Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 398 (2022) 115307
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

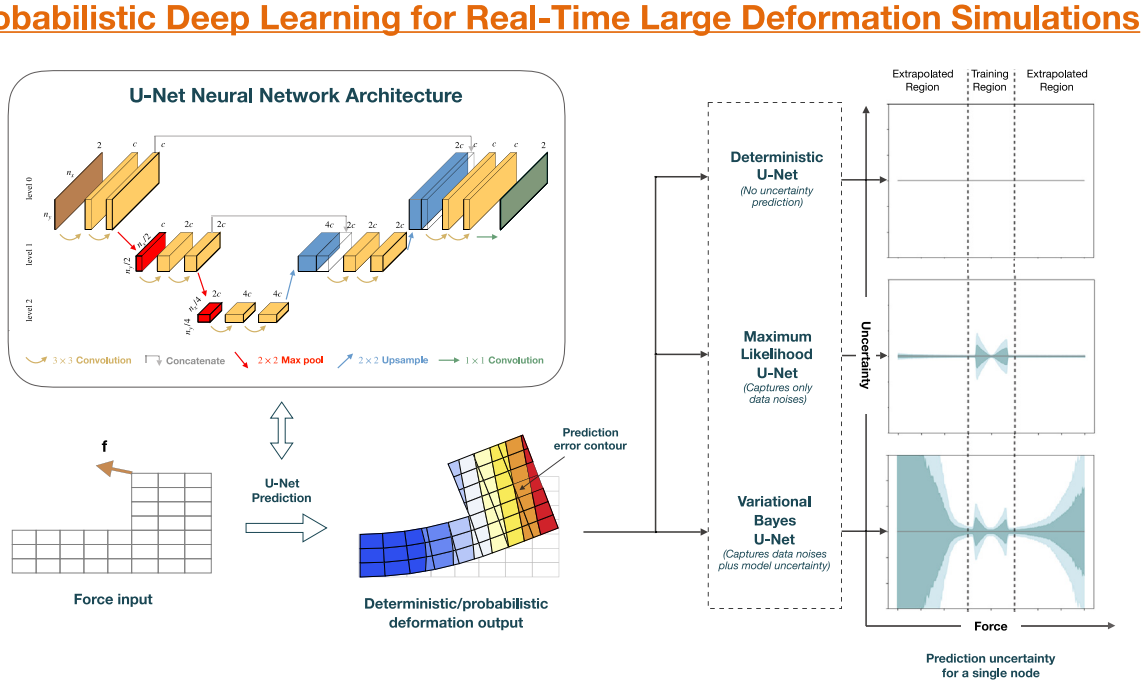
Probabilistic deep learning for real-time large deformation simulations

Saurabh Deshpande[a], Jakub Lengiewicz[a,b], Stéphane P.A. Bordas[a,*]

[a] *Department of Engineering; Faculty of Science, Technology and Medicine; University of Luxembourg, Luxembourg*
[b] *Institute of Fundamental Technological Research, Polish Academy of Sciences, Poland*

Received 27 December 2021; received in revised form 26 June 2022; accepted 27 June 2022
Available online xxxx

Graphical Abstract



Abstract

For many novel applications, such as patient-specific computer-aided surgery, conventional solution techniques of the underlying nonlinear problems are usually computationally too expensive and are lacking information about how certain can

* Corresponding author.
E-mail address: stephane.bordas@alum.northwestern.edu (S.P.A. Bordas).

https://doi.org/10.1016/j.cma.2022.115307

we be about their predictions. In the present work, we propose a highly efficient deep-learning surrogate framework that is able to accurately predict the response of bodies undergoing large deformations in real-time. The surrogate model has a convolutional neural network architecture, called U-Net, which is trained with force–displacement data obtained with the finite element method. We propose deterministic and probabilistic versions of the framework. The probabilistic framework utilizes the Variational Bayes Inference approach and is able to capture all the uncertainties present in the data as well as in the deep-learning model. Based on several benchmark examples, we show the predictive capabilities of the framework and discuss its possible limitations.

Keywords: Convolutional neural network; Bayesian inference; Bayesian deep learning; Large deformations; Finite element method; Real-time simulations

1. Introduction

Reliable and computationally efficient models are crucial in the design, optimization, or control for various application domains, including aerospace engineering, robotics, or bio-medicine. For instance the increasing interest in biomedical simulations [1–5] may require having real-time responses. Finding convenient trade-offs between the accuracy and response time of such computational models is currently an active area of research in the context of digital twins, and is also one of the motivations for the research presented in this work.

When accuracy is important, the most general and widely used methodology in engineering for solving boundary value problems is the finite element method (FEM) [6]. This accuracy, especially when it comes to highly non-linear or history-dependent problems, may require a significant computational effort. The advancements in hardware development and software optimization enabled to some extent speeding up FEM computations, which involves specialized solution strategies to take advantage of high-performance computing architectures. A notable example is the class of Finite Element Tearing and Interconnecting (FETI) methods [7]. In these methods, the global domain is partitioned into a set of disconnected sub-domains, which are computed in parallel on different processors/nodes. However, in many applications, it is not possible to meet real-time responses on the hardware available for industrial consumers. This is due to a limited number of available cores and a significant communication burden that deteriorates the overall time performance of such solution strategies.

There are various specialized FEM-based approaches to cut down the solution time at the cost of sacrificing the accuracy, see, e.g., [8]. An important class of such approaches is the model order reduction (MOR) methods, with Proper Orthogonal Decomposition (POD) being one of the notable examples. The general concept of POD, e.g., applied to a discretized FEM formulation, is to find a low dimension subspace in order to approximate the full space at an acceptable loss of accuracy. This potentially enables controlling the trade-off between accuracy and computation time. POD was adapted to work within the large-deformation regime, see, e.g., [9], which was for instance applied to simulate and control soft robotic arms [10] or to reduce computational costs in nonlinear fracture mechanics problems [11]. However, the efficiency of POD deteriorates in high non-linear regimes since it relies on a linear combination of few basis vectors and thus oversimplifies the model [12]. Proper Generalized Decomposition (PGD) is another MOR technique, in which the solution of the complete problem is computed as a finite sum of separable functions. The compact solution, though not optimal, in general, provides a very light format to store the solution in the form of a meta-model, thereby speeding up the solution times. Niroomandi et al. [13] implemented PGD based approach for computationally efficient simulations of hyper-elastic responses. However, the accuracy of PGD methods decreases when the separation of variables assumption cannot respect the problem to solve [14].

Importantly for the present work, we distinguish yet another family of FEM-based approaches, in which the expected speedups and approximation capabilities originate from underlying Deep Neural Networks (DNNs) with Deep Learning (DL) techniques used to train these networks. Generally, DL approaches make an important part of machine learning techniques and have allowed to solving highly complex problems that had eluded scientists for decades. In particular, DL-based methods have also been developed to efficiently solve problems in engineering [15]. One of the popular approaches utilizes the idea of the so-called Physics Informed Neural Networks (PINNs), in which both the data (either synthetic or experimental) and the assumed governing Partial Differential Equations (PDEs) are incorporated in the training phase; for early traces of these see, e.g., [16–18], and for a recent study

see, e.g., [19,20]. One of the benefits of this approach is that possibly much less data is needed for training, which can be an important factor for many data-driven applications. Note, however, that even if the physics is not explicitly enforced in the training phase (non-PINN case), it can still be recovered in the trained model by implicitly following a large amount of training data (synthetic or experimental). In the case when all training data are synthetically provided from FEM simulations, see, e.g., [21–23], we will refer to it as the *direct FEM-based approach*.

In this work, we propose a framework that falls into the above-mentioned class of direct FEM-based DNN approaches. The framework is based on a particular DNN architecture – the U-Net architecture [24] – which in turn can be viewed as a type of Convolutional Neural Networks (CNNs); further explanations are provided later in this work. Originally, the U-Net architecture has been developed for the purpose of biomedical image segmentation, however, it turned out to be also suitable for other applications. In particular, the present work is inspired by the recent results of [22], in which the authors demonstrate quite accurate real-time non-linear force–displacement predictions done by U-Nets trained on FEM-based data. It has been noticed, see [25,26], that this good accuracy is not accidental but can be possibly linked to the strong resemblance of U-Net architectures and multi grid solution schemes [27]. Such a point of view makes the U-Net approach less of a brute-force black-box approximation and more of a suited solution scheme, which makes this line of research very promising.

An equally important aspect that is studied in the present work is the capabilities of DNNs to quantify uncertainties. This is motivated by the fact that in many real-life applications, such as surgical simulations [5] or autonomous driving [28], it is crucial to produce reliable uncertainty estimates in addition to the predictions. Otherwise, model predictions can lead to harmful consequences in these critical tasks. Deterministic neural networks are usually certain about their predictions, and this overconfidence is especially evident when facing data far from the training set. Generally, uncertainties can be categorized as those associated with the misfit of neural network models (epistemic uncertainties) and those that refer to the noise in training data (aleatoric uncertainties) [29,30]. Uncertainties can either fall within or outside the training data region (interpolated and extrapolated regions, respectively). Within the data region, the prediction uncertainties are caused both by noisy data and by the model misfit, which can be captured and quantified in many ways, the Maximum Likelihood Estimation (MLE) method is one of the most straightforward [30] to do so. However, for the extrapolated region we have no data support, therefore, no direct quantification can be done there. What we can only reasonably assume is that the uncertainty should generally increase when moving away from the data region. To achieve this, in this work, we will extend the idea proposed in [31–33] which relies on converting a neural network to its stochastic counterpart by replacing discrete parameters with probability distributions and using a special training technique that is based on Bayesian Inference.

Bayesian approaches have been already considered in the context of FEM models; for instance in [34] uncertainties are quantified for hyperelastic soft tissues by incorporating stochastic parameters in the FEM model, while in [35,36] a thorough tutorial on using Bayesian Inference to solid mechanics problems is provided. In the present work, however, we focus on Bayesian Inference and related Bayesian Neural Networks (BNNs). Here, in the context of deep networks with millions of parameters, the application of widely used Markov Chain Monte Carlo (MCMC) type of methods would be computationally intractable. For that reason, in this work we use one of the well-known methods for approximate Bayesian inference – Variational Inference (VI) [37] – in which an approximate distribution is used instead of the true Bayesian posterior over model parameters.

To sum up, the scope of the present work is to study the applicability of U-Net deep learning architectures to performing real-time predictions for large-deformation problems with uncertainties, where synthetic training data is provided by FEM simulations. The organization of the paper is the following. In Section 2 we present the general methodology applied to a deterministic version of U-Net. In Section 3 we introduce the extension of the framework to the Variational Bayesian Inference case. Then, in Section 4, an extensive study of the proposed framework is performed, which is based on several 2D and 3D benchmark examples. The conclusions and future research directions are outlined in Section 5.

2. General FEM-based U-Net methodology

The proposed approach can be divided into two main phases. The first phase involves finite element simulations to prepare necessary datasets. The second phase consists of building and training deterministic/probabilistic U-Net deep neural networks, using the training datasets generated in the first phase. The trained U-Nets are then used as surrogate models.

