB = 5$)	# of coarse-scale Nd. ($NB = 5$)	# of DVs
Mesh 1	54,000	58,621	432	637	54,000
Mesh 2	128,000	136,161	1,024	1,377	128,000
Mesh 3	432,000	450,241	3,456	4,225	432,000

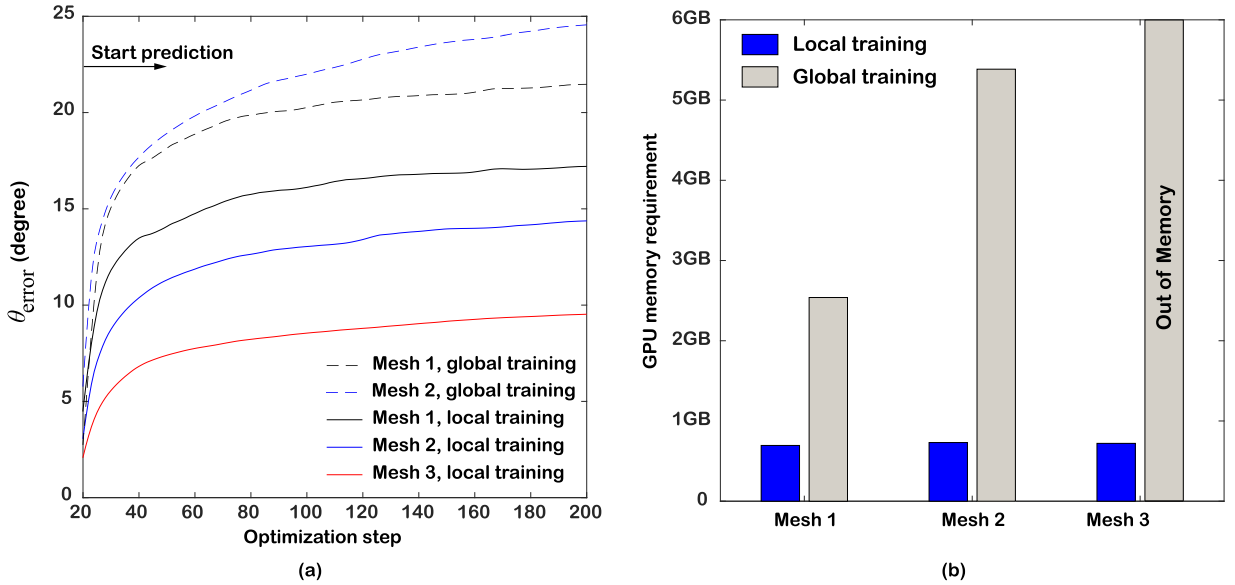

Fig. 6. (a) The evolution of θ_{error} as function of the optimization steps, after step 20, obtained by both local and global training for Mesh 1, Mesh 2, and Mesh 3. (b) The GPU memory used in the training of DNN by both local and global strategies for Mesh 1, Mesh 2, and Mesh 3. In (a), the results of the global training for Mesh 3 are not included because the GPU runs out of memory (i.e., the memory requirement is more than 12GB).

considered as input in the global training. To train the DNNs, we consider the one-time training (i.e., $N_F > 200$) with the parameter setup of $N_I = 10$ and $W_I = 10$. Under this setup, the data from optimization steps 10 to 19 are used to train the DNN and the prediction begins at optimization step 20. In Fig. 6(a), we show the evolution of θ_{error} as function of the optimization step after step 20 for both local and global training strategies. We also plot in Fig. 6(b) the GPU memory used in the training of DNN by both local and global strategies. For mesh 3, we could not obtain the results for global training because the GPU is out of memory, i.e., the memory requirement is more than 12GB. We can draw several conclusions from the comparison in those figures. First, within the same number of hidden layers and the identical number of neurons in each hidden layer, greater prediction accuracy is achieved by the DNN trained using the local strategy rather than the one trained by the global strategy, as shown in Fig. 6(a). Second, as shown in Fig. 6(b), as we increase the number of design variables, the local training strategy can maintain a constant GPU memory cost while the GPU memory required for the global training strategy increases drastically. This is because, as we increase the problem size (i.e., the number of the design variables), the number of training samples will increase but the size of the input and output data will stay the same in the local training strategy. However, in the global training strategy under the same situation, the number of training samples will not change but the sizes of the input and output data will increase accordingly. As a result of this feature, the local training can lead to a more scalable framework that could potentially work with topology optimization problem of any size.

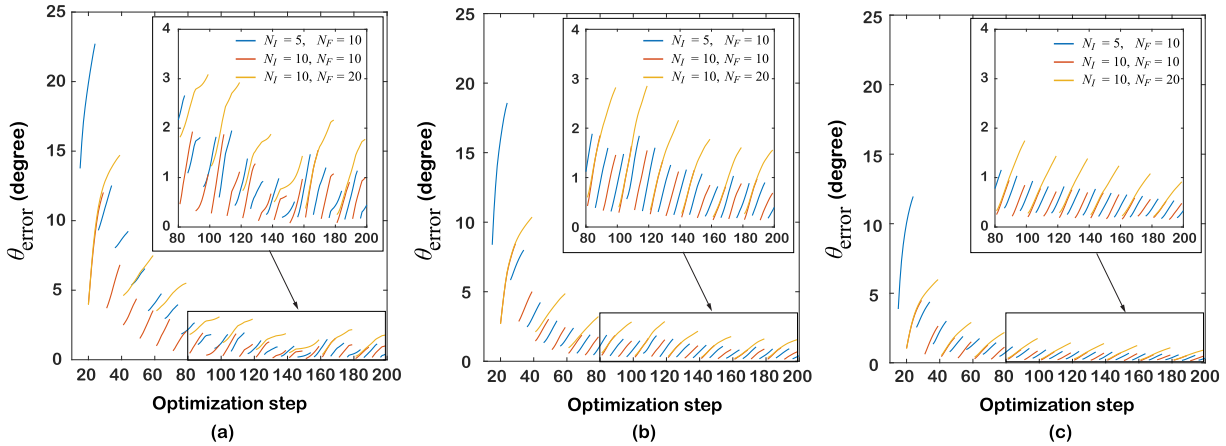

Fig. 7. The deviation angle θ_{error} as function of the optimization steps obtained by online update schemes with the 3 sets of parameters (i.e., $N_I = 5, N_F = 10$; $N_I = 10, N_F = 10$; and $N_I = 10, N_F = 20$) for (a) Mesh 1, (b) Mesh 2 and (c) Mesh 3.

5.2. *Influence of the online update and block sizes*

As demonstrated in Fig. 6(a), in one-time training, the deviation angle θ_{error} increases monotonously after the initial prediction step. This leads to an unsatisfactory level of long-term accuracy, meaning that, the predication accuracy of the DNN keeps deteriorating as we move away from those training steps. Thus, the online update scheme is necessary to improve the long-term accuracy of the prediction. To demonstrate this, we use the same data collected from the cantilever design example, and perform online update every N_F optimization steps after the initial prediction step. Three sets parameters are considered: (1) $N_I = 5$ and $N_F = 10$; (2) $N_I = 10$ and $N_F = 10$; and (3) $N_I = 10$ and $N_F = 20$. In Fig. 7(a)–(c), we plot the deviation angles θ_{error} as function of the optimization steps obtained with all the 3 sets of parameters for Mesh 1, Mesh 2 and Mesh 3, respectively. Notice that, in all the figures, we only plot the deviation angles in those optimization steps where the predictions are performed. Thus, each line in the figures is piece-wise continuous and consists of multiple intervals, each of which represents the evolution of deviation angle between either the initial training and the first online update or any two consecutive online updates. We can see from the figures that, for all the meshes and sets of parameters considered, the deviation angle θ_{error} is reduced drastically every time the online update is performed and, as a result, even though θ_{error} still increases monotonously within each interval, the maximum deviation angle of each interval keeps decreasing. This suggests that the predication accuracy of the DNN is constantly improved throughout the optimization history with the online update scheme. Moreover, by comparing the results obtained with different sets of N_Is and N_Fs, we also conclude that, both decreasing N_I and increasing N_F (which corresponds to decreasing the frequency of the online update) will increase the maximum deviation angle of each interval, although the overall decreasing trend of the deviation angle throughout the optimization remains.

In the last part of this section, we study the influences of the block size N_B on the prediction accuracy by considering four different block sizes values, i.e., $N_B = 4$, $N_B = 6$, $N_B = 10$, and $N_B = 15$. In this study, we restrict our attention to Mesh 3 and fix the parameters of the online update scheme as $N_I = 10$, $N_F = 10$, $W_I = 10$, and $W_U = 2$. In Fig. 8(a), we plot the deviation angle θ_{error} as function of the optimization step for those four block sizes values, which is obtained using the strain vector from the coarse-scale mesh as input (see discussions in Section 4.4.2). It is observed from the results that, the influence of block size on the predication accuracy is monotonous, especially in those steps right after initial training (the ones in the zoomed portion of Fig. 8(a)). As the block size N_B decreases, the prediction accuracy increases accordingly. In contrast, in Fig. 8(b), we plot the deviation angle θ_{error} as function of the optimization steps for the same set of N_B, which is obtained using the nodal displacement vector from the coarse-scale mesh as input (see discussion in Section 4.4.2). Unlike Fig. 8(a), the results in Fig. 8(b) suggest a non-monotonous behavior of the prediction accuracy as we decrease N_B. As illustrated in the zoomed portion of Fig. 8(b), the prediction accuracy first improves as we decrease N_B from 15 to 6 and then deteriorates as we decrease N_B from 6 to 4. If we extrapolate the above behaviors to the limit

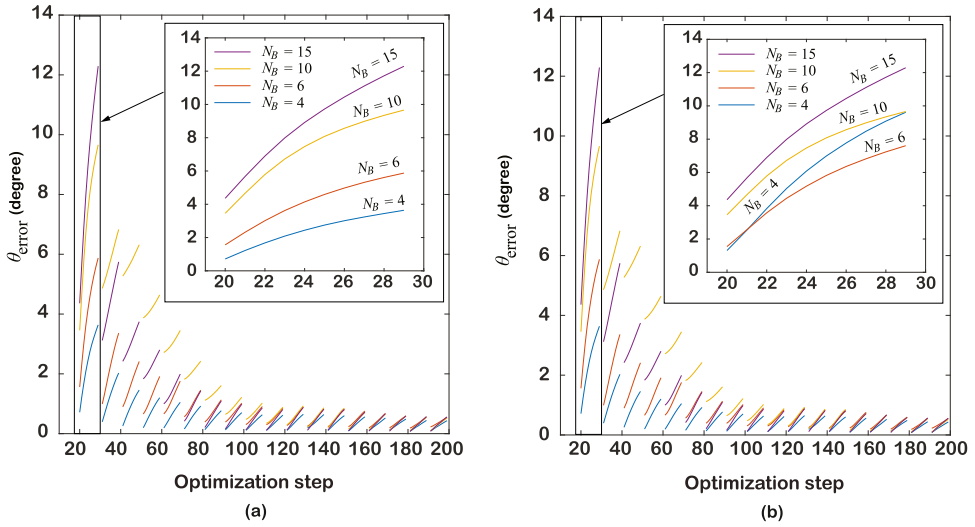

Fig. 8. (a) The deviation angle θ_{error} as function of the optimization steps for $N_B = 4$, 6, 10, and 15, which are obtained using the strain vector from the coarse-scale mesh as input. (b) The deviation angle θ_{error} as function of the optimization steps for $N_B = 4$, 6, 10, and 15, which are obtained using the nodal displacement vector from the coarse-scale mesh as input.

case of $N_B = 1$, we will obtain consistent behaviors observed in the numerical study of Section 4.4.2. Thus, this study provides additional numerical evidence to support that *we should always use the strain vector rather than the nodal displacement vector from the coarse-scale mesh as the input to the machine learning module of the proposed framework*.