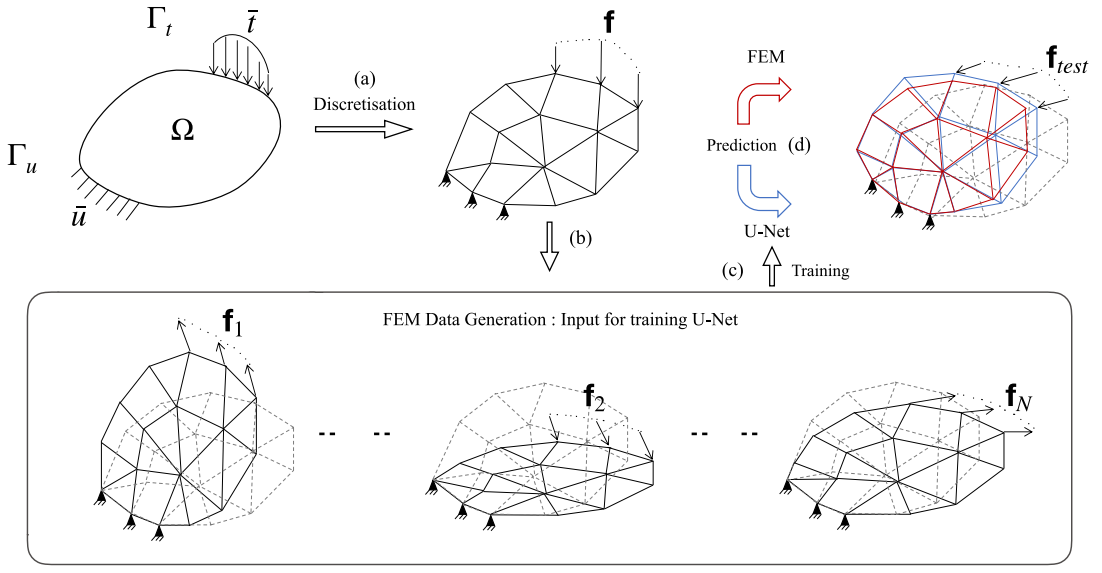


Fig. 1. Schematic of the framework (a) Continuum problem is discretized by FEM mesh (b) Training/testing examples are generated by applying random point forces on Neumann boundary. (c) The U-Net is trained on the generated dataset (d) Trained U-Net predicts the deformation for a test force (blue mesh). FEM solution (red) is used for cross-validation. Gray dashed meshes indicate undeformed configurations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.1. FEM-based deep learning approach

As a problem to be solved, we consider the boundary value problem of a hyperelastic solid, with a constant prescribed Dirichlet BC. Large deformations exhibit non-linear stress–strain behavior when applied with external forces, hence one needs to consider hyper-elastic constitutive laws for simulating such systems.

Consider a boundary value problem in continuum mechanics in the domain Ω, Dirichlet and Neumann boundary conditions are applied on Γ_D, Γ_N respectively. Neglecting the body forces, the virtual work principle for nonlinear elastostatic equation reads

$$\int_{\Omega} \boldsymbol{P}(\boldsymbol{u}) \cdot \nabla \delta \boldsymbol{u} \, dV - \int_{\Gamma_t} \bar{\boldsymbol{t}} \cdot \delta \boldsymbol{u} \, dS = 0 \qquad \forall \delta \boldsymbol{u}, \tag{1}$$

where $\boldsymbol{u}$ and $\delta \boldsymbol{u}$ belong to appropriate functional spaces, $\boldsymbol{u} = \bar{\boldsymbol{u}}$ and $\delta \boldsymbol{u} = \boldsymbol{0}$ on Γ_u, and $\boldsymbol{P}(\boldsymbol{u})$ is the first Piola–Kirchhoff stress tensor. The required constitutive relationship will be defined through the (hyper-)elastic strain energy potential $W(\boldsymbol{F})$ as

$$\boldsymbol{P}(\boldsymbol{F}) = \frac{\partial W(\boldsymbol{F})}{\partial \boldsymbol{F}} \tag{2}$$

where $\boldsymbol{F} = \boldsymbol{I} + \nabla \boldsymbol{u}$ is the deformation gradient tensor. (The particular form of W, used in this work, will be presented in Section 4.1.1.)

After standard FE discretization, the problem expressed by Eq. (1) will take the form of system of non-linear equations

$$\mathbf{R}(\mathbf{u}) = \mathbf{f}_{\text{int}}(\mathbf{u}) - \mathbf{f}_{\text{ext}} = \mathbf{0}, \tag{3}$$

which expresses the balance between external and internal nodal forces. By solving the system of equations (3) (e.g., with the Newton–Raphson method) for a given external force vector $\mathbf{f}_{\text{ext}} = \mathbf{f}$ we obtain a solution in the form of nodal displacements $\mathbf{u}$.

External forces can be applied to a selected region on the surface described by Γ_t. For the current framework, we consider a single FE discretization of a given domain Ω. As described in Fig. 1, we apply a prescribed family of load distributions, given by vectors of force $\mathbf{f}_i$ on the nodes present in Γ_t to generate nodal displacements $\mathbf{u}_i$.

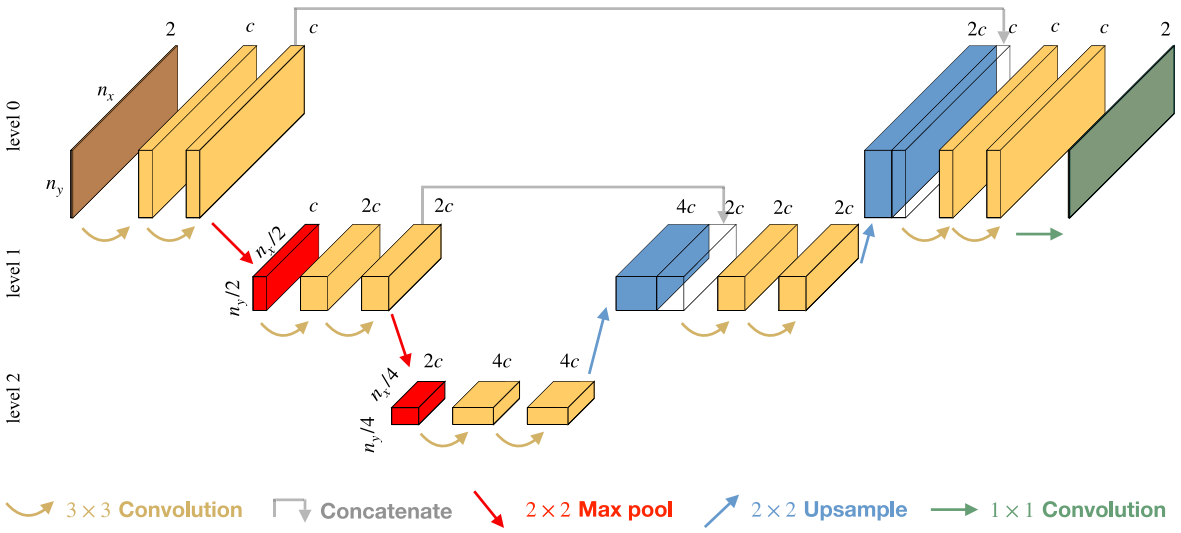


Fig. 2. A schematic of exemplary U-Net architecture for 2D domains, (n_x, n_y) stand for number of nodes in x, y direction of 2D domain. Boxes indicate U-Net layers (colors indicate different types of layers), first step contains 'c' channels.

This creates a dataset $\mathcal{D} = (\mathbf{f}_i, \mathbf{u}_i)_{i=1}^{N}$ of corresponding pairs of nodal force and displacement vectors, which is then used as input to train DNNs. Thus the input of the neural network is a vector of nodal forces, and the predicted output is a vector of nodal displacements.

2.2. U-Net deep neural network architecture

As motivated in the introduction, we use a specific family of CNN, U-Nets [24]. They owe their name to a specific U-shape of the architecture diagram, e.g., see Fig. 2, which is an effect of applying cascades of max pooling operations, followed by cascades of upsampling operations.

At its input, the layer $\boldsymbol{d}^0$, the U-Net network, $\mathcal{U}$, accepts the vector of external forces, $\mathbf{f}$, in the original mesh format $n_x \times n_y \times n_z \times 3$, where n_x, n_y, n_z are the dimensions of the structured 3D mesh (for 2D problems the format is $n_x \times n_y \times 2$). The output displacements, in the layer $\boldsymbol{d}^L$, are in the analogous mesh format. The model parameters $\boldsymbol{\theta}_{\text{det}}$ are all weights (kernel $\boldsymbol{k}$ and biases $\boldsymbol{b}$) of the neural network (see below for details).

For the sake of clarity, below we will introduce the idea for a 2D case only, which can be straightforwardly transformed into a 3D case. For 2D problems, we use architecture as described in Fig. 2. To a given input mesh, we first add double zero padding in each spatial direction (this is done to avoid the loss of information of corner nodes). There are two layers in each encoding and decoding phase. To the padded input, we apply two convolutions with batch normalization followed by rectified linear unit activation (ReLU), see Eq. (4). In the encoding phase, at each step, 2×2 Max Pooling layers are applied which decrease spatial dimensions by half, and we multiply the number of channels by 2 (c=128 for the first level). In decoding steps, 2×2 Up-sampling layers are applied which increases the spatial dimension by two and the number of channels is halved. At each level, encoding and decoding outputs are concatenated together. In the end, 1×1 convolution is applied with linear activation to get the output.

In order to better understand the idea of convolution operator, the $3 \times 3 \times c$ operator in 2D on the first U-Net level can be imagined as a filter window that is applied to a $n_x \times n_y \times c$ mesh. For a two-dimensional domain, the input tensor is of the dimension $n_x \times n_y \times 2$, which is identical to the FEM mesh. Here 2 stands for the number of channels of input convolutions, n_x, n_y stands for the number of nodes in the x, y direction, respectively. To best leverage the CNNs, we keep x&y-dofs in separate channels. We operate a convolutional filter on a local region and then is slid along spatial directions x, y with a stride of 1 as illustrated in Fig. 3.

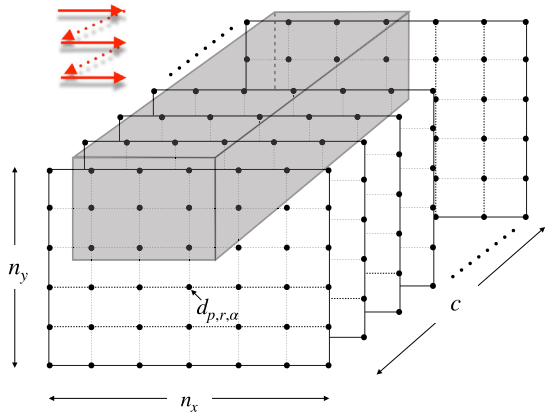


Fig. 3. x and y dofs are stored in different channels, 3×3 convolutional filter (gray) acts locally along the channel direction and it slides along with the step of 1 in both horizontal and vertical directions (red).

An example of a non-restrictive convolution operation (in 2D), between subsequent U-Net layers l and $l+1$, for the filter size $3 \times 3 \times c^n$ reads

$$d^{l+1}_{p,r,\beta} = \mathcal{A}\left(b^{l+1}_{\beta} + \sum_{i=1,}^{3} \sum_{j=1,}^{3} \sum_{\alpha=1}^{c^l} d^{l}_{p+i-2,r+j-2,\alpha} k^{l+1}_{i,j,\alpha,\beta} \right), \tag{4}$$

where $d^{l+1}_{p,r,\beta}$ are neural network nodes at layer $l+1$, the weights $k^{l+1}_{i,j,\alpha,\beta}$ are parameters of the convolution operator, the weights b^{l+1}_{β} are biases at a layer $l+1$, and $\mathcal{A}(\cdot)$ is an activation function (ReLU). Indices i, j stand for the components of the convolutional filter (3×3 in our case) and the indices p, r are related to nodes in a 2D grid of the output layers. They directly refer to the underlying structured FEM mesh. In our case, we add zero pad in each dimension of input before applying the convolution, to ensure the same size of the input and output. Indices $1 \leq \alpha \leq c^l$ and $1 \leq \beta \leq c^{l+1}$ represent the channel number. Note that the number of channels in subsequent layers need not be equal, i.e., in general $c^l \neq c^{l+1}$. Note also that in the first and in the last layer, the number of channels correspond to the spatial dimension of the problem (2D or 3D).

Max-pooling operation is responsible for reducing the spatial dimensions of its input, channel dimensions are unaffected by it. It reads as follows

$$d^{l+1}_{p,r,\alpha} = \max_{\substack{2p-1 \leqslant i \leqslant 2p \\ 2r-1 \leqslant j \leqslant 2r}} d^{l}_{i,j,\alpha} \tag{5}$$

Upsampling operator can be seen as the reverse of max pooling, it increases the spatial dimensions of the input without affecting the channel dimension. As showed in Fig. 2, in the decoder phase, outputs of up-sampling are concatenated with respective layers from the encoder phase of the U-Net (in case of symmetric U-Nets, the lth layer is concatenated with the $(L-l-2)$-th layer, where L is the index of output layer in a U-Net). The up-sampling with concatenation read

$$d^{l+1}_{p,r,\alpha} = \begin{cases} d^{l}_{\lfloor p/2 \rfloor + 1, \lfloor p/2 \rfloor + 1, \alpha} & , \quad 1 \leqslant \alpha \leqslant c^l \\ d^{L-l-2}_{p,r,(\alpha - c^l)} & \quad c^l + 1 \leqslant \alpha \leqslant c^l + c^{L-l-2} \end{cases} \tag{6}$$

The final 1×1 convolution operation reads

$$d^{L}_{p,r,\beta} = b^{L}_{\beta} + \sum_{\alpha=1}^{c^{L-1}} d^{L-1}_{p,r,\alpha} k^{L}_{\alpha,\beta}, \qquad \beta = 1, 2. \tag{7}$$

For the 3D version of U-Nets, the operations given by Eqs. (4)–(7) are straightforwardly extended by one additional dimension. This results in adding one index to nodes' and biases' specifications, d and b, respectively, and two indexes to 3×3 convolution weights, k. For 2D/3D cases, trainable parameters of the deterministic U-Net

are

$$\boldsymbol{\theta}_{\text{det}} = \bigcup_{l=1}^{L} \{\mathbf{k}^l, \mathbf{b}^l\}. \tag{8}$$

$\mathcal{U}(\mathbf{f}, \boldsymbol{\theta}^{\text{det}})$ is defined recursively (the forward propagation), starting from the input layer $\boldsymbol{d}^0 = \mathbf{f}$, then subsequently applying appropriate transformations given by one of Eqs. (4)–(6), and finally applying the transformation given by Eq. (7), see also Fig. 2. Finally the prediction of the deterministic U-Net is

$$\mathcal{U}(\mathbf{f}, \boldsymbol{\theta}_{\text{det}}) = \mathbf{d}^L \tag{9}$$

For a given training dataset $\mathcal{D} = \{(\mathbf{f}_1, \mathbf{u}_1), \ldots, (\mathbf{f}_N, \mathbf{u}_N)\}$, the deterministic U-Net is trained by minimizing the following mean squared error loss function

$$\mathcal{L}_{\text{det}}(\mathcal{D}, \boldsymbol{\theta}_{\text{det}}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathcal{U}(\mathbf{f}_i, \boldsymbol{\theta}_{\text{det}}) - \mathbf{u}_i\|_2^2 \tag{10}$$

which gives the optimal parameters

$$\boldsymbol{\theta}^*_{\text{det}} = \arg \min_{\boldsymbol{\theta}_{\text{det}}} \mathcal{L}_{\text{det}}(\mathcal{D}, \boldsymbol{\theta}_{\text{det}}). \tag{11}$$

A particular training strategy, used in this work, is introduced in Section 4.1.2.

Remark: The current framework (as well as all other neural-network based approaches mentioned in the introduction) requires retraining a network when changing the FE discretization. Recently proposed operator-based learning approaches, called neural operators [38–40], promise to overcome this disadvantage.

3. Probabilistic U-Net framework

There are various sources of uncertainties linked to engineering systems. These can be broadly categorized as noises in the observation (aleatoric uncertainty or data uncertainty) and uncertainty in the assumption of our model (epistemic uncertainty or model uncertainty) [30]. Aleatoric uncertainty is inherent to the data and it cannot be reduced, whereas epistemic uncertainty can be reduced by providing more training data. An important part of epistemic uncertainty is being able to tell that the more data we have, the more certain we are about the predictions. This uncertainty is expected to be high while doing predictions on inputs away from the training region. Deterministic U-Nets explained in Section 2.2 fail to account for these uncertainties. In order to capture these effects, in this work, we propose the Bayesian approach, which is introduced in this section.

3.1. Variational Bayesian inference

Bayesian methods provide an approach to quantify the uncertainty of prediction in deep neural networks. To do so, in this framework, we replace the deterministic parameters with probability distributions [31]. Originally this idea was only used to prevent overfitting, but was observed to also increase the variability of outputs in the extrapolated region. In order to suitably control the level of introduced perturbations to parameters, we use Bayesian Inference. This gives us a formal theoretical framework, allowing us to apply suitable computational techniques (VI) to efficiently train networks and predict results. The input of a network remains the same, but some of the model parameters become probability distributions (stochastic), and for that reason also the output of the network becomes a distribution over possible outputs. As per the standard Bayesian approach, we specify a prior distribution $P(\boldsymbol{w})$ over parameters, we consider $\mathcal{D} = \{(\mathbf{f}_1, \mathbf{u}_1), \ldots, (\mathbf{f}_N, \mathbf{u}_N)\}$ as the given training dataset, then for a new vector $\mathbf{f}^{\text{test}}$, prediction $\mathbf{u}^{\text{test}}$ is given by

$$P(\mathbf{u}^{\text{test}}|\mathbf{f}^{\text{test}}, \mathcal{D}) = \int P(\mathbf{u}^{\text{test}}|\mathbf{f}^{\text{test}}, \boldsymbol{w}) P(\boldsymbol{w}|\mathcal{D}) \mathrm{d}\boldsymbol{w} \tag{12}$$

The Bayesian inference involves the calculation of true parameter posterior $P(\boldsymbol{w}|\mathcal{D})$ conditioned over the training data. As mentioned in the introduction, variational inference (VI) is used to approximate true posterior densities in Bayesian neural networks [37], i.e. true posterior is approximated by a variational posterior $q(\boldsymbol{w}|\boldsymbol{\theta})$, parameterized by

$\boldsymbol{\theta}$. Variational learning finds optimal parameters $\boldsymbol{\theta}^*$ by minimizing the Kullback–Leibler divergence (KL-divergence) between true and variational posteriors, as per the following equation:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \mathrm{KL}\big[q(\boldsymbol{w}|\boldsymbol{\theta}) \parallel P(\boldsymbol{w}|\mathcal{D})\big] \\ &= \arg\min_{\boldsymbol{\theta}} \int q(\boldsymbol{w}|\boldsymbol{\theta}) \log \frac{q(\boldsymbol{w}|\boldsymbol{\theta})}{P(\boldsymbol{w})P(\mathcal{D}|\boldsymbol{w})} dw \\ &= \arg\min_{\boldsymbol{\theta}} \mathrm{KL}[q(\boldsymbol{w}|\boldsymbol{\theta}) \parallel P(\boldsymbol{w})] - \mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{\theta})}[\log P(\mathcal{D}|\boldsymbol{w})].\end{aligned} \tag{13}$$

The resulting cost function is the loss function for training the neural network. It consists of two parts, first is the prior dependent part represented by the KL-divergence term, it can be referred to as model complexity cost; it tells how close approximate posteriors are to priors. And later is the data-dependent part which can be referred to as likelihood cost, it tells how well the network fits the data. Bayesian neural networks with prior distributions are well known to induce regularization effect [41]; in particular, using Gaussian priors is equivalent to weight decay (L_2 regularization) [42].

During the forward pass, weights are sampled from the variational posterior $q(\boldsymbol{w}|\boldsymbol{\theta})$. Now, during the backpropagation, the issue is that one cannot get a gradient of the sampled points, because the sampling operation cannot be differentiated. To avoid this issue, the following reparameterization trick is used [43]. A sampled weight, w, is obtained by sampling a parameter-free distribution (the unit Gaussian), which is then scaled by a standard deviation $\boldsymbol{\sigma}$ and shifted by a mean $\boldsymbol{\mu}$. We parameterize the standard deviation point-wise as $\boldsymbol{\sigma} = \log(1 + \exp(\boldsymbol{\rho}))$ to have $\boldsymbol{\sigma}$ always non-negative. Thus the sample is $\boldsymbol{w} = \boldsymbol{\mu} + \log(1 + \exp(\boldsymbol{\rho})) \odot \boldsymbol{\epsilon}$, where $\odot$ is point-wise multiplication, $\boldsymbol{\epsilon}$ is drawn from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Hence the variational posterior parameters are $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\rho})$ [31]. In our framework we use Gaussian priors with its parameters being $(\boldsymbol{\mu}_p, \boldsymbol{\sigma}_p)$. For the reasons mentioned in Section 3.3, we include prior means $(\boldsymbol{\mu}_p)$ in training procedure.

3.2. Maximum likelihood estimation

In addition to the Bayesian approach, we also introduce the Maximum Likelihood Estimation (MLE) method—a popular frequentist approach. We do it to compare both methods in their capabilities to quantify uncertainties. Parameters of the MLE model are deterministic, but we take the double number of outputs compared to the deterministic counterpart. They stand for means and non-constant (heteroscedastic) standard deviations [33], thus yielding distributions as the outputs. These non-constant standard deviations can only capture the noises in the data, MLE inherently fails to account for uncertainties in the extrapolated region. The loss function for MLE can be recovered from Eq. (13) by removing the KL divergence part (since we do not have distributions on parameters of the MLE model), and the MLE model is trained on the Gaussian negative log-likelihood loss:

$$\boldsymbol{\theta}^*_{\mathrm{MLE}} = \arg\min_{\boldsymbol{\theta}_{\mathrm{MLE}}} -\log P(\mathcal{D}|\boldsymbol{\theta}_{\mathrm{MLE}}). \tag{14}$$

3.3. Trainable priors: Use of Empirical Bayes

Since NN parameters are latent variables of the model, it is very difficult to make a proper choice of priors. If one sets the priors far from their true values, then the posterior may be unduly affected by such choice. To overcome this, we incorporate Empirical Bayes (EB) approach [44], a method that uses the observed data to estimate the prior hyperparameters. In our approach, in the training phase, we update the prior means, keeping the prior standard deviation constant. Hence we minimize the loss function by also considering gradients with respect to the prior means. This treatment enables us to obtain a good fit to the data, while at the same time giving high prediction uncertainties in the region where little or no data is available.

3.4. Loss functions for probabilistic U-Net

We modify the deterministic U-Net architectures by replacing their layers with probabilistic layers, as a result, the output of the network is a probability distribution itself. We choose Gaussian distributions to represent priors and approximate posteriors of probabilistic layers. For the Bayesian U-Net, we use loss function as given in Eq. (13).