6. Design examples

Having gained sufficient understanding of the proposed framework and its proper parameter setup, this section presents four design examples to showcase the effectiveness and significant speedup of the proposed framework as compared to the standard topology optimization. We emphasize that, unlike any other machine learning-based approaches in the literature of topology optimization, our proposed framework can be universally applied to any design problem without any pre-collected information about that problem. For all the design examples, the Young's modulus and Poisson's ratio for the solid material are chosen as $E_Y = 1$ and $\nu = 0.3$, respectively, the move limit m is taken as $m = 0.1$, and the damping coefficient η is set as $\eta = 0.5$. In terms of our specific implementation, two Titan Xp GPUs, each having 12GB of memory, are used. One is used to solve the state equations on both coarse-scale and fine-scale meshes. The Preconditioned Conjugate Gradient (PCG) method with the Jacobi preconditioner is adopted. The other GPU is used to train the deep learning model. In this section, in order to distinguish the designs produced by the proposed framework from the ones by the standard framework, we plot the 3D designs obtained by the proposed framework in red color and the ones obtained by the standard approach in gray color.

6.1. Example 1: An illustrative example in 2D

We intend to use the first 2D example as an illustration of the proposed framework. To that end, we consider a 2D Messerschmitt–Bölkow–Blohm (MBB) beam design domain whose dimensions, load and support conditions are given in Fig. 9. The magnitude of the load is $F = 2$. Due to symmetry, we consider half of the MBB domain in our implementation with the appropriate boundary conditions and discretize it with a 1,920 by 640 fine-scale mesh consisting of 1228,800 elements. The block size N_B is taken to be 5, leading to a 364 by 148 coarse-scale mesh with 53,872 elements. As for the optimization parameters, the radius of the density filter is taken to be $R = 0.008$ and the volume fraction is set as $\overline{V} = 50\%$. We adopt a continuation scheme for the penalization parameter p, in which p is increased from 1 to 3 with increment 1. The maximum optimization step number for $p = 1$ and $p = 2$ is set as 20, whereas the one for $p = 3$ is chosen as 200. We start to apply the proposed framework at $p = 3$.

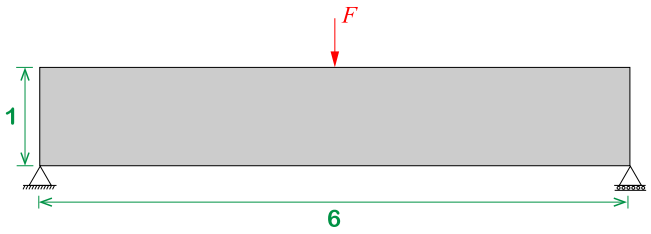

Fig. 9. The dimensions, load, and boundary conditions of the 2D MBB domain.

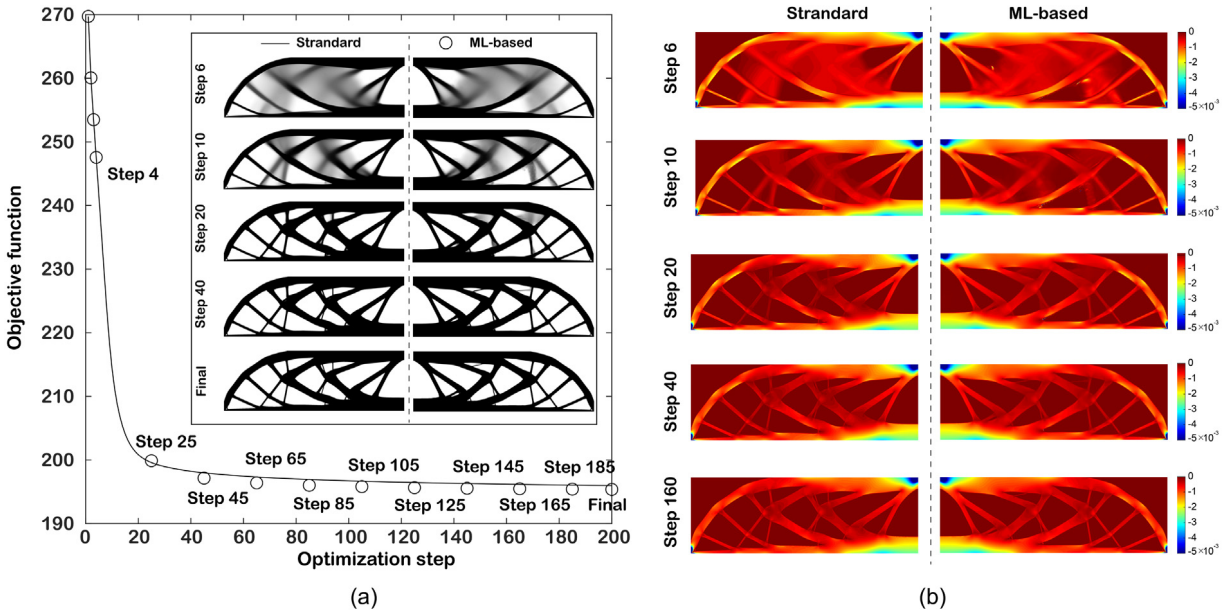

Fig. 10. Comparison between the standard and proposed approaches: (a) convergence history of the objective function, and (b) sensitivities with respect to the filtered design variable.

The associated parameters are chosen as $N_I = 1$, $N_F = 20$, $W_I = 4$ and $W_U = 4$. As for the machine learning model, we utilize a fully-connected DNN with 4 hidden layers, each of which has 200 neurons. For both online training and updating, the maximum epoch number is taken to be 2,000 and the batch size is chosen as 30,000. During the training, we utilize a decaying learning rate schedule: the initial learning rate is set to $\ell_r = 0.0005$ and is subsequently reduced by half every 400 epochs until the minimum $\ell_r = 1 \times 10^{-5}$ is reached.

In Fig. 10(a), we show the convergence history of the objective function obtained by the standard approach and proposed framework for the optimization steps starting with $p = 3$. It is immediate from the comparison that the proposed framework can lead to a design with almost identical topology and objective value (195.38 (proposed) v.s. 195.98(standard)) to the ones obtained by the standard approach. The proposed framework, however, only solves the fine-scale system 14 times while the standard approach solves the fine-scale systems 200 times (for $p = 3$). Moreover, in Fig. 10(b), we present fringe plots of the sensitivities of the objective function with respect to the filtered design variables for several intermediate steps obtained with the standard approach (left column) and the proposed framework (right column). In the proposed framework, the sensitivity plots are directly predicted by the DNN. Comparing the sensitivity plots, we can draw several conclusions. First, the proposed online training and updating scheme produce sensitivities that are sufficiently accurate as compared to true ones. Second, with the online updating scheme, the predicted sensitivity becomes more accurate throughout the optimization process. For instance, it is apparent that the sensitivity predicted at optimization step 160 (with several online updating) is more accurate than the one predicted at optimization step 6 (with no online updating).

Table 3
Statistics of the fine-scale and coarse-scale meshes considered in the 3D cantilever design problem.

Meshes	# of fine-scale El.	# of fine-scale Nd.	# of coarse-scale El. ($NB = 5$)	# of coarse-scale Nd. ($NB = 5$)	# of DVs
Mesh 1	85,750	92,016	686	960	85,750
Mesh 2	250,000	262,701	2,000	2,541	250,000
Mesh 3	1458,000	1498,861	11,664	13,357	1458,000

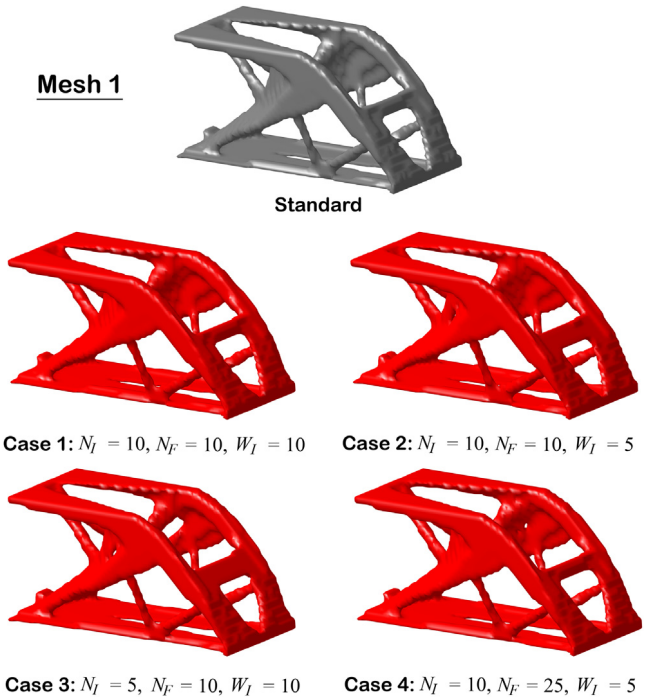

Fig. 11. The final topologies obtained on Mesh 1 (see Table 3) with the four cases of parameter setup for the cantilever design problem.

6.2. Example 2: Benchmark 3D cantilever beam design

The cantilever design problem illustrated in Fig. 4 is considered and, through this example, we aim to demonstrate the scalability and accuracy of the proposed machine learning-based framework for meshes with different sizes. To that end, we consider three fine-scale meshes with different levels of refinement and set the block size N_B to be 5 for all of them. The detailed statistics of these three meshes are summarized in Table 3. In addition, We will also discuss in this example how different parameters choices (e.g., N_I and N_F) affect the performance of the proposed machine learning-based framework. To achieve this, we consider four cases with different parameter setup as follows:

- **Case 1**: $N_I = 10$, $N_F = 10$ and $W_I = 10$;
- **Case 2**: $N_I = 10$, $N_F = 10$ and $W_I = 5$;
- **Case 3**: $N_I = 5$, $N_F = 10$ and $W_I = 10$;
- **Case 4**: $N_I = 10$, $N_F = 25$ and $W_I = 10$.

For all the cases, the parameter W_U, which is the window size for online update, is taken to be 2 and the drop out rate for void sample P_{drop} is set as 0.9. Moreover, the fully-connected DNN is trained or updated using batch sizes of 500, 1,000 and 10,000 for Mesh 1, Mesh 2, and Mesh 3, respectively.

In Figs. 11–13, we depict the final structural topologies obtained on Mesh 1, Mesh 2, and Mesh 3, respectively. For each mesh, the results include the designs obtained using the proposed approach with four cases of parameters and the design obtained from the standard topology optimization. Comparing the overall layouts between the

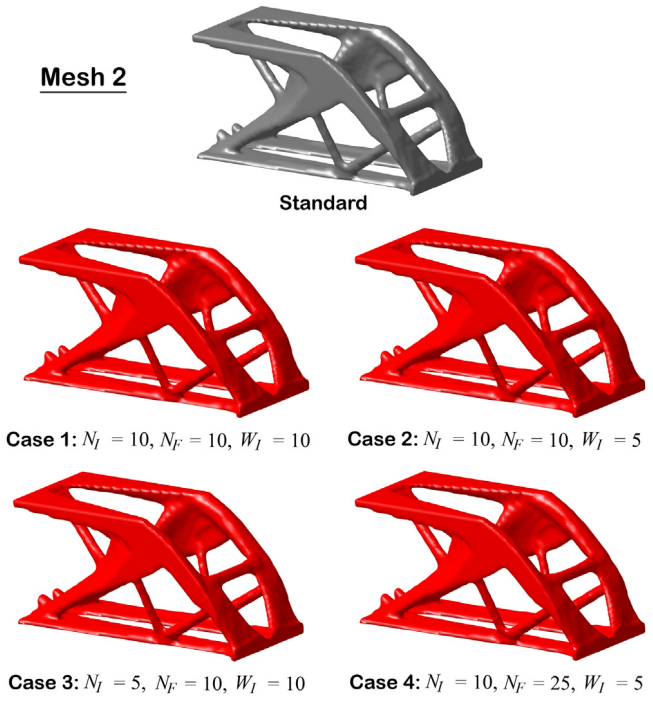

Fig. 12. The final topologies obtained on Mesh 2 (see Table 3) with the four cases of parameter setup for the cantilever design problem.