Expectations of Eq. (13) are approximated by $\mathcal{M}$ Monte Carlo samples drawn from the approximate posterior $q(\boldsymbol{w}|\boldsymbol{\theta})$ as referred below

$$\begin{aligned}\mathcal{L}_{\mathrm{VB}} = \; & \mathrm{KL}[q(\boldsymbol{w}|\boldsymbol{\theta}) \parallel P(\boldsymbol{w})] - \mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{\theta})}[\log P(\mathcal{D}|\boldsymbol{w})] \\ \approx \; & \sum_{i=1}^{\mathcal{M}} \log q(\boldsymbol{w}^{(i)}|\boldsymbol{\theta}) - \log P(\boldsymbol{w}^{(i)}) - \log P(\mathcal{D}|\boldsymbol{w}^{(i)})\end{aligned} \tag{15}$$

If we substitute Gaussian probability density functions, the expression in the RHS of Eq. (15) turns out to be as given in Eq. (16). We consider '$\mathcal{G}$' probabilistic parameters (Gaussian distributions) for our Bayesian U-Net, where every distribution is parameterized by its mean and standard deviation values. Since the standard deviations $\boldsymbol{\sigma}$ must be positive, we first train the network on untransformed standard deviations $\boldsymbol{\rho}$ which are later transformed to $\boldsymbol{\sigma}$ through soft-plus function. Also, for the reasons discussed in Section 3.3, we involve prior means, $\boldsymbol{\mu_p}$, in the training procedure as well. Hence parameters to be learned in the training procedure are $\boldsymbol{\theta}_{\mathrm{VB}} = (\boldsymbol{\theta}, \boldsymbol{\mu_p})$. Finally, the loss function for the Variational Bayes is given as follows

$$\begin{aligned}\mathcal{L}_{\mathrm{VB}}(\mathcal{D}, \boldsymbol{\theta}_{\mathrm{VB}}) \approx \sum_{i=1}^{\mathcal{M}} \Bigg[& \sum_{j=1}^{\mathcal{G}} \left(-\log\left(\sqrt{2\pi}\ \sigma_j^{(i)}\right) - \frac{\left(w_j^{(i)} - \mu_j^{(i)}\right)^2}{2\left(\sigma_j^{(i)}\right)^2} + \log\left(\sqrt{2\pi}\ \sigma_p\right) + \frac{\left(w_j^{(i)} - (\mu_p)_j^{(i)}\right)^2}{2\sigma_p^2} \right) \\ & - \sum_{k=1}^{N} \sum_{l=1}^{\mathcal{F}} \left(-\log\left(\sqrt{2\pi}\ d_\sigma^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)})\right) - \frac{\left(u_l^{(k)} - d_\mu^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)})\right)^2}{2\left(d_\sigma^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)})\right)^2} \right) \Bigg]\end{aligned} \tag{16}$$

where

$$\begin{aligned} d_\sigma^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)}) &= \log(1 + \exp(d_\rho^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{w}^{(i)}))), \\ w_j^{(i)} &= \mu_j^{(i)} + \sigma_j^{(i)} \epsilon_j^{(i)}, \epsilon_j^{(i)} \sim \mathcal{N}(0, 1), \\ \sigma_j^{(i)} &= \log(1 + \exp(\rho_j^{(i)})). \end{aligned} \tag{17}$$

$(\boldsymbol{d_\mu}(\mathbf{f}, \boldsymbol{w}), \boldsymbol{d_\rho}(\mathbf{f}, \boldsymbol{w}))$ are the outputs at the penultimate layer of the Bayesian U-Net, which stand for means and heteroscedastic (non-constant) standard deviations. And the last output layer is a distribution layer with the same parameters. Since $(\boldsymbol{d_\mu}(\mathbf{f}, \boldsymbol{w}), \boldsymbol{d_\rho}(\mathbf{f}, \boldsymbol{w}))$ are variables in themselves, in order to get the prediction one needs to sample over this output distribution. N, $\mathcal{F}$ are total number of training examples and dof per problem respectively. σ_p stands for the standard deviation of each the prior, which is kept constant in the training procedure. Optimized parameters, $\boldsymbol{\theta}^*_{\mathrm{VB}} = (\boldsymbol{\theta}^*, \boldsymbol{u}^*_{\boldsymbol{p}}$, for the Variational bayes case are obtained by minimizing the above loss function:

$$\boldsymbol{\theta}^*_{\mathrm{VB}} = \arg\min_{\boldsymbol{\theta}_{\mathrm{VB}}} \mathcal{L}_{\mathrm{VB}}(\mathcal{D}, \boldsymbol{\theta}_{\mathrm{VB}}) \tag{18}$$

Once the optimized parameters are computed, we replace the true posterior $P(\boldsymbol{w}|\mathcal{D})$ in Eq. (12) with the variational posterior $q(\boldsymbol{w}|\boldsymbol{\theta}^*)$ to get the predictive distribution:

$$P(\mathbf{u}|\mathbf{f}, \mathcal{D}) = \int P(\mathbf{u}|\mathbf{f}, \boldsymbol{w}) P(\boldsymbol{w}|\mathcal{D}) \mathrm{d}\boldsymbol{w} \approx \int P(\mathbf{u}|\mathbf{f}, \boldsymbol{w}) q(\boldsymbol{w}|\boldsymbol{\theta}^*) \mathrm{d}\boldsymbol{w} \tag{19}$$

The resultant predictive distribution can be approximated by Monte Carlo integration of Eq. (19) by sampling weights over optimized distributions, $\widetilde{\boldsymbol{w}}_t \sim q(\boldsymbol{w}|\boldsymbol{\theta}^*)$. At last for a given input force array, $\boldsymbol{f}$, probabilistic displacement prediction is obtained as an output. We represent this output distribution by the mean $\mathcal{U}_\mu(\mathbf{f}, \boldsymbol{w})$ and the standard deviation $\mathcal{U}_\sigma(\mathbf{f}, \boldsymbol{w})$ of the prediction, $P(\mathbf{u}|\mathbf{f}, \mathcal{D})$. This is done by taking mean and standard deviation of T stochastic forwarded passes for the same input as follows:

$$\begin{aligned} \mathcal{U}_\mu(\mathbf{f}, \boldsymbol{w}) &\approx \frac{1}{T} \sum_{t=1}^{T} P(\mathbf{u}|\mathbf{f}, \widetilde{\boldsymbol{w}}_t) \\ \mathcal{U}_\sigma^2(\mathbf{f}, \boldsymbol{w}) &\approx \frac{1}{T} \sum_{t=1}^{T} P(\mathbf{u}|\mathbf{f}, \widetilde{\boldsymbol{w}}_t)^T P(\mathbf{u}|\mathbf{f}, \widetilde{\boldsymbol{w}}_t) - \mathcal{U}_\mu(\mathbf{f}, \boldsymbol{w})^T \mathcal{U}_\mu(\mathbf{f}, \boldsymbol{w}) \end{aligned} \tag{20}$$

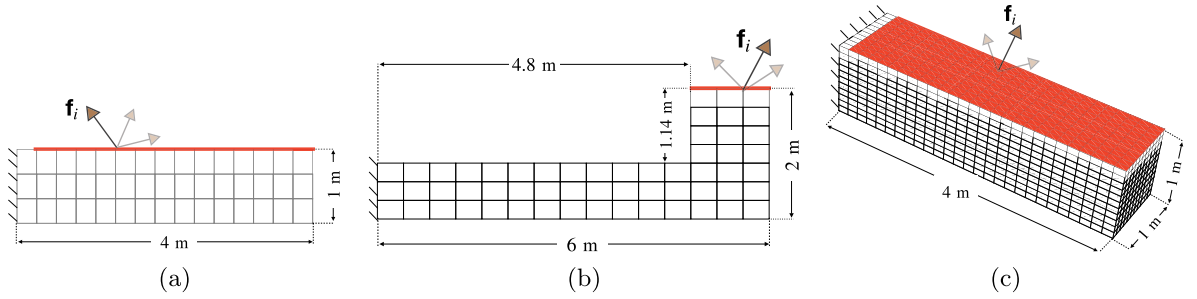


Fig. 4. Schematics of three benchmark examples (a) 2D beam, (b) 2D L-shape and (c) 3D beam. The parts of top surfaces marked in red color indicate the nodes at which random nodal forces are applied to generate training datasets.

In case of MLE, we do not place distributions over parameters, and they are discrete like in the case of the deterministic network, as in Eq. (8). In the penultimate layer, we take $(\boldsymbol{d}_{\boldsymbol{\mu}}(\mathbf{f}, \boldsymbol{w}), \boldsymbol{d}_{\boldsymbol{\rho}}(\mathbf{f}, \boldsymbol{w}))$ outputs standing for means and heteroscedastic standard deviations, which are then used to form the final Gaussian distribution output layer. Optimal parameters of the network are computed by minimizing the following loss function

$$\mathcal{L}_{\mathrm{MLE}}(\mathcal{D}, \boldsymbol{\theta}_{\mathrm{MLE}}) \approx -\sum_{k=1}^{N}\sum_{l=1}^{\mathcal{F}}\left[-\log\left(\sqrt{2\pi}\; d_{\sigma}^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{\theta})\right) - \frac{\left(\hat{u}_l^{(k)} - d_{\mu}^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{\theta}_{\mathrm{MLE}})\right)^2}{2\left(d_{\sigma}^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{\theta}_{\mathrm{MLE}})\right)^2}\right], \tag{21}$$

where

$$d_{\sigma}^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{\theta}_{\mathrm{MLE}}) = \log(1 + \exp(d_{\rho}^{(l)}(\mathbf{f}^{(k)}, \boldsymbol{\theta}_{\mathrm{MLE}}))). \tag{22}$$

At last, optimal parameters of MLE U-Net models are computed by minimizing the loss functions as

$$\boldsymbol{\theta}^{*}_{\mathrm{MLE}} = \arg\min_{\boldsymbol{\theta}_{\mathrm{MLE}}} \mathcal{L}_{\mathrm{MLE}}(\mathcal{D}, \boldsymbol{\theta}_{\mathrm{MLE}}) \tag{23}$$

4. Results

4.1. The numerical experiment procedure

4.1.1. Generation of training data from hyperelastic FEM simulations

Two 2D and one 3D benchmark problems are considered in this work, as schematically shown in Fig. 4. The Neo-Hookean hyperelastic material model is used, with Young's modulus $E = 0.5$ kPa and the Poisson's ratio $\nu = 0.4$. We use the following version of Neo-Hookean strain energy potential

$$W(\boldsymbol{F}) = \frac{\mu}{2}(I_c - 3 - 2\ln J) + \frac{\lambda}{4}(J^2 - 1 - 2\ln J), \tag{24}$$

where the invariants J and I_{c} are given in terms of deformation gradient $\boldsymbol{F}$ as

$$J = \det(\boldsymbol{F}), \quad I_{\mathrm{c}} = \mathrm{tr}(\boldsymbol{F}^T\boldsymbol{F}), \quad \text{where} \quad \boldsymbol{F} = \boldsymbol{I} + \nabla\boldsymbol{u}, \tag{25}$$

while μ and λ are Lame's parameters, which can be expressed in terms of the Young's modulus, E, and the Poisson's ratio, ν, as

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}. \tag{26}$$

As introduced in Section 2.1, for a given discretized problem, the training/testing dataset is constructed as follows. Within nodes occupying a prescribed region of the boundary (in red color in Fig. 4), a particular family of external force distribution is considered. Each loading case consists of a single excited node, while for the remaining nodes the external forces are $\mathbf{0}$. For a given training/testing example, a single node is chosen for which the external force vector is generated randomly, component-wise, from a uniform distribution within a given range of magnitude. The example is then solved with FEM, and the entire vector $\mathbf{f}$ of prescribed nodal external forces (including unloaded

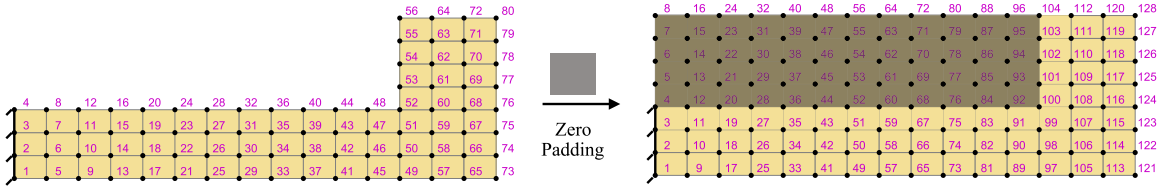


Fig. 5. Extra nodes with zero force/displacement are added to convert the data to a structured format. Node numbers are mapped accordingly to follow the assumed order of U-Net architecture.

Table 1
FE datasets. The number of DOFs with the asterisk refers to the zero-padded 2D L-Shaped mesh. 2D beam$^{\#}$ is an additional dataset used for analyzing probabilistic U-Nets in Section 4.3.

Problem	N.of FEM DOFs ($\mathcal{F}$)	Force component range [N]	Dataset size $N+M$
2D beam	128	-2.5 to 2.5	$5700 + 300$
2D beam$^{\#}$	128	-1 to 1	$3800 + 200$
2D L-shape	160 (256*)	-1 to 1	$3800 + 200$
3D beam	12 096	-2 to 2	$33688 + 1782$

nodes) and the vector $\mathbf{u}$ of computed nodal displacements are saved. The procedure is repeated for all $N + M$ examples, which creates the training/testing dataset $D = \{(\mathbf{f}_{(1)}, \mathbf{u}_{(1)}), \ldots, (\mathbf{f}_{(N+M)}, \mathbf{u}_{(N+M)})\}$.

The finite element simulations have been performed with the AceGen/AceFem framework [45] (standard library displacement-based Neo-Hookean finite elements are used). The non-linear FE problems are solved with the Newton–Raphson method, and an adaptive load-stepping scheme is used to avoid convergence issues for large load cases. A single quad/hexahedral FE mesh per problem is only considered.

Remark: For 2D/3D beam examples the structured FE mesh is used, which is compatible with the U-Net architecture introduced in Section 2.2. In the L-shaped example, the FE mesh is not structured, which makes it impossible to directly transform it to a compatible node numbering, with a possible consequence of accuracy drop, as explained in Section 4.2.1. To correct this, a special zero-padding operation is applied to each $\mathbf{f}_{(i)}$ and $\mathbf{u}_{(i)}$ before using the dataset for training/testing, see Fig. 5. Note here that unstructured meshes can be handled in several other ways. One way would be to embed a structured grid on the unstructured mesh and map the unstructured nodal values to the structured nodes. Another promising approach would be to use recently developed graph networks [46,47]. These more sophisticated approaches are, however, out of the scope of the present work.

The datasets are randomly split into training sets, N (95%), and testing sets, M (5%). The characteristics of FE meshes and datasets for all three problems are provided in Table 1.

4.1.2. Implementation and training of U-Nets

For the 2D cases, we use 3 level U-Net architectures as in Fig. 2, at each level, we apply two convolutional operators with 3×3 filters with c=128 channels in the first level. For Bayesian U-Nets, we replace half of the layers with probabilistic layers (one layer out of two at each level is replaced with a probabilistic layer). For the 3D case, we use a 4 level U-Net architecture, we apply two convolutional operators with $3 \times 3 \times 3$ filters with c=128 channels in the first level. Additionally, for both cases, we use batch-normalization on each layer. This technique standardizes the inputs to a layer for each mini-batch [48]. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Training: Network is trained by minimizing loss function for the given training dataset, minimization is performed using Adam optimizer, a well-known adaptive stochastic gradient-descent algorithm. We set the learning rate to 1×10^{-4} and set other optimizer parameters as per recommendations [49]. For the Monte Carlo sampling of loss function ($\mathcal{L}_{\mathrm{VB}}$) in Eq. (16), we use *Flipout* estimator [50] with its recommended parameter values. Trainings of deterministic and probabilistic versions of U-Net are carried out using Keras [51] and Tensorflow-probability [52] libraries respectively. All the implementations are done on Tesla V100-SXM2 GPU, on HPC facilities of the University of Luxembourg [53] using a batch size of 4 and 600/75 epochs for 2D/3D cases. All the experiments

in this work are performed using a single-precision arithmetic ('float32'), which is the usual default choice for all the deep learning libraries. The use of double-precision increased the memory requirements and the training time, without any improvement in the accuracy, and hence is unnecessary. (For the 2D beam example, double precision implementation took 4 times the training time that of the single-precision implementation.)

Since the prediction of Bayesian U-Net is a distribution, we take 300 stochastic forward passes for the same input to get the mean and uncertainty of the prediction ($T = 300$ in Eq. (20)).

4.1.3. Validation metrics for the testing phase

For the test set $\{(\mathbf{f}_1, \mathbf{u}_1), \ldots, (\mathbf{f}_M, \mathbf{u}_M)\}$, we use the following mean absolute error norm as the validation metric:

$$e_m = \frac{1}{\mathcal{F}} \sum_{i=1}^{\mathcal{F}} |\mathcal{U}(\mathbf{f}_m)^i - \mathbf{u}_m^i|. \tag{27}$$

$\mathcal{F}$ is the number of dofs of the mesh. For mth test example, $\mathcal{U}(\mathbf{f}_m)$ is the deterministic network prediction and $\mathbf{u}_m$ is the finite element solution. To have a single validation metric over the entire test set, we compute the average mean norm $\bar{e}$ and the corrected sample standard deviation $\sigma(e)$ as follows:

$$\bar{e} = \frac{1}{M} \sum_{m=1}^{M} e_m, \qquad \sigma(e) = \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} (e_m - \bar{e})^2}. \tag{28}$$

(Note: It is the standard deviation of averaged errors across the test set, not the standard deviation of all errors.)

In the case of Bayesian U-Nets, the output of the network is a probability distribution, for that reason, we sample over the output distribution by taking multiple forward passes as described in Eq. (20). Mean over these samples, $\mathcal{U}_\mu(\boldsymbol{f}_m)$, is taken as the mean prediction of the Bayesian U-Net, while the standard deviation of these samples, $\mathcal{U}_\sigma(\boldsymbol{f}_m)$, gives us the confidence intervals of predictions. Now the error norm for mth test example is given as

$$e(\mathcal{U}_\mu(\mathbf{f}_m), \mathbf{u}_m) = \frac{1}{\mathcal{F}} \sum_{i=1}^{\mathcal{F}} |\mathcal{U}_\mu(\mathbf{f}_m)^i - \mathbf{u}_m^i|. \tag{29}$$

Again the average error norm and the corrected sample standard deviation for all test examples is computed as

$$\bar{e} = \frac{1}{M} \sum_{i=1}^{M} e(\mathcal{U}_\mu(\mathbf{f}_m), \mathbf{u}_\mathbf{m}), \qquad \sigma(e) = \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} \left(e(\mathcal{U}_\mu(\mathbf{f}_m), \mathbf{u}_m) - \bar{e}\right)^2}. \tag{30}$$

4.2. Deterministic U-Nets

4.2.1. Advantages of the U-Net convolutional architecture

U-Nets vs. fully-connected NNs

U-Nets leverage the fact that nearby nodes of the FEM mesh show strong local correlation, and provide computationally efficient topology that is able to capture non-linearities. However, if we had to use a fully connected neural network to capture these non-linearities, the number of latent parameters of this network would be significantly larger as compared to that of the U-Net.

To show this effect, we consider the simplest fully connected network, with only input and output layers, without no hidden layers nor activation functions, as a surrogate model for the 3D beam example (as in Fig. 4(c)). This example has 12096 dof, so the dimension of the input and output layer is 12096 each. Because of the absence of hidden layer/activation functions, this network is only able to capture a linear response of the force–displacement relationship. In order to have the best-linearized approximation, we initialize trainable parameters of the fully connected network with the inverse of the FEM stiffness matrix. Table 2 shows that the fully connected network (which is an inaccurate assumption) has 1.5 more parameters than the deterministic U-Net, while the accuracy is greatly reduced. In order to do a better (non-linear) approximation, one would need to use a multi-layer fully connected network, which would require even more parameters, and hence the training time would be significantly higher. Hence, the choice of U-Nets makes complete sense, in particular for complex non-linear problems.

Table 2
Comparison of U-Net vs. feed forward network for 3D Beam example.

NN type	N. of trainable parameters	$\bar{e}$ [m]	$\sigma(e)$ [m]
Deterministic U-Net	94.1 E+6	0.6 E−3	0.3 E−3
Fully-connected	146.3 E+6	7.0 E−3	8.0 E−3

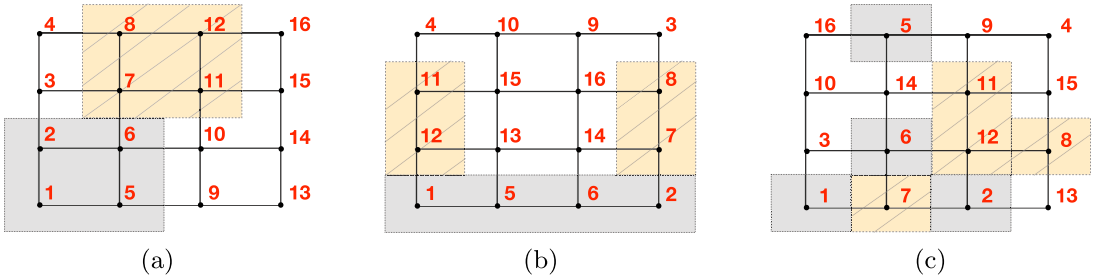


Fig. 6. Different node numbering strategies. (a) Numbering assumed by the TensorFlow (the preferred one; used in this work), (b) Gmsh preprocessor numbering, and (c) random numbering.

Table 3
Error metrics for the preferred and randomly ordered case for the 3D beam problem.

Ordering strategy	$\bar{e}$ [mm]	$\sigma(e)$ [mm]
Preferred ordering (as in Fig. 6(a))	0.6	0.3
Random ordering (as in Fig. 6(c))	5.4	2.8

Effect of DOF ordering

The topology of input FEM mesh plays a crucial role in the training of U-Nets, and it must be compatible with that of the U-Net architecture topology. However, different FEM pre-processors have different ways of numbering nodes. For instance, Gmsh [54], a popular FE mesh generator, first numbers corner nodes, then edge nodes followed by internal nodes, see Fig. 6(b). This is not compatible with the expected U-Net input, which effects deteriorating the predictive capabilities of the U-Net. A completely random ordering, see Fig. 6(c), performs even worse, see Table 3. To fully leverage the advantages of U-Nets, care has to be taken to order nodes properly. This is the reason why the zero-padding has been done to the L-shaped case, see the remark in Section 4.1.1, and Fig. 5.

4.2.2. Prediction accuracy

Deterministic U-Nets are trained on FEM datasets generated as described in Section 4.1.1. Below, we analyze in a more detail some selected test examples for each case, and compare their FEM and U-Net solutions. For all the examples, we show the overlap of deformed meshes obtained using FEM and U-Net models. In Figs. 7–11 and Fig. 13, gray, blue and red meshes represent undeformed configuration, U-Net solution and FEM solution, respectively. In addition to that, we also present the interpolated node-wise L_2 norm of the prediction error (the error of the nodal displacement between FEM and U-Net.