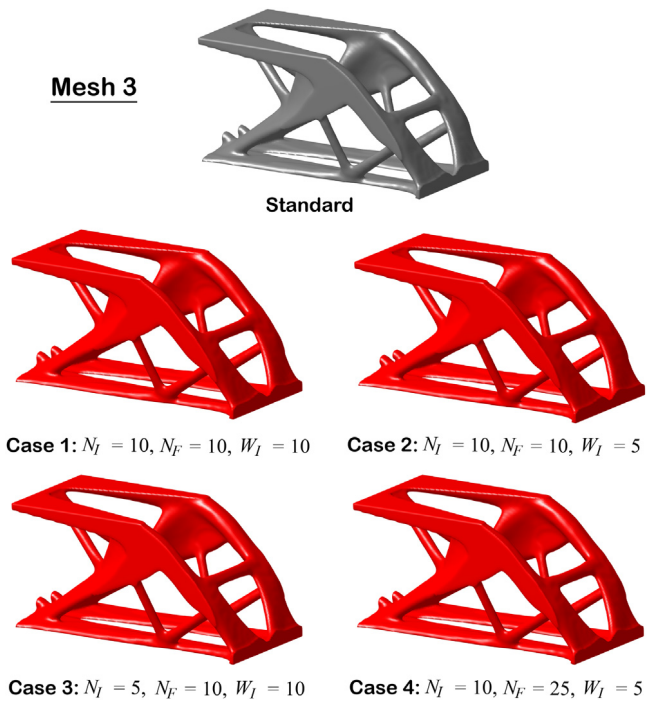

Fig. 13. The final topologies obtained on Mesh 3 (see Table 3) with the four cases of parameter setup for the cantilever design problem.

designs, we obtain from the proposed and standard approaches, we conclude that the proposed machine learning-based approach can yield structural designs which closely resemble the ones produced by the standard topology

Table 4
Summary of the quantitative comparison between the results obtained by the proposed machine learning-based and standard topology optimization approaches for the 3D cantilever beam design problem. The comparison is in terms of both accuracy and computational cost.

		Standard	Case 1	Case 2	Case 3	Case 4
Mesh 1	Final Objective	370.11	370.023	371.14	371.23	368.61
	Difference	0%	−0.023%	0.28%	0.3%	−0.4%
	ML time (s)	0	499	469.78	560	289
	Total time (s)	1,346	1,248	1,160	1,250	924
Mesh 2	Final Objective	379.54	379.67	379.53	379.57	379.37
	Difference	0%	0.03%	−0.002%	0.009%	−0.046%
	ML time (s)	0	787	769	876	451
	Total time (s)	3,665	2,115	1,991	2,105	1,515
Mesh 3	Final Objective	383.51	383.84	384.12	383.76	383.75
	Difference	0%	0.085%	0.159%	0.065%	0.062%
	ML time (s)	0	1,947	1,842	2,229	1,193
	Total time (s)	25,455	8,710	7,954	8,331	6,069

optimization procedure for a wide range of mesh sizes. We also notice that, as the mesh becomes finer, the designs produced by the machine learning-based and standard topology optimization become more similar to each other. For example, for Mesh 1, which is the coarsest mesh considered, noticeable differences in structural layout can still be found among the designs. However, no visual difference is observed for Mesh 2 and Mesh 3 between the designs obtained by the proposed framework and standard one.

Furthermore, Table 4 provides quantitative comparisons between the proposed machine learning-based and standard approaches in terms of accuracy (with respective to the final objective value) and the computational cost. From the accuracy perspective, we notice that the final designs obtained from the proposed machine learning-based approach have almost identical objective values to the ones produced by the standard topology optimization procedure — for all the meshes and all four cases considered, the differences are within 0.5%. We also notice that, the variation of the percentage differences between the four cases is less for Mesh 3 and Mesh 2 than for Mesh 1, implying that, as the mesh becomes more refined, the performance of the proposed approach becomes more stable and less sensitive to different choices of parameters. This conclusion is in agreement with our previous observation from the qualitative comparison of the designs on different meshes (c.f. Figs. 11–13).

Regarding the computational efficiency, we show in Fig. 14 the total computational time spent by the proposed and standard frameworks for the 4 cases on Mesh 1, Mesh 2 and Mesh 3. In conjunction with Table 4, we conclude that the proposed machine learning-based framework is in general more efficient than the standard topology optimization. For instance, with the parameter setup in Case 4, we gain more than 4 times of speedup with the proposed machine learning-based approach on Mesh 3. In addition, we see a clear trend that, the larger the mesh is, the more speedup is achieved. For smaller meshes, although the machine learning model avoids solving the state equations in many optimization steps, training the machine learning model takes the majority of the computational time saved. Thus the net saving is not substantial. However, this is not the case for finer meshes as training the machine learning model is relatively more efficient than solving the state equations.

To further quantify the computational efficiency of the optimization process, we plot in Fig. 14(a)–(c) the convergence history of the objective function versus the total computational time for Mesh 1, Mesh 2, and Mesh 3, respectively. In those figures, for a given objective value in the y axis, the value in the x axis corresponds to the total computational time spent to reach that value. For the proposed framework, the time spent in training the deep learning model is also included. Those figures can provide insight into the computational efficiency of the optimization process by measuring how much computational time each approach needs in order to reach a certain objective value. For Mesh 1, we notice that the convergence history of the objective values obtained by the machine learning-based framework (i.e., the circular markers) falls slightly above the one by the standard topology optimization (i.e., the solid line) except for the Case 4. This suggests that, for Mesh 1, the proposed machine learning-based framework is in fact a slightly less efficient than the standard one in its optimization process. Even

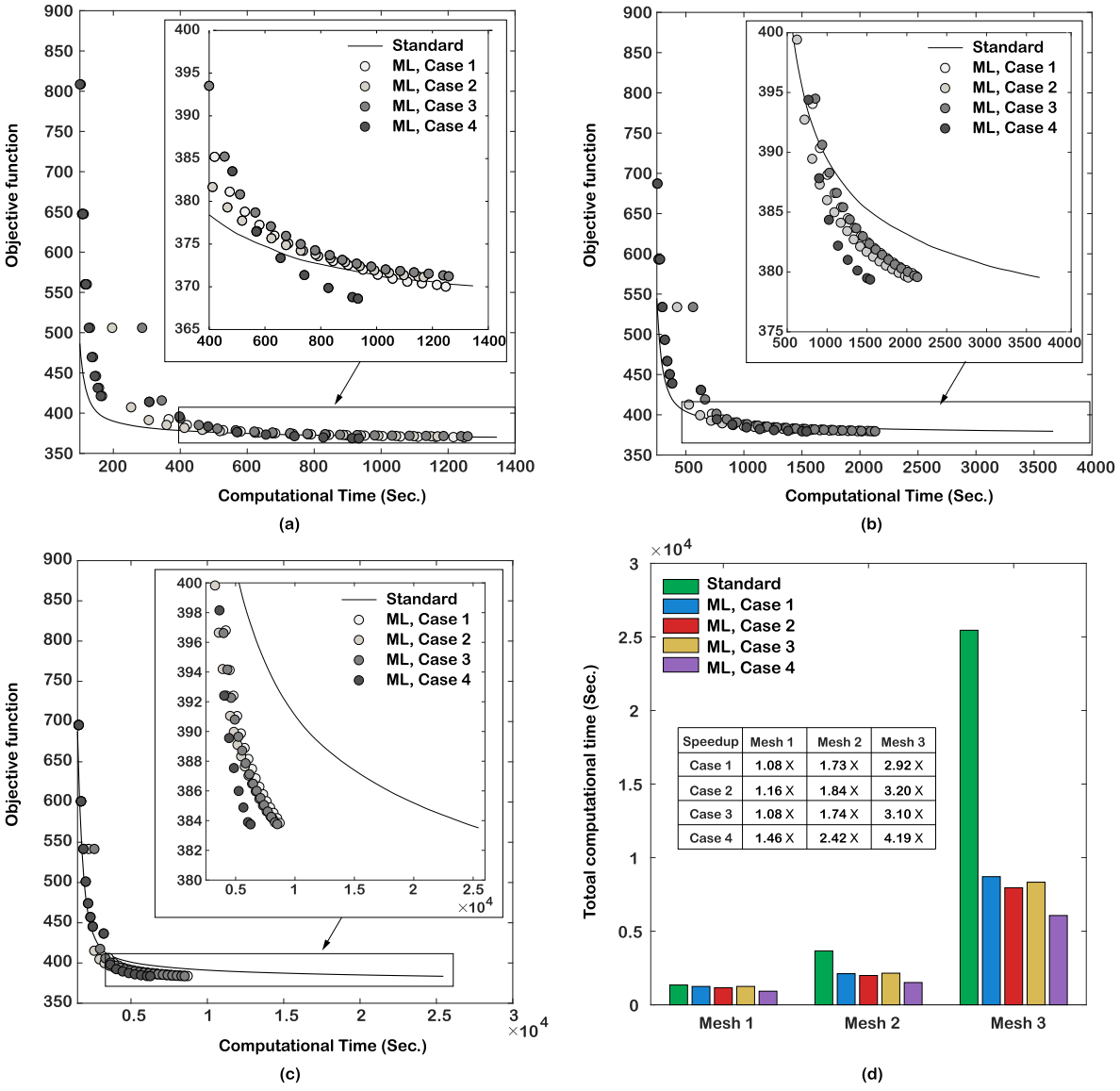

Fig. 14. Convergence history for the 3D MBB beam design problem in terms of the objective function versus the total computational time (in seconds) for: (a) Mesh 1, (b) Mesh 2, and (c) Mesh 3. For the machine learning-based framework, the total computational time includes the time for performing the two-scale topology optimization as well as the time for training the deep learning model. (d) The total computational time spent by the proposed and standard frameworks for the 4 cases on Mesh 1, Mesh 2 and Mesh 3. The speedups obtained by the proposed framework as compared to the standard one in the four cases are summarized in (d) as well.

though the total time is slightly less for the proposed framework, the amount of time saved is not sufficient to compensate the difference in objective values during the convergence history. On the contrary, for Mesh 2 and Mesh 3, we can see that the convergence histories of the objectives values obtained by the machine learning-based framework are below the ones by the standard topology optimization for all the cases, indicating that the proposed framework does achieve a higher computational efficiency during its optimization process. To summarize, we conclude from this example that the main feature of the proposed machine learning-based framework is that it becomes more efficient and accurate for finer meshes, which makes it a suitable framework for large-scale topology optimization problems.

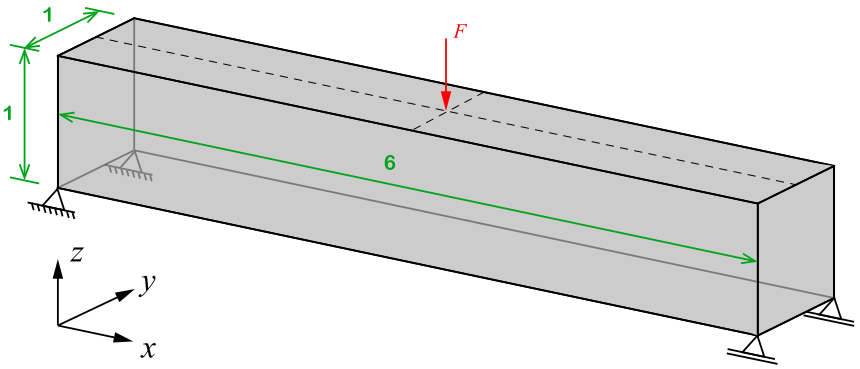

Fig. 15. The MBB design domain and its dimensions, load and boundary conditions.