In Fig. 7, we show a test example of the 2D-beam case. A point force is applied at the corner node of the beam and the deformation of mesh is predicted using the deterministic U-Net. As we can see, the deformed mesh predicted with U-Net is overlapping with the reference FEM solution. As explained above, we also plot the nodal error field on the deformed mesh, one can observe that the error is relatively higher in the high displacement region, i.e, near the free end. The relative error for the tip with respect to its displacement magnitude is only 0.6%.

Fig. 8 shows an example of the 2D L-shape case. Again the deterministic U-Net solution is overlapping with the reference FEM solution. L_2 error contour shows that a high error trend is observed near the free end again, the relative error at the top corner node with respect to displacement magnitude is 0.4% only.

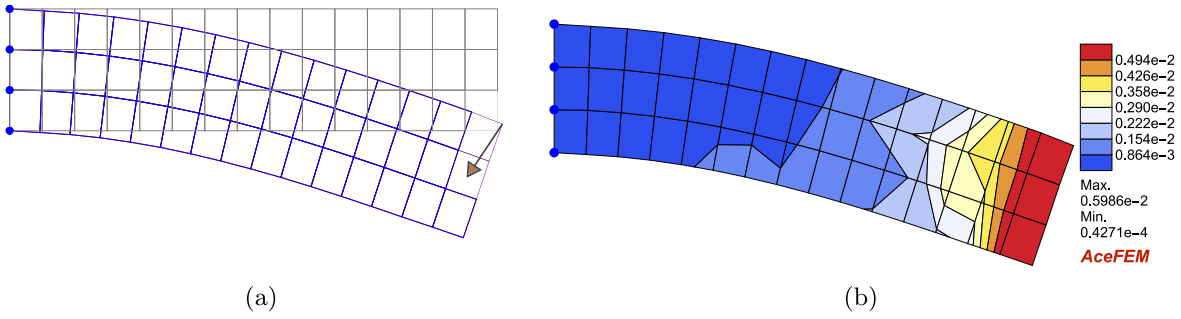


Fig. 7. Deformation of 2D beam computed using the deterministic U-Net, for the point force at the free end. (a) Comparison of deformed meshes, both blue mesh (U-Net solution) and red mesh (reference FEM solution) are overlapping. The magnitude of the tip displacement is 0.95 m. (b) L_2 error of nodal displacements between deterministic U-Net and FEM solutions.

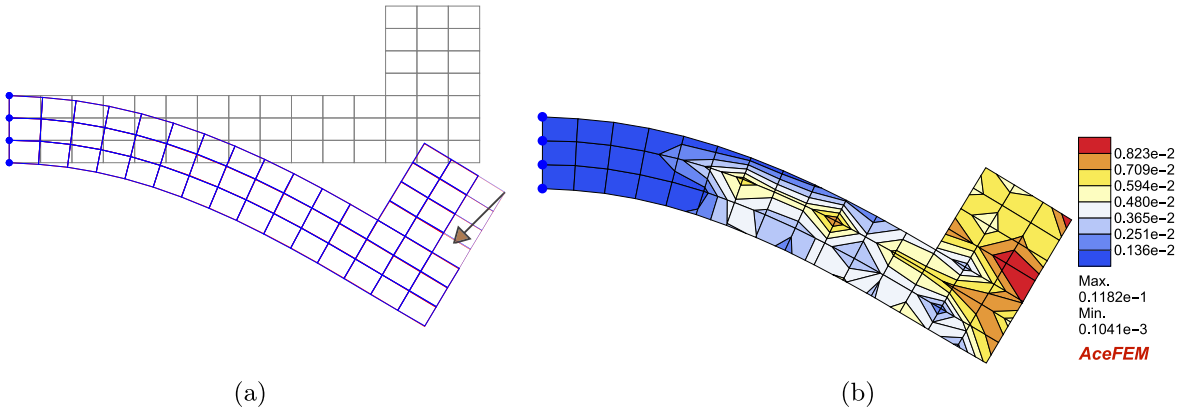


Fig. 8. Deformation of 2D-L shape computed using deterministic U-Net. (a) Initial and deformed meshes predicted using deterministic U-Net(blue) and FEM(red), the magnitude of tip displacement is 2.39 m, and (b) L_2 error of nodal displacements between deterministic U-Net and FEM solutions.

We further take a look at two 3D-beam test examples, one with the force applied near the free end and another with the force applied in the middle region of the 3D beam. For both cases, the deterministic U-Net solutions are overlapping with the FEM solutions. Insets in Fig. 9 show that the U-Net is capable of predicting high local non-linear deformations. For the first example in Figs. 9(a)–9(b), the error field shows high error region near the point of application of the force. The relative error for the tip for this case is only 0.6%. Whereas, Figs. 9(c)–9(d) shows an example with the force applied relatively near to the fixed end. In this case, a high error field is observed at the point of application of force as well as near the free end. The relative L_2 error of the tip for this example is 1.6%. From this, we can say that errors are usually higher near the nodes with higher displacement magnitudes.

Till now we looked at the prediction accuracy for individual examples, now we would like to judge the performance of deterministic U-Net over the entire test sets (5% of the generated data is designated for testing purposes). Table 4 provides such comparison in a form of averaged error over entire test sets. We can see that, on average, the error is at a reasonably low level. To extend this analysis, in Fig. 10 we plot the mean error (e) of each test example of the three benchmark problems. We sort these errors as per the increasing displacement magnitude at the point of application of force. To get a relation between displacement and mean error (e), we do a least square linear fit for all three cases. From Fig. 10, all the three examples show generally low sensitivity to the increase of displacement magnitude.

Effect of changing the distribution of applied forces

Deterministic U-Net has been trained by using single point load examples only, but we would like to check how it performs when multiple point load input is given for the prediction. Fig. 11 shows one such example where random multiple forces are applied on the top edge, U-Net is able to closely follow the reference FEM solution.

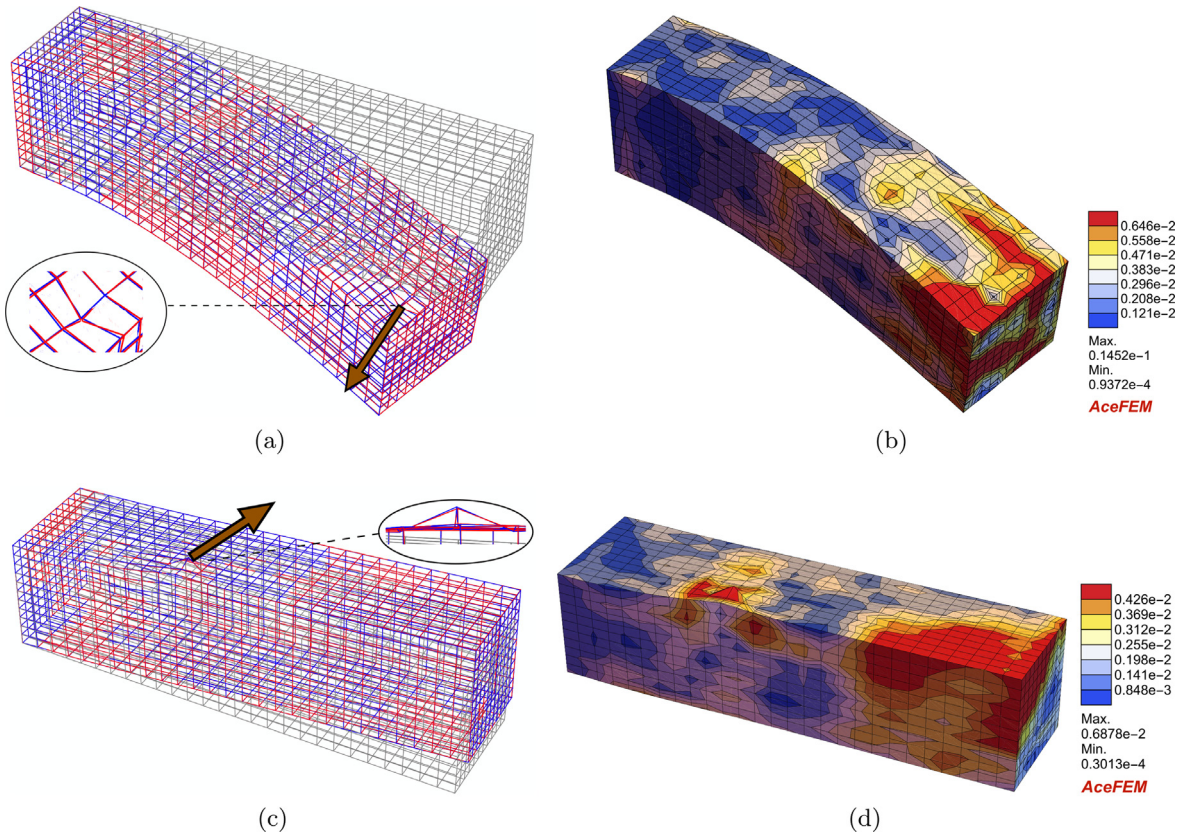


Fig. 9. Deformation of 3D beam computed using deterministic U-Net (blue) for two force cases, for comparison FEM solution (red) is presented. (a)&(c) deformed meshes for both examples. The magnitude of tip displacement for the first case (force near the free end) is 1.1 m and for the second case (force in the middle region) is 0.26 m. High localized deformations are shown in the insets. (b)&(d) L_2 error of nodal displacements between deterministic U-Net and FEM solutions.

Table 4
Error metrics for 2D and 3D test sets for predictions using deterministic U-Net. M stands for the number of test examples, and $\bar{e}$ and $\sigma(e)$ are error metrics defined in Section 4.1.3.

Example	M	$\bar{e}$ [m]	$\sigma(e)$ [m]
2D Beam	300	0.3 E−3	0.2 E−3
2D L-Shaped	200	0.8 E−3	0.4 E−3
3D Beam	1782	0.6 E−3	0.3 E−3

Fig. 11(b) shows the L_2 norm of the error across the beam, it shows a different trend for this example. Though the deformation is higher in the free end region and at the point of application of forces, a higher error is observed at a different location also. Solution accuracy of multiple point load cases can be improved by incorporating multiple point loads in the training phase. Also, the relative error for the tip with respect to its displacement magnitude for this example is 0.6%.

In most engineering applications, we are interested in the cases where force is applied in a given prescribed region of interest (e.g., the Neumann boundary). Here, we would like to check how U-Net performs when this assumption is violated, i.e, we apply forces on the nodes which were not involved in the training procedure. To do so, we use the same 2D beam case with the training set generated by applying point forces on the top edge (indicated by the red line in Fig. 4 in Section 4.1.1). What we change is the prediction phase, during which we apply forces on nodes located on the vertical free edge of the beam (see schematics in Fig. 13). In the example, we apply

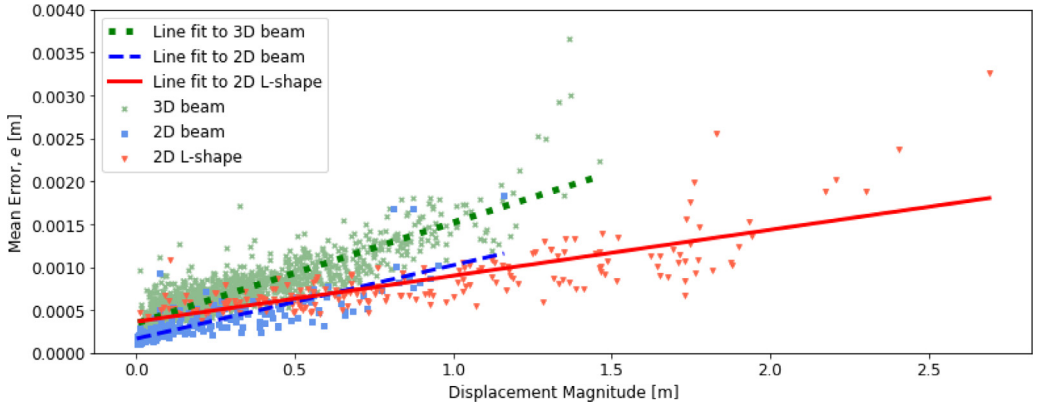


Fig. 10. Mean errors (e) for all test examples of three benchmark cases. The regression lines $y \propto 0.0008 \times x$ (2D-beam), $y \propto 0.0005 \times x$ (2D-L shape), $y \propto 0.001 \times x$ (3D beam) show low sensitivity of the deterministic U-Nets to displacement magnitudes.