Table 5
Statistics of Mesh 1 and Mesh 2 together with their associated coarse-scale meshes considered in the 3D MBB beam design problem.

	Mesh 1			Mesh 2		
	N_B	# of elements	# of nodes	N_B	# of elements	# of nodes
Fine-scale		324,000	342,271		1327,104	1373,617
Coarse-scale	5	2,592	3,367	4	20,736	23,725
	6	1,500	2,046	6	6,144	7,497
	10	324	532	8	2,592	3,367
	15	96	195	12	768	1,125

6.3. Example 3:3D MBB beam design

The main goal of the third design example is to study the impact of various choices of block size N_B and void sample dropout rate P_{drop} on the training efficiency, prediction accuracy, and the topology optimization performance. To that end, we consider a MBB design example whose dimensions and boundary conditions are shown in Fig. 15. Because of symmetry, a quarter of the design domain is simulated. In terms of the design parameters, we assume that the allowable volume fraction is $V_{\max} = 12\%$ and set the radius of the density filter to be $R = 0.08$. This example consists of two parts. The first part fixes the parameters N_I, N_F, W_I, and W_U and studies the impact of various choices of block size N_B and void sample dropout rate P_{drop} on the training efficiency, prediction accuracy, and the performance of the proposed framework. The second part discusses the trade-off between small and large block sizes and discusses how N_B impacts the choices of N_I, N_F, W_I and W_U as well as the overall computational efficiency.

In the first part, the domain is discretized by two fine-scale meshes: Mesh 1 with 324,000 elements and Mesh 2 with 1373,617 elements. Four block size values are considered for each mesh. For Mesh 1, the block sizes are chosen to be $N_B = 5$, $N_B = 6$, $N_B = 10$ and $N_B = 15$ and; for Mesh 2, the block sizes are adopted as $N_B = 4$, $N_B = 6$, $N_B = 8$ and $N_B = 12$. The statistics of both fine-scale meshes and their corresponding coarse-scale meshes are summarized in Table 5. In additional to various block size values, we also consider two void sample drop-out rates: $P_{\text{drop}} = 0.9$, which corresponds to dropping out void samples with a probability of 90%, and $P_{\text{drop}} = 0$, which corresponds to keeping all the void samples. The parameters N_I, N_F, W_I and W_U are chosen to be $N_I = 5$, $N_F = 10$, $W_I = 10$ and $W_U = 4$.

We utilize a DNN with 4 hidden layers and 1,000 neurons per hidden layer. For each mesh and block size, the hyper-parameters of the DNN are summarized in Table 6. Decaying learning rate schedules are adopted in both initial training and online update stages: the initial learning rate is reduced by half every 400 and 500 epochs, respectively for mesh 1 and mesh 2, until a minimum value 10^{-5} is reached.

In Figs. 16(a)–(b) and 17(a)–(b), we plot the convergence histories of objective function obtained from the machine learning-based framework for mesh 1 and mesh 2, respectively. The convergence histories of the objective function by the standard topology optimization approach are also included for comparison purposes. Additionally,

Table 6
Summary of hyper-parameters for the DNNs adopted for the 3D MBB beam design problem.

Mesh 1				Mesh 2			
N_B	ℓ_r	Max. epochs	Batch size	N_B	ℓ_r	Max. epochs	Batch size
5	0.005	2,000	1,500	4	0.005	3,000	10,000
6	0.005	2,000	1,000	6	0.005	3,000	3,000
10	0.005	2,000	300	8	0.005	3,000	1,000
15	0.005	2,000	80	12	0.005	3,000	400

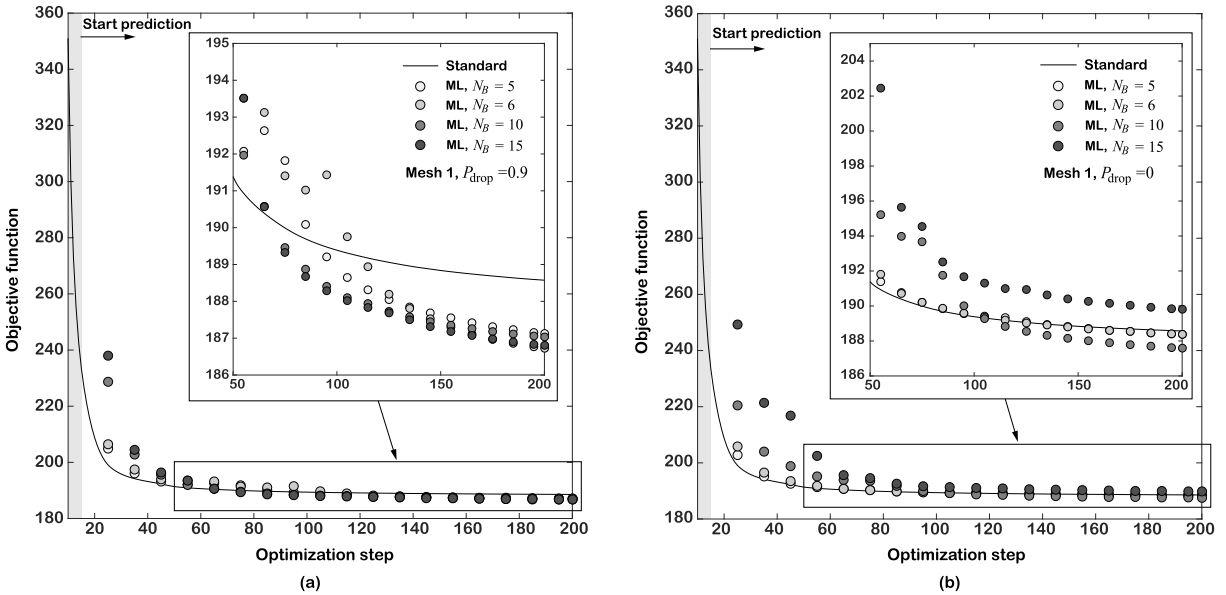

Fig. 16. Convergence history for the 3D MBB beam design problem in terms of the objective obtained from the machine learning-based framework with four different N_B values for Mesh 1 with: (a) $P_{\text{drop}} = 0.9$, (b) $P_{\text{drop}} = 0$.

Figs. 18 and 19 depict the optimal designs obtained from both machine learning-based and standard approaches for Mesh 1 and Mesh 2, respectively, with four different N_B values and $P_{\text{drop}} = 0.9$.

Several observations are made. First, we find that performance of the proposed machine learning-based framework is insensitive to different choices of block sizes N_B for both meshes. As shown in Table 7, the differences in final objective values between the machine learning-based and standard approaches typically stay within 0.5%. Larger N_B values, however, seem to lead to larger oscillations in the convergence history of the objective function. In most cases, the machine learning-based framework even produces designs with smaller objective values than the standard one, suggesting that it has the potential of driving the optimization to a better local minima. On the other hand, from qualitative comparison of the designs, we conclude that different choices of N_B all lead to optimal designs that closely resemble the ones obtained by the standard topology optimization for both Mesh 1 and Mesh 2, while the machine learning-based approach with larger N_B values is likely to produce designs without thin structural members. Second, from a computational efficiency perspective, the machine learning-based framework is shown to be able to greatly accelerate the design process and the total speedup increases as the mesh becomes larger, which is in agreement with the first design example. In addition, the larger the block size N_B is, the more speedup the machine learning-based framework can introduce as a result of the smaller coarse-scale systems and the smaller sample size. For example, by choosing $N_B = 12$ and $P_{\text{drop}} = 0.9$, the machine learning-based framework can accelerate the design process more than 4 times on Mesh 2, whereas the speedup is more than 2 when considering $N_B = 4$ and $P_{\text{drop}} = 0.9$ instead. Third, we conclude that the randomly dropping out void samples from the training set improves both the accuracy and training efficiency of the machine learning-based framework. In terms of the accuracy, we notice from the final objective values in Table 7 that choosing $P_{\text{drop}} = 0.9$ leads to less variations of

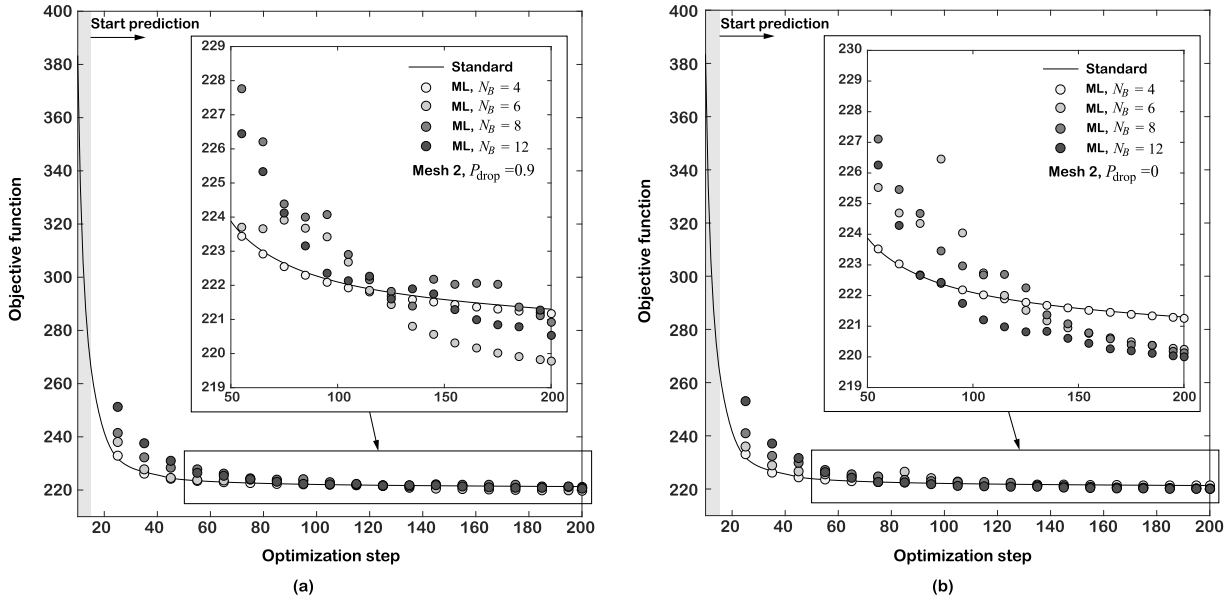

Fig. 17. Convergence history for the 3D MBB beam design problem in terms of the objective obtained from the machine learning-based framework with four different N_B values for Mesh 2 with: (a) $P_{\text{drop}} = 0.9$, (b) $P_{\text{drop}} = 0$.

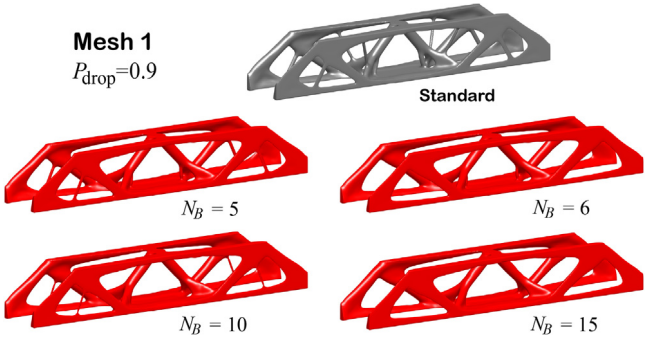

Fig. 18. Optimal MBB beam design obtained from the machine learning-based and standard approaches for Mesh 1. The results obtained from the machine learning-based approach utilize the void sample drop out rate $P_{\text{drop}} = 0.9$.

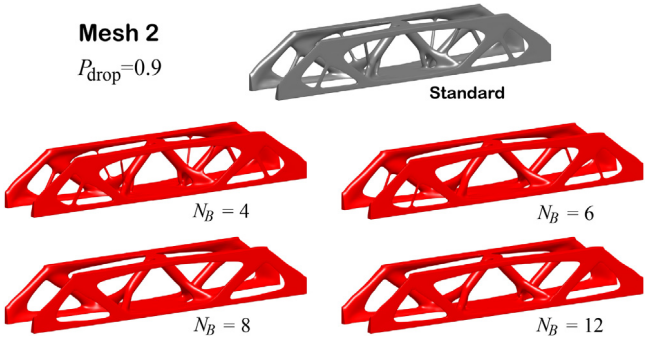

Fig. 19. Optimal MBB beam design obtained from the machine learning-based and standard approaches for Mesh 2. The results obtained from the machine learning-based approach utilize the void sample drop out rate $P_{\text{drop}} = 0.9$.

Table 7
Summary of comparison between the results obtained for the 3D MBB beam design problem by standard topology optimization and machine learning-based approach, and their respective computational times for Mesh 1 and Mesh 2.

Mesh 1		Standard	$N_B = 5$	$N_B = 6$	$N_B = 10$	$N_B = 15$
$P_{\text{drop}} = 0.9$	Final Objective	188.57	186.58	186.72	187.03	186.81
	Difference	0%	−1.06%	−0.98%	−0.82%	−0.94%
	ML time (s)	0	1,552	1,148	684	826
	Total time (s)	7,529	3,297	2,704	2,211	2,338
$P_{\text{drop}} = 0$	Final Objective	188.57	188.39	188.38	187.59	189.81
	Difference	0%	−0.099%	−0.10%	−0.52%	0.65%
	ML time (s)	0	2,626	1,930	1,016	982
	Total time (s)	7,529	4,242	3,508	2,652	2,599
Mesh 2		Standard	$N_B = 4$	$N_B = 6$	$N_B = 8$	$N_B = 12$
$P_{\text{drop}} = 0.9$	Final Objective	221.30	221.17	219.78	220.92	220.54
	Difference	0%	−0.06%	−0.69%	−0.17%	−0.35%
	ML time (s)	0	9,244	3,982	3,023	1,942
	Total time (s)	43,206	18,205	12,531	11,382	10,299
$P_{\text{drop}} = 0$	Final Objective	221.30	221.26	220.25	220.13	220.00
	Difference	0%	−0.018%	−0.48%	−0.53%	−0.59%
	ML time (s)	0	17,645	7,352	5,067	2,785
	Total time (s)	43,206	26,634	15,851	13,461	11,008

final objective values when different N_B values are considered, especially for Mesh 1, than choosing $P_{\text{drop}} = 0$. This suggests that, for the present example, randomly dropping out void training samples can help stabilize the training process and improve the predication accuracy of the DNN as compared to keeping all the void samples in the training set. In terms of the training efficiency, Figs. 20 and 21 show the evolution of the training set size and training time of the machine learning module during the optimization under various N_B and P_{drop} for Mesh 1 and Mesh 2, respectively. We conclude that, comparing to keeping all the void instances (i.e., $P_{\text{drop}} = 0$) in the training set, randomly dropping out most of them (i.e., $P_{\text{drop}} = 0.9$) from the training set can greatly reduce the sample size and therefore the total time to train the DNN. The smaller the block size N_B is, more reductions are achieved. This suggests that, in practice, one should always perform random drop out of void samples with a large probability P_{drop}, such as $P_{\text{drop}} = 0.9$ (adopted in this section), in the machine learning-based framework to ensure both the accuracy and efficiency.

Moreover, we segment the computational time of the proposed framework into four main portions: (1) fine-scale solver, (2) coarse-scale solver, (3) Optimizer (i.e., OC), and (4) online training and update; and report the detailed breakdown of computational time into each segment (Fig. 22). As shown in Fig. 22, solving the fine-scale systems plus the online training and update make up the most of computational cost for both Meshes 1 and 2. As we increase the block size values, the time for online training and update is reduced accordingly. As expected, the time associated with the coarse-scale solver is significantly less than the one spent by the fine-scale solver, especially for the larger mesh. Lastly, the computational time associated with the optimizer constitutes a considerably larger portion of the total time in Mesh 2 than in Mesh 1, implying the need of a parallel implementation of the optimizer [54] for larger meshes in our proposed framework.

We noticed that there exists a trade-off between small and large block sizes. Thus, in this second part of this example, we illustrate this trade-off and discuss how it impacts the choices of parameters N_I, N_F, W_I and W_U as well as the computational efficiency of the proposed framework. From now on, we restrict our attention to Mesh 2 and consider three block size values, $N_B = 2$ (small), $N_B = 4$ (intermediate) and $N_B = 8$ (large). In general, as the N_B becomes larger, we need a DNN with larger model capacity and more frequent online updating to maintain sufficiently accurate performance. We find the following parameters and hyper-parameters setups are suitable for each choice of block size:

- For $N_B = 2$, we use a fully-connected DNN with 4 hidden layers and 200 neurons per hidden layer. The initial learning rate is set as $\ell = 0.001$ and the batch size is taken as 15,000. the parameters N_I, N_F, W_I and W_U are chosen as 7, 50, 3 and 1, respectively.

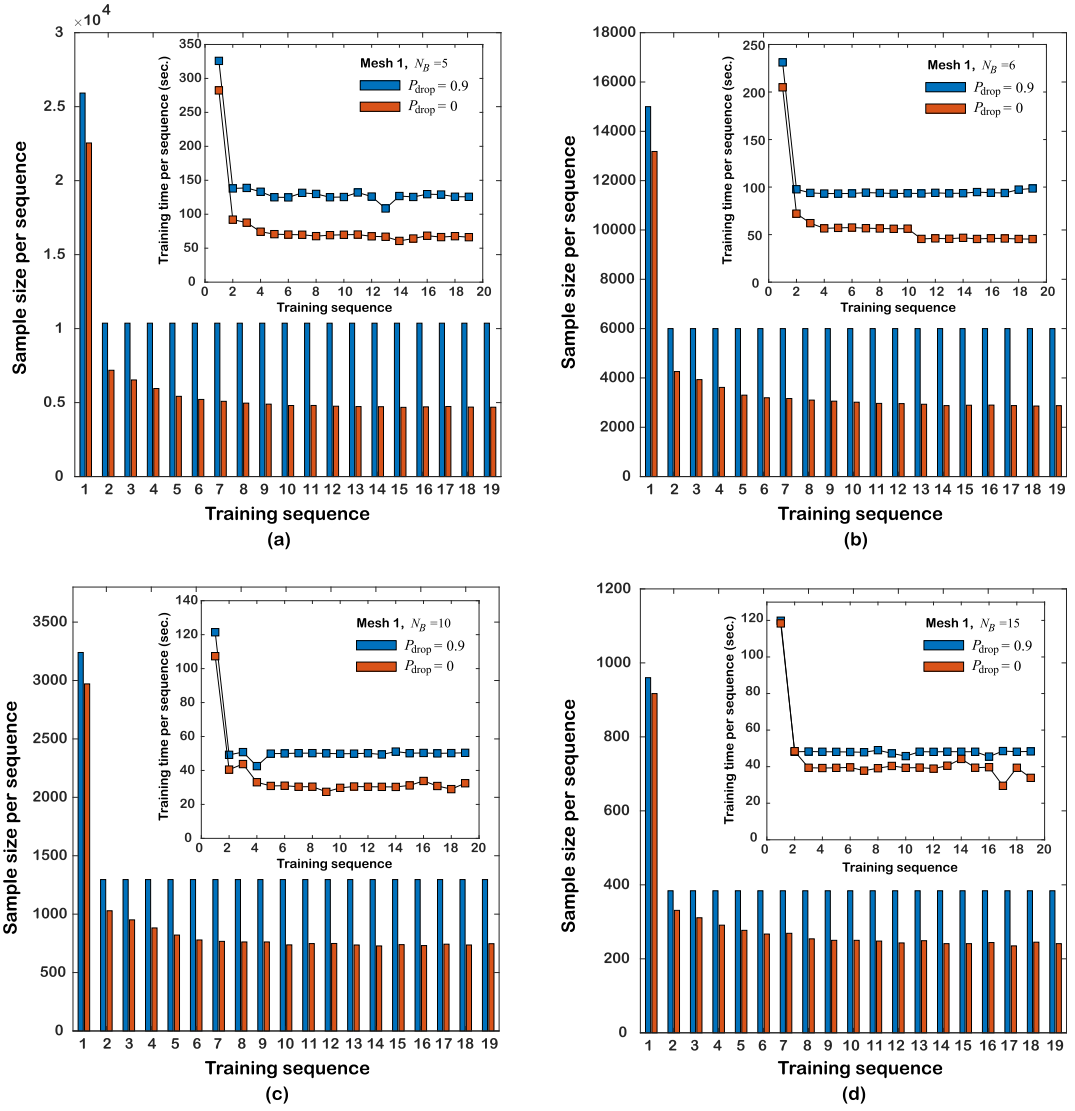

Fig. 20. The evolution of sample size and training time for training/updating the DNN throughout the optimization with both $P_{\text{drop}} = 0.9$ and $P_{\text{drop}} = 0$ for Mesh 1 considering the 3D MBB beam design problem. The block size for each figure is: (a) $N_B = 5$, (b) $N_B = 6$, (c) $N_B = 10$, and (d) $N_B = 15$.

- For $N_B = 4$, we use a fully-connected DNN with 4 hidden layers and 500 neurons per hidden layer. The initial learning rate is set as $\ell = 0.0005$ and the batch size is taken as 8,000 and the parameters N_I, N_F, W_I and W_U are chosen as 5, 45, 5 and 2, respectively.
- For $N_B = 8$, we use a fully-connected DNN with 4 hidden layers and 1,000 neurons per hidden layer. The initial learning rate is set as $\ell = 0.0005$ and the batch size is taken as 1,000 and the parameters N_I, N_F, W_I and W_U are chosen as 5, 10, 10 and 4, respectively.

For all the cases, the maximum epoch is 3,000 and the initial learning rate is reduced by half until a minimum of $\ell = 10^{-5}$ is reached.