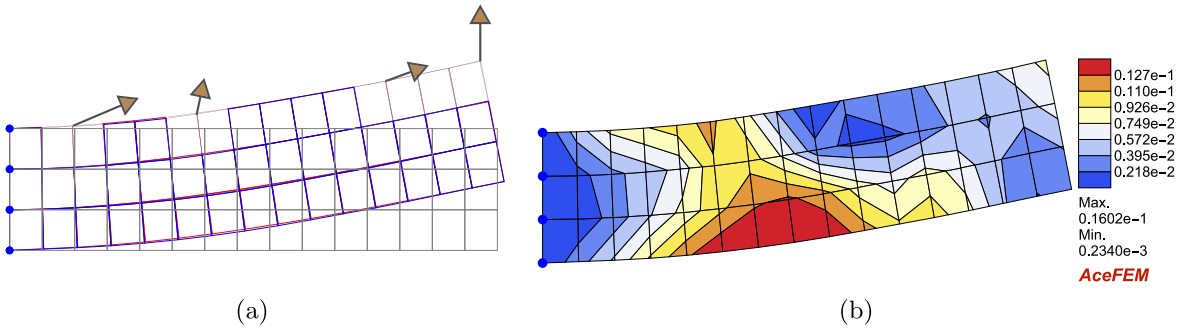


Fig. 11. Deformation of the 2D beam subjected to multiple point loads. (a) Comparison of deformed meshes predicted with deterministic U-Net and FEM, the magnitude of tip displacement is 0.55 m. (b) L_2 error of nodal displacements between deterministic U-Net and FEM solution.

a vertical force of 1.5 N on each of the 4 nodes of the free edge of the beam. Fig. 13(a) shows that mesh (blue) predicted with U-Net deviates more and more from the true FEM solution, as we move away from the training line. The U-Net solution is much worse when the force is applied on the 4th node as compared to the 2nd, i.e. when the point of force application is farthest from the training line. In Fig. 13(b), we plot the mean and maximum errors of all 4 examples, and we can observe a significant accuracy drop reaching two orders of magnitude when predicting outside the training region. Also, we can see that the errors are increasing when moving away from the training dataset. This proves that the U-Nets extrapolate predictions poorly when moving away from the training range in spatial directions.

Training convergence The choice of the amount of training data and the appropriate neural network architecture are two important criteria in the case of neural network surrogate modeling. This is crucial to ensure that neither underfitting nor overfitting is observed. For all the cases in this work, training convergence is ensured by observing loss plots of training and validation errors, i.e., the training error does not decrease, and validation error does not go higher with the number of epochs. For the reference, the loss plots for 2D cases are shown in Fig. 12.

4.3. Probabilistic U-nets

The goal of our probabilistic U-Net framework is to get reliable predictions and uncertainty associated with those predictions. Further in this section, we will check this for the case of data and model uncertainties for selected examples analogous to the deterministic case.

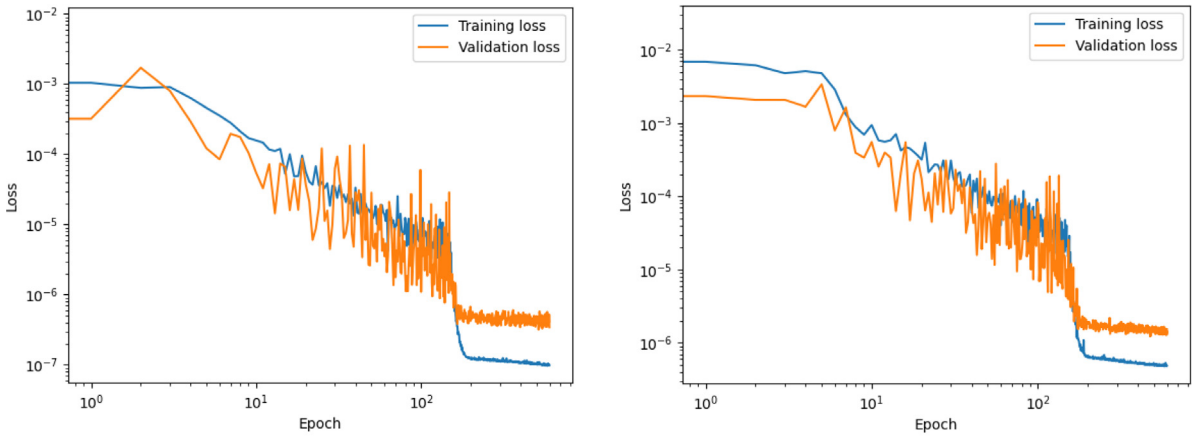


Fig. 12. Mean squared error log-loss plots for (a) 2D beam and (b) 2D L-shape case.

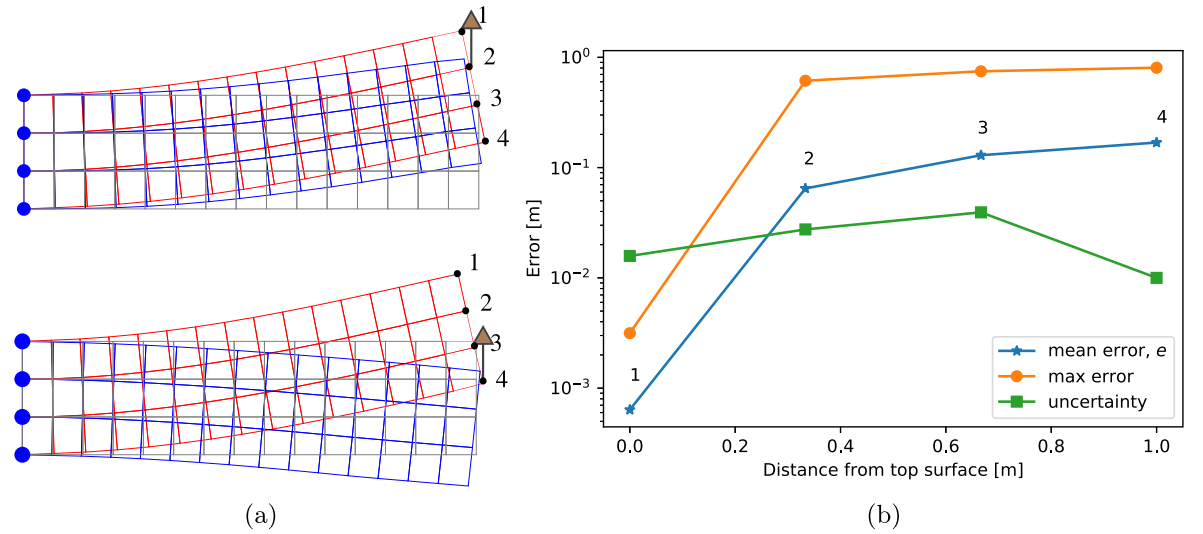


Fig. 13. Application of point loads on the nodes away from the training region. (a) Deformed meshes obtained by FEM (red) and with U-Net (blue) when point load is applied on 2nd or 4th node. (b) Mean error (e) and maximum error for each of the 4 point loads cases. Green line shows the uncertainty prediction of Bayesian U-Net, for the node on which the force is applied.

4.3.1. Prediction accuracy

We train the probabilistic U-Net framework on the same datasets as used in deterministic cases. Because the output of the network is a distribution, we make 300 stochastic forward passes to get the mean and uncertainty predictions for a given input. Mean prediction of Bayesian U-Net is treated as the solution of the network, whereas uncertainty predictions give information of credible intervals of predictions. Table 5 gives the error metrics for the Bayesian U-Net predictions over the entire test sets, for comparison we have shown the errors of deterministic counterparts as well.

Similar to the deterministic case, we do the analysis of the error metric (e) for all the test examples predicted using Bayesian U-Net this time. Fig. 14 shows errors sorted as per the increasing displacement magnitudes of the point of application of forces. We perform a least-squares line fit to the error data. Slopes for 2D-beam and 2D L-shape cases are small, proving a little sensitivity of errors to the displacement magnitudes.

Hereafter we focus on particular examples to get more insights on Bayesian U-Net predictions. Similar to the deterministic cases, we take node-wise L_2 norm of the error (Error of FEM and mean prediction of Bayesian U-Net)

Table 5
Error metrics for 2D test sets using Bayesian U-Nets. D = Deterministic, VB = Variational Bayes.

Example	M	$\bar{e}$ [m]	$\sigma(e)$ [m]
2D Beam (VB)	300	1.3 E−3	1.3 E−3
2D Beam (D)	300	0.3 E−3	0.2 E−3
2D L-Shaped (VB)	200	5.3 E−3	3.7 E−3
2D L-Shaped (D)	200	0.8 E−3	0.4 E−3

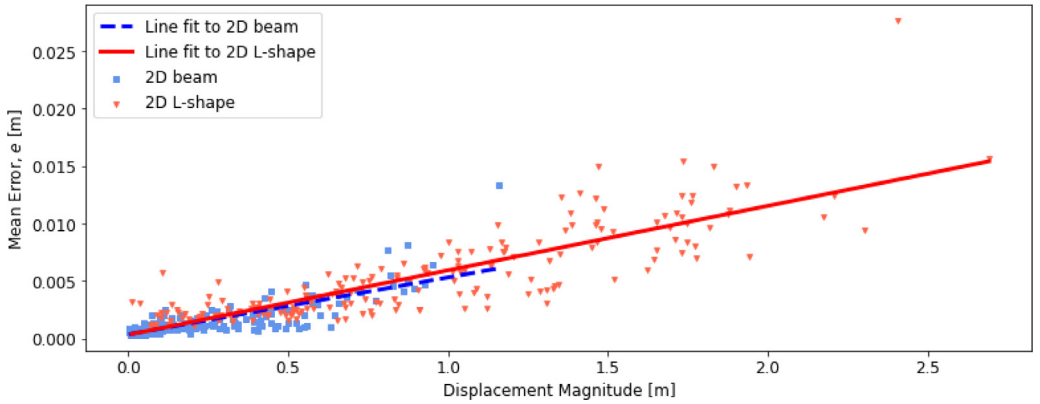


Fig. 14. Mean errors (e) for all test examples of 2D cases, for predictions using Bayesian U-Net. The regression lines $y \propto 0.005 \times x$ (2D beam), $y \propto 0.006 \times x$ (2D-L shape) show low sensitivity of Bayesian U-Net errors to displacement magnitudes.