Fig. 23(a)–(c) present the convergence history versus computational time, the total computational cost, and the final designs for both proposed framework and the standard approach. While the proposed framework with all the block sizes yields designs with identical compliances, the intermediate block size $N_B = 4$ is considerably more efficient than the other two. This is explained by Table 8. According to the table, while the proposed framework

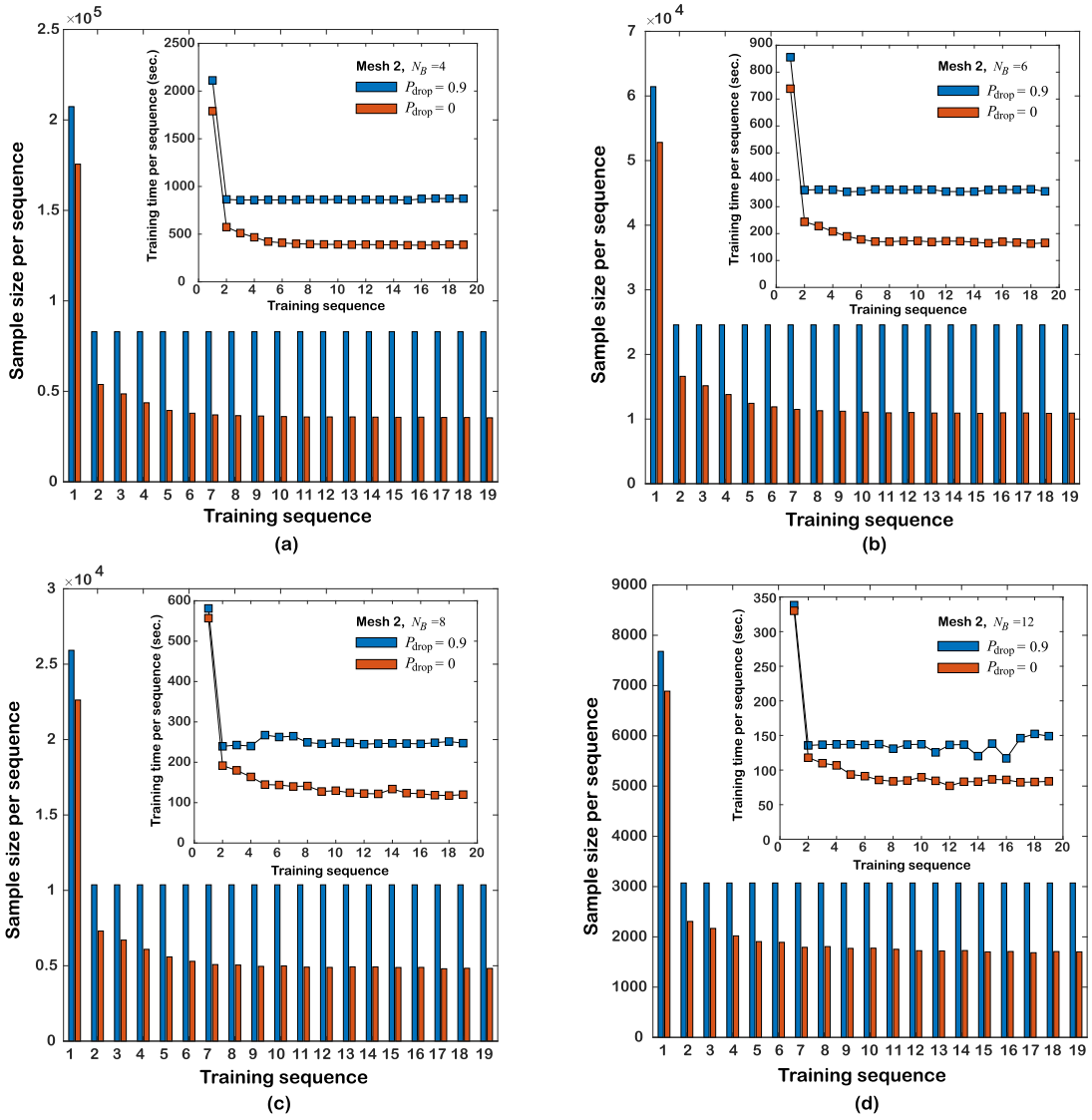

Fig. 21. The evolution of sample size and training time for training/updating the DNN throughout the optimization with both $P_{\text{drop}} = 0.9$ and $P_{\text{drop}} = 0$ for Mesh 2 considering the 3D MBB beam design problem. The block size for each figure is: (a) $N_B = 4$, (b) $N_B = 6$, (c) $N_B = 8$, and (d) $N_B = 12$.

with $N_B = 2$ uses the least frequent online updating, its costs considerably more in solving the coarse-scale state equations. Additionally, online training and updating also spend more time in $N_B = 2$ than $N_B = 4$ due to a larger training sample size. On the contrary, while $N_B = 8$ is the most efficient in solving the coarse-scale systems, it requires much more frequent online updating than the other two N_Bs to main sufficient prediction accuracy, leading to significantly more time in online training & updating and solving the fine-scale systems. The intermediate choice $N_B = 4$ provides most balanced efforts in online training and solving the coarse-scale system and, thus, leads to the most speedup in computational time (i.e., 9.2 versus 3.8 in other two choices). This example suggests that, to maximize the performance of the proposed framework, a proper choice of block size N_B should be neither too large nor too small.

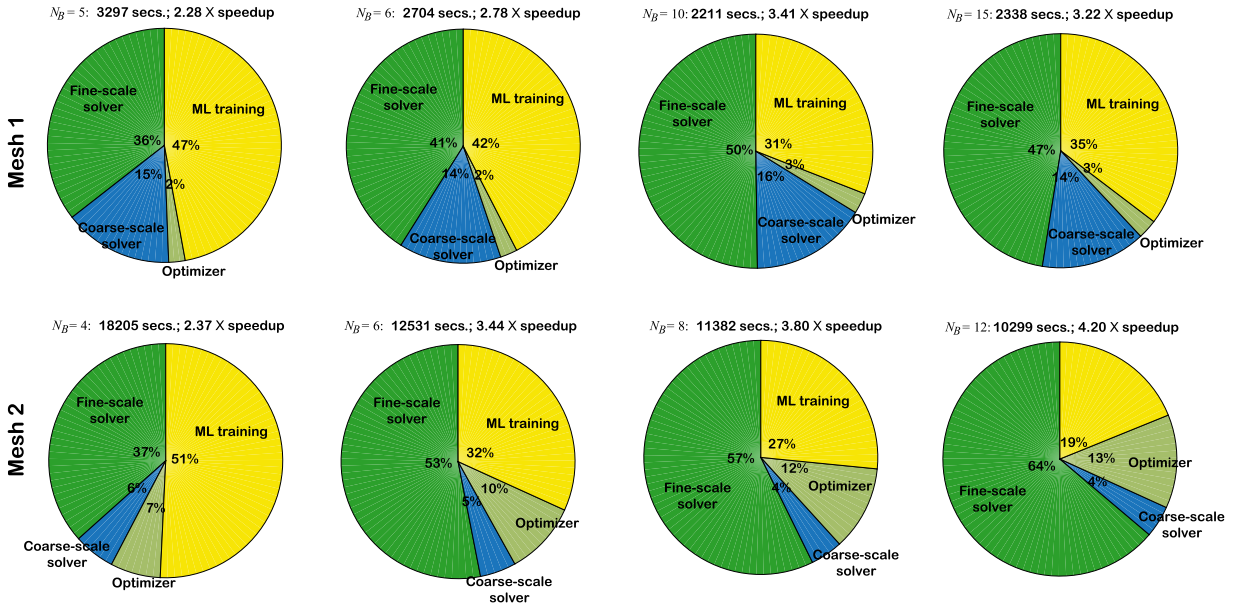

Fig. 22. Breakdown of the total computational time for the 3D MBB beam design problem considering Mesh 1 and Mesh 2 with various choices of block sizes N_B.

Table 8
Breakdown of the total computational time for the 3D MBB beam design problem into four segments: fine-scale FE solver, coarse-scale FE solver, design update, and online training.

Time break down (s)	Standard	$N_B = 2$	$N_B = 4$	$N_B = 8$
Fine-scale FE	42,047	2,159	2,141	6,521
Coarse-scale FE	0	6,705	1,045	497
Update	1,160	1,162	1,067	1,330
ML	0	1,270	446	3,023
Total	43,207	12,296	4,699	11,371

6.4. Example 4: A 3D bridge design

Non-designable regions are typical in engineering designs but add additional challenges to the proposed machine learning-based framework because they are never explicitly defined on the coarse-scale mesh. Thus, one goal of this last design example is to showcase the potential of the proposed machine learning-based framework to accurately capture the presence of non-designable regions with various block size N_B. Moreover, this design example also aims to study the influence of various choices of hyper-parameters of the DNN, like the batch size and the maximum training iteration, on the performance of the proposed framework. To that end, we consider a bridge design example whose dimensions and boundary conditions are shown in Fig. 24. The presence of a bridge deck is incorporated in the topology optimization through a non-designable solid layer, which has a thickness of $4/3$ and is composed of solid finite elements whose densities are kept constant throughout the optimization process. The top of the non-designable layer, which is located at $z = H/2$ (where H is the height of the design domain), is subjected to a distributed load in the negative z direction of magnitude $\tau = 0.1$. In addition to the non-designable solid layer, we also introduce a void region below the bridge deck, whose design variables are kept as $z_{\min}$ throughout the optimization process. In terms of design parameters, the allowable volume fraction (excluding the bridge deck) is assumed to be $\overline{V} = 7\%$ and radius of the density filter is taken to be $R = 3$.

Making use of the symmetries in both x and y directions, we employ only a quarter of the design domain. To discretize the domain using the two-scale discretization methodology, we consider a fine-scale mesh consisting of 864,000 elements and 903,991 nodes and take the block size to be either $N_B = 5$ or $N_B = 10$. The detailed statistics

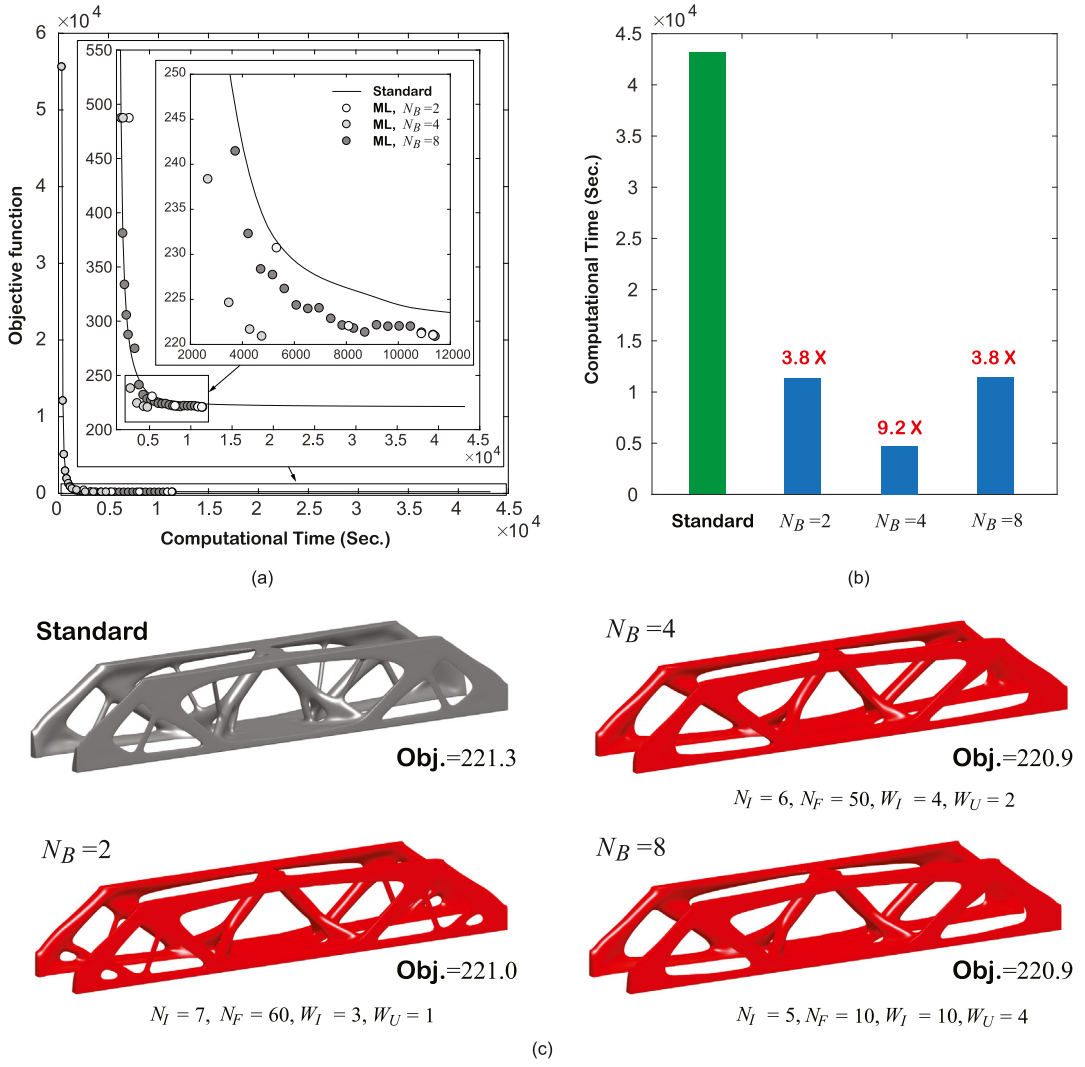

Fig. 23. (a) Convergence history of the objective function versus computational time for the proposed framework with various choices of N_B as well as the standard topology optimization approach. (b) Comparison of the total computational time spent by the standard topology optimization and the proposed framework with various choices of N_B (c) Final designs obtained from the standard topology optimization and the proposed framework with varying N_B and their corresponding parameter setup.

Table 9
Statistics of the fine-scale and coarse-scale meshes considered in the bridge design problem.

	# of elements	# of nodes
Fine-scale	864,000	903,991
Coarse-scale ($N_B = 5$)	6,912	8,575
Coarse-scale ($N_B = 10$)	864	1,300

of the two-scale discretizations is summarized in Table 9. For each block size value, we consider three cases with different batch size and maximum training parameters. Thus, a total of six cases are considered in this example, whose parameter and hyper-parameter setups are summarized in Table 10. For all the six cases, the initial learning rate ℓ_r for training/online update is 0.0005; P_{drop} is taken to be 0.9; and N_I, W_I, W_U, and N_F are taken to be 10,

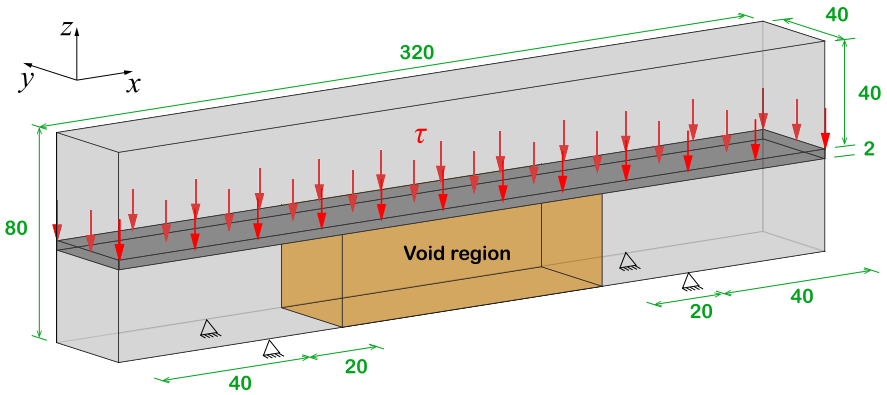

Fig. 24. The dimensions, load and boundary conditions of the bridge design domain. A non-designable solid region is placed in the middle height, representing the bridge deck; and a non-designable region (the yellow one) is assigned below the bridge deck.

Table 10
The six cases considered in the bridge example with their parameter & hyper-parameter setups.

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
N_B	5	5	5	10	10	10
Batch size	3,000	3,000	Full	300	300	Full
Max. training iterations	2,000	4,000	2,000	2,000	4,000	2,000
Weight Decay	10^{-5}	10^{-5}	10^{-5}	5×10^{-5}	5×10^{-5}	5×10^{-5}

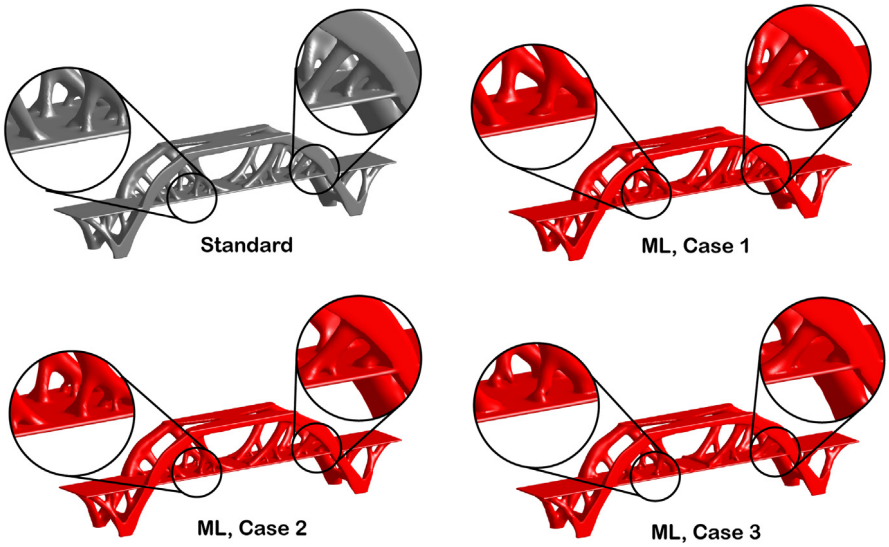

Fig. 25. Comparison of the final bridge designs obtained from the standard and machine learning-based approaches for cases where the block size $N_B = 5$.

5, 4, and 10, respectively. For cases with maximum training iteration being 2,000, the learning rate decays by a factor of 2 every 500 steps; and for those with maximum training iteration being 4,000, the learning rate decays by a factor of 2 every 800 steps.

In Figs. 25 and 26, we depict the final designs obtained from the machine learning-based approach for $N_B = 5$ and $N_B = 10$, respectively. The final objective values and the associated total computational costs for each of cases are quantified in Table 11. For comparison purposes, we also perform the bridge design using the standard topology optimization with the same design parameters and provide the results as well. Three observations are made. First,

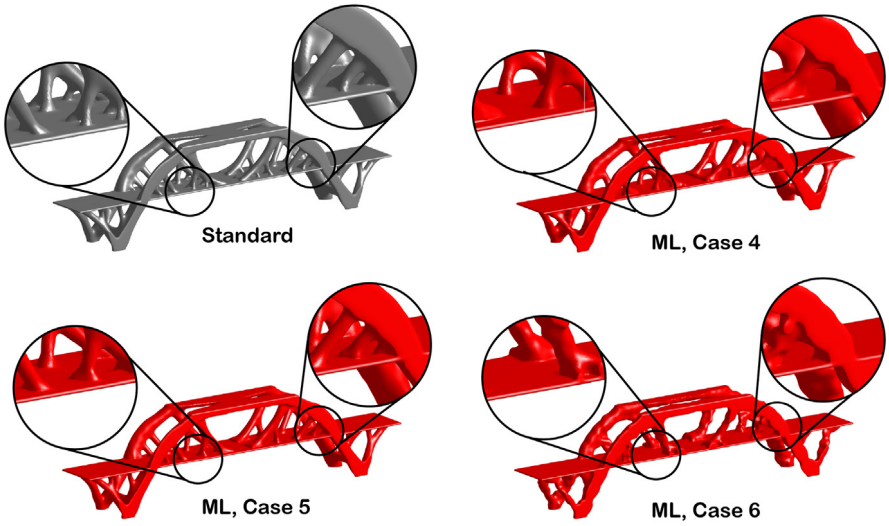

Fig. 26. Comparison of the final bridge designs obtained from the standard and machine learning-based approaches for cases where the block size $N_B = 10$.

Table 11
Summary of comparison between the results obtained by standard topology optimization and machine-learning for the bridge design problem.

	Standard	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Final Obj.	410.29	412.05	410.95	416.28	423.33	411.35	483.12
Diff.	N/A	0.43%	0.16%	1.46%	3.18%	0.25%	17.75%
Total time (s)	19,034	6,660	9,156	6,037	5,440	6,835	4,484

with the presence of the non-designable regions, the performance of the machine learning-based framework becomes more sensitive to the choice of block sizes than in the other examples. From Table 11, it is apparent that cases with $N_B = 5$ typically produce better designs than cases with $N_B = 10$. For example, in terms of final objective values, the maximum difference between the machine learning-based and standard approaches can be as high as 18% for $N_B = 10$, whereas the one for $N_B = 5$ is only 1.46%. In addition, as shown from the comparison in Figs. 25 and 26, the differences between the designs are mainly located at those regions above and below the bridge deck, and the presence of the bridge deck (which has a small thickness) is better captured by the coarse-scale mesh with $N_B = 5$ than $N_B = 10$. Second, it is observed that, for both $N_B = 5$ and $N_B = 10$, choosing a smaller batch size consistently gives better designs (in terms of objective function) than using a full batch size (i.e., equal to the sample size). This is because that, as compared to full batch, choosing a small batch can add random noise to the update of weights of the DNN to help them escape from bad local minima and saddle points [55]. However, one has to keep in mind the trade-off between efficiency and accuracy when choosing the batch size. As indicated in Table 12, given the same number of training iterations, choosing a smaller batch size leads to a longer training time, resulting in a larger total computational time with the proposed framework. For example, case 1 (with a batch size of 3,000) takes around 600 seconds more (with a full batch) to produce the final design than case 3. Finally, we also find that increasing the training iteration of the DNN is typically helpful to improve its prediction accuracy, especially when the block size N_B is big. For the large block size $N_B = 10$, it appears to be necessary in the proposed framework to train the DNN with 4,000 iterations instead of 2,000 to achieve a good performance (i.e., under 0.5% difference in objective value).

7. Concluding remarks

In this work, we propose a general machine learning-based topology optimization framework that can greatly accelerate the process of large-scale design problems without sacrificing accuracy. Unlike the existing approaches, in which the machine learning models are trained offline before the topology optimization, this work proposes a unique

Table 12
Detailed information about computational time spent in the training of DNN for each run case considered in the bridge design example.

Time (s)	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Ini. train time	207	396	152	111	215	34
Ave. time per update train	124	258	91	68	144	26
Total time	2,435	5,036	1,789	1,343	2,801	493

online training concept, which allows the machine learning model to be trained during the topology optimization process. The proposed framework uses history data of topology optimization as the training samples and, based on them, trains a machine learning model to directly predict the sensitivities without solving the state equations. Moreover, to promote better integration between machine learning and topology optimization, several new schemes are introduced. First, we present a two-scale topology optimization formulation which employs both coarse-scale and fine-scale meshes. The setup enables the online training to be performed based on local data, which is shown to significantly improve both the accuracy and scalability of the framework. Second, we propose an online updating scheme which continuously improves the machine learning model during the optimization process by providing new training data generated from physical simulations. Finally, a void sample drop out scheme is devised to improve the training efficiency and prediction accuracy of the machine learning model. Through design examples, the proposed framework is shown to significantly accelerate topology optimization with various load and boundary conditions, design requirements (e.g., volume fraction, filter radius, and non-designable regions) without sacrifice in accuracy. For example, the proposed machine learning-based framework is able to produce a design with the identical objective and achieve almost an order of magnitude speedup as compared to the standard topology optimization approach in the 3D MBB beam design example. In addition, we highlight that the proposed machine learning-based framework can offer an ideal tool for accelerating large-scale topology optimization problems as it is shown to achieve more speedup for problems with larger mesh sizes.

Through the design examples, some insights on how to properly choose the parameters & hyper-parameters in the proposed framework are gained, which are summarized as follows:

- When the block size N_B is small, the performance of the proposed framework is insensitive to the parameters N_I, N_F, W_I and W_U. When the block size N_B is large, the performance of the proposed framework is sensitive to parameters N_I, N_F, W_I and W_U. In addition, for large block sizes, the convergence of the objective function may experience oscillation and the optimization is more likely to converge to different local minima from the one obtained by the standard topology optimization.
- In terms of computational efficiency, there exists a trade-off between small and large block sizes. On the one hand, although small block sizes (e.g., $N_B = 2$) allow us to choose a larger online update frequency N_F, which leads to less fine-scale system solves, each coarse-scale system solve and online training take more time. On the other hand, while larger block sizes (e.g., $N_B = 10$) require less time in solving the coarse-scale system, it needs more frequent online updates which increases the effort in solving fine-scale systems and online training. We find an intermediate block size (e.g., 4 or 5) balances the efforts on both sides and can lead to the best acceleration performance of the proposed framework.
- The practice of randomly dropping out void samples with a large P_{drop} value (e.g., $P_{\text{drop}} = 0.9$) can improve both the training efficiency and prediction accuracy of the machine learning model.
- With the presence of non-designable (especially solid) regions, it is beneficial to choose a relatively small block size and to use more epochs in online training so that the presence of the non-designable layer be captured well in the coarse-scale mesh and the machine learning model.
- In both online training and update, choosing a batch size which is smaller than the size of the training sample can lead to better predication accuracy in the proposed framework.

Finally, we highlight several future research directions of the proposed framework. First, we remark that the proposed machine learning-based framework is universal in the sense that it can work with any suitable machine learning model. Therefore, future extension of the present work includes incorporating other machine learning and deep learning models, such as the Polynomial Chaos (PC) [56], Kriging [42], Low Rank Approximations (LRA) [57,58], Support Vector Machine (SVM) [42,43], CNNs [44,45]) and ResNets [46], with the proposed

framework, and exploring the comparative advantages and disadvantages of each model. Second, although being applied to deterministic topology optimization problems, we believe that the proposed framework also has the potential in topology optimization under uncertainty, such as robust topology optimization. Unlike the present work, where the training data are deterministic, the training data from optimization under uncertainty will be stochastic, which requires more investigation on the proper machine learning model to correctly capture the uncertainty in the training data.

Acknowledgments

The authors acknowledge the financial support from Siemens Corporate Technology under the project titled "Deep Learning Enhanced Topology Optimization". The inception of this research was the Siemens FutureMakers Challenge at Georgia Tech (May 4-5, 2018), which included an institute-wide hackathon. Two authors of this paper were members of the hackathon winning team: HC, YZ & GHP. The other members of the hackathon team were Mrs. Emily D. Sanders and Mr. Yang Jiang. GHP was the hackathon faculty advisor for the team. HC and GHP acknowledge the support from the Raymond Allen Jones Chair at the Georgia Institute of Technology.

References

[1] Martin Philip Bendsøe, Noboru Kikuchi, Generating optimal topologies in structural design using a homogenization method, Comput. Methods Appl. Mech. Engrg. 71 (2) (1988) 197–224.
[2] George I.N. Rozvany, A critical review of established methods of structural topology optimization, Struct. Multidiscip. Optim. 37 (3) (2009) 217–237.
[3] Peter Christensen, Anders Klarbring, An Introduction to Structural Optimization, Springer Science & Business Media, Linköping, 2009.
[4] Martin Philip Bendsøe, Ole Sigmund, Topology Optimization: Theory, Methods, and Applications, Springer Science & Business Media, 2013.
[5] Raphael T. Haftka, Zafer Gürdal, Elements of structural optimization, vol. 11, Springer Science & Business Media, 2012.
[6] Thomas Borrvall, Joakim Petersson, Large-scale topology optimization in 3D using parallel computing, Comput. Methods Appl. Mech. Engrg. 190 (46–47) (2001) 6201–6229.
[7] Niels Aage, Erik Andreassen, Boyan Stefanov Lazarov, Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework, Struct. Multidiscip. Optim. 51 (3) (2015) 565–572.
[8] Niels Aage, Erik Andreassen, Boyan S. Lazarov, Ole Sigmund, Giga-voxel computational morphogenesis for structural design, Nature 550 (7674) (2017) 84.
[9] Shun Wang, Eric de Sturler, Glaucio H. Paulino, Large-scale topology optimization using preconditioned Krylov subspace methods with recycling, Internat. J. Numer. Methods Engrg. 69 (12) (2007) 2441–2468.
[10] Oded Amir, Niels Aage, Boyan S. Lazarov, On multigrid-CG for efficient topology optimization, Struct. Multidiscip. Optim. 49 (5) (2014) 815–829.
[11] Yoon Young Kim, Gil Ho Yoon, Multi-resolution multi-scale topology optimization—a new paradigm, Int. J. Solids Struct. 37 (39) (2000) 5529–5559.
[12] Tam H. Nguyen, Glaucio H. Paulino, Junho Song, Chau H. Le, A computational paradigm for multiresolution topology optimization (MTOP), Struct. Multidiscip. Optim. 41 (4) (2010) 525–539.
[13] Hui Liu, Yiqiang Wang, Hongming Zong, Michael Yu Wang, Efficient structure topology optimization by using the multiscale finite element method, Struct. Multidiscip. Optim. (2018) 1–20.
[14] Christian Robert, Machine Learning, a Probabilistic Perspective, Taylor & Francis, 2014.
[15] Trevor Hastie, Robert Tibshirani, Jerome Friedman, James Franklin, The elements of statistical learning: data mining, inference and prediction, Math. Intell. 27 (2) (2005) 83–85.
[16] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, et al., Deep face recognition, in: BMVC, vol. 1, 2015, p. 6.
[17] K.K. Kim, K.I. Kim, J.B. Kim, H.J. Kim, Learning-based approach for license plate recognition, in: Neural Networks for Signal Processing X, Proceedings of the 2000 IEEE Signal Processing Society Workshop (Cat. No. 00TH8501), vol. 2, IEEE, 2000, pp. 614–623.
[18] Yuyu Zhang, Hanjun Dai, Chang Xu, Jun Feng, Taifeng Wang, Jiang Bian, Bin Wang, Tie-Yan Liu, Sequential click prediction for sponsored search with recurrent neural networks, in: Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014.
[19] Tie-Yan Liu, Learning to rank for information retrieval, Found. Trends® Inf. Retrieval 3 (3) (2009) 225–331.
[20] Shuai Zhang, Lina Yao, Aixin Sun, Yi Tay, Deep learning based recommender system: A survey and new perspectives, ACM Comput. Surv. (ISSN: 0360-0300) 52 (1) (2019) 5:1–5:38, http://dx.doi.org/10.1145/3285029, URL http://doi.acm.org/10.1145/3285029.
[21] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Deep learning, Nature 521 (7553) (2015) 436.
[22] Jürgen Schmidhuber, Deep learning in neural networks: An overview, Neural Netw. 61 (2015) 85–117.
[23] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT press, 2016.
[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.

[25] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, Demis Hassabis, Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484.
[26] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, Demis Hassabis, Mastering the game of go without human knowledge, Nature 550 (7676) (2017) 354–359.
[27] Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mu Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, Ming Zhou, Achieving human parity on automatic Chinese to English news translation, 2018, arXiv preprint arXiv:1803.05567.
[28] Erva Ulu, Rusheng Zhang, Levent Burak Kara, A data-driven investigation and estimation of optimal topologies under variable loading configurations, Comput. Methods Biomech. Biomed. Eng.: Imaging Vis. 4 (2) (2016) 61–72.
[29] Ivan Sosnovik, Ivan Oseledets, Neural networks for topology optimization, 2017, arXiv preprint arXiv:1709.09578.
[30] Saurabh Banga, Harsh Gehani, Sanket Bhilare, Sagar Patel, Levent Kara, 3D topology optimization using convolutional neural networks, 2018, arXiv preprint arXiv:1808.07440.
[31] Yonggyun Yu, Taeil Hur, Jaeho Jung, Deep learning for determining a near-optimal topological design without any iteration, Struct. Multidiscip. Optim. (2019) 1–13.
[32] Xin Lei, Chang Liu, Zongliang Du, Weisheng Zhang, Xu Guo, Machine learning-driven real-time topology optimization under moving morphable component-based framework, J. Appl. Mech. 86 (1) (2019) 011004.
[33] Cameron Talischi, Glaucio H. Paulino, Anderson Pereira, Ivan F.M. Menezes, PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes, Struct. Multidiscip. Optim. 45 (3) (2012) 329–357.
[34] Martin P. Bendsøe, Optimal shape design as a material distribution problem, Struct. Optim. 1 (4) (1989) 193–202.
[35] George I.N. Rozvany, Ming Zhou, Torben Birker, Generalized shape optimization without homogenization, Struct. Optim. 4 (3–4) (1992) 250–252.
[36] Martin P. Bendsøe, Ole Sigmund, Material interpolation schemes in topology optimization, Arch. Appl. Mech. 69 (9–10) (1999) 635–654.
[37] Thomas J.R. Hughes, The Finite Element Method: Linear Static and Dynamic Finite Element Analysis, Courier Corporation, 2012.
[38] Krister Svanberg, The method of moving asymptotes—a new method for structural optimization, Internat. J. Numer. Methods Engrg. 24 (2) (1987) 359–373.
[39] Andrew R. Barron, Approximation and estimation bounds for artificial neural networks, Mach. Learn. 14 (1) (1994) 115–133.
[40] Kurt Hornik, Maxwell Stinchcombe, Halbert White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[41] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, Liwei Wang, The expressive power of neural networks: A view from the width, in: Advances in Neural Information Processing Systems, 2017, pp. 6231–6239.
[42] Maliki Moustapha, Bruno Sudret, Surrogate-assisted reliability-based design optimization: a survey and a unified modular framework, Struct. Multidiscip. Optim., 1–20.
[43] Bernhard Scholkopf, Alexander J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, 2001.
[44] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, Lawrence D Jackel, Backpropagation applied to handwritten zip code recognition, Neural Comput. 1 (4) (1989) 541–551.
[45] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[47] Karl Pearson, LIII. On lines and planes of closest fit to systems of points in space, Lond. Edinb. Dublin Phil. Mag. J. Sci. 2 (11) (1901) 559–572.
[48] Harold Hotelling, Analysis of a complex of statistical variables into principal components., J. Educ. Psychol. 24 (6) (1933) 417.
[49] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[50] Tam H. Nguyen, Glaucio H. Paulino, Junho Song, Chau H. Le, Improving multiresolution topology optimization via multiple discretizations, Internat. J. Numer. Methods Engrg. 92 (6) (2012) 507–530.
[51] Adeildo S. Ramos, Glaucio H. Paulino, Filtering structures out of ground structures–a discrete filtering tool for structural design optimization, Struct. Multidiscip. Optim. 54 (1) (2016) 95–116.
[52] Xiaojia Zhang, Adeildo S. Ramos, Glaucio H. Paulino, Material nonlinear topology optimization using the ground structure method with a discrete filtering scheme, Struct. Multidiscip. Optim. 55 (6) (2017) 2045–2072.
[53] A.N. Tikhonov, A.V. Goncharsky, V.V. Stepanov, A.G. Yagola, Numerical Methods for the Solution of Ill-Posed Problems, vol. 328, Springer Science & Business Media, 2013.
[54] Niels Aage, Boyan S. Lazarov, Parallel framework for topology optimization using the method of moving asymptotes, Struct. Multidiscip. Optim. 47 (4) (2013) 493–505.
[55] Rong Ge, Furong Huang, Chi Jin, Yang Yuan, Escaping from saddle points—online stochastic gradient for tensor decomposition, in: Conference on Learning Theory, 2015, pp. 797–842.
[56] Emiliano Torre, Stefano Marelli, Paul Embrechts, Bruno Sudret, Data-driven polynomial chaos expansion for machine learning regression, J. Comput. Phys. 388 (2019) 601–623.

[57] Anthony Nouy, Proper generalized decompositions and separated representations for the numerical solution of high dimensional stochastic problems, Arch. Comput. Methods Eng. 17 (4) (2010) 403–434.
[58] Ivan Markovsky, Konstantin Usevich, Low Rank Approximation, Springer, 2012.