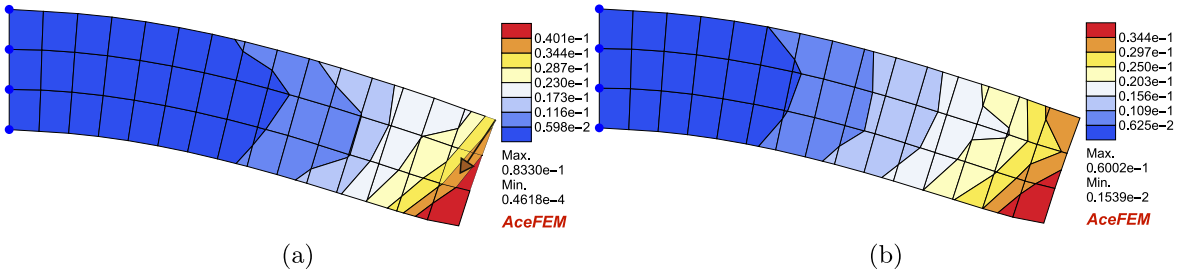


Fig. 15. Deformation of 2D Beam predicted by the Bayesian U-Net, for the same example as in Fig. 7. (a) The error between Bayesian U-Net and FEM solution plotted on the deformed mesh. (b) Uncertainty of prediction obtained using Bayesian U-Net plotted on the deformed mesh.

and also that of the uncertainty prediction from Bayesian U-Net. Both error and uncertainty values are interpolated within the element to get respective fields, which are plotted on the deformed mesh obtained using Bayesian U-Net.

We consider the same 2D-beam test case as in Fig. 7, (as in deterministic case). This time we make the prediction using Bayesian U-Net. Fig. 15 shows the comparison of error and uncertainty associated with the prediction (we plot single standard deviation values associated with the prediction of respective dof). One can see that both are strongly co-related spatially.

A similar kind of analysis is done for the multiple point load case Figure, see 11, in the deterministic section. Fig. 16 compares the error and uncertainty fields obtained using the Bayesian U-Net, we can see that they are correlated and closely follow each other as well.

Force outside training range: Let us consider a test example in which a force of 5 N is applied on the corner node, which is far away from the training range (which is −2.5 to 2.5 N). Again we compare error and uncertainty associated with the Bayesian U-Net prediction. For reference, the FEM solution is presented (red mesh) with the error contour plot. Both error and uncertainty are plotted on the deformed mesh predicted with the Bayesian U-Net. In Fig. 17, one can see that both are strongly correlated, rather both values are close to each other across the spatial

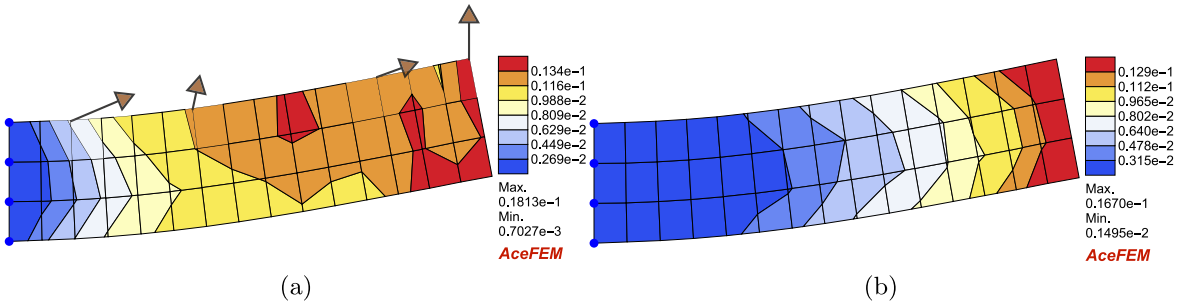


Fig. 16. Deformation of 2D Beam for multiple point forces using Bayesian U-Net, for the same example as in Fig. 11. (a) The error between Bayesian U-Net and FEM solution plotted on the deformed mesh. (b) Uncertainty of prediction obtained using Bayesian U-Net plotted on the deformed mesh.

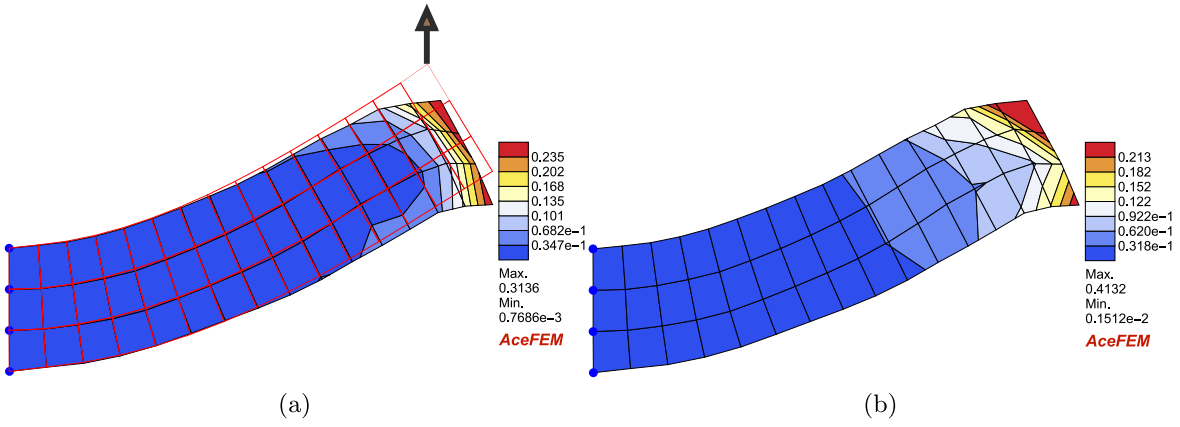


Fig. 17. Deformation of 2D Beam using Bayesian U-Net for an input force outside the training range. (a) Error between Bayesian U-Net and FEM solution plotted on the deformed mesh. (b) Uncertainty of prediction obtained using Bayesian U-Net plotted on the deformed mesh.

dimensions of the beam. Thus, the uncertainty predictions can give us an idea about the error of U-Net predictions, irrespective of whether an input is within or outside the training region.

For each of the above examples shown in Figs. 14–16, we can see that the U-Net solution is deviating from the true FEM solution, which is given by the error contour, i.e., the U-Net model is not able to fit the data exactly. And uncertainty prediction obtained using the Bayesian U-Net is able to capture this fitting error.

The example in Fig. 17 can be considered as a case of extrapolation in the sense of the magnitude of force being outside the training range. One can also think of extrapolation in the sense of applying force on the nodes which were not included in the training, i.e., extrapolation in the spatial dimensions of the geometry. To analyze such cases we consider the same example as shown in Fig. 13 in the deterministic Section 4.2.2. In Fig. 13(b) we have shown the uncertainty of Bayesian U-Net prediction (one standard deviation), for the node on which point load is applied. As one can infer, Bayesian U-Net is not giving reliable uncertainty estimates when we move away from the training region in spatial directions. Intuitively the predicted uncertainty should be more for the case when force is applied on the farthest node from the training line, but on contrary, we observed a low prediction uncertainty for this point. One possible explanation of this limitation is, gradients w.r.t the spatial dimensions are not available neither in the data nor in the U-Net models. Hence there is no natural way of extrapolating information of solutions or uncertainties.

Hereafter we focus on displacement prediction of a single dof with Bayesian U-Net. We do this to see how the associated uncertainty varies with the value of input force, depending on whether the input is within or outside the training range. In Figs. 18–21, we keep a constant direction of the input force, but gradually increase the magnitude and study the displacement prediction using Bayesian U-Net. The output of the Bayesian U-net is the displacement

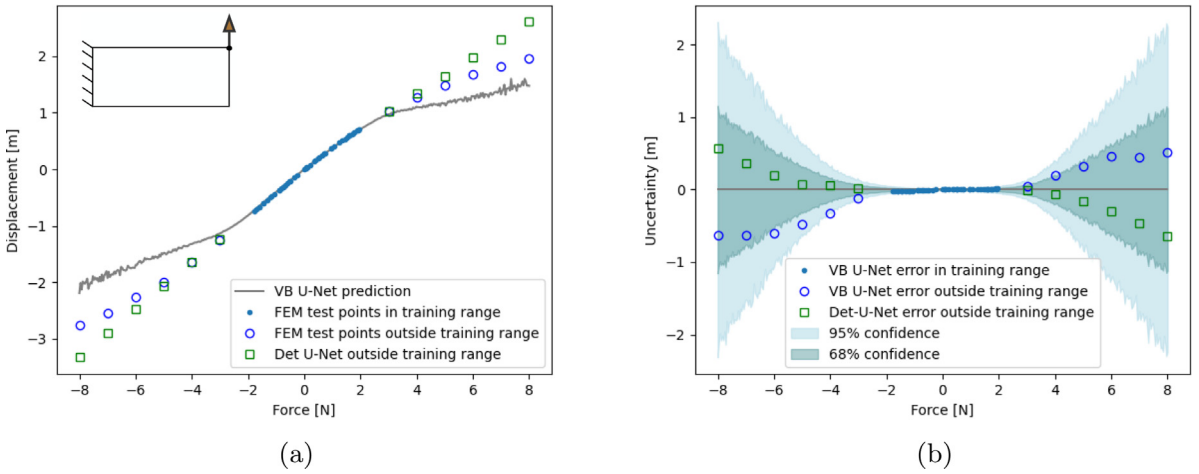


Fig. 18. Outputs of Bayesian U-Net when a range of forces is applied on the corner node of the 2D Beam (see inset). (a) The magnitude of Y-displacement of the corner node predicted with Variational Bayesian U-Net. (b) The uncertainty associated with the predictions of displacement solutions.

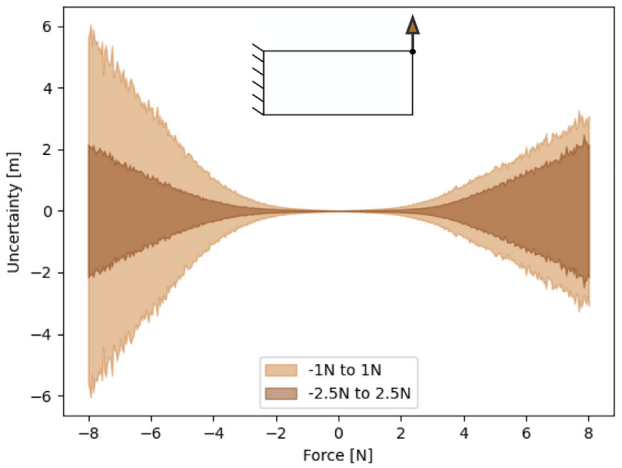


Fig. 19. Uncertainty intervals for the Y-displacement of the corner node of the 2D beam (see inset), predicted using the Bayesian U-Nets trained on different training sets. Uncertainty reduces with the increase of force range in training data.

solution and the uncertainty associated with it, in Figs. 18–20 we provide separate plots for both of these outputs. Whereas in Fig. 21 we only study the prediction uncertainties for noisy data cases.

2D Beam: We apply multiple vertical forces varying from −8 N to 8 N on the corner node of the beam and predict its displacements using the Bayesian U-Net. Fig. 18(a) gives the prediction of displacement magnitude, as one can see prediction matches with test FEM solution within the training region. Outside the training region, Bayesian U-Net prediction deviates from the FEM solution. For reference, we provide deterministic U-Net solutions as well, even they deviate from FEM solutions outside the training range. Whereas Fig. 18(b) gives confidence intervals associated with these predictions. One can see that network has very little uncertainty i.e. it is confident in the region of training data (−2.5 to 2.5 N) but as one moves away, the uncertainty of the prediction increases. We can also see that 95% confidence is able to capture the error of Bayesian U-Net predictions outside the training region, for reference, errors of deterministic U-Nets are presented as well.

In this paragraph, we compare uncertainty intervals for Bayesian U-Nets trained on two different datasets. In addition to the existing 2D beam dataset (force range: −2.5 to 2.5 N), we consider another training dataset with a lower force range this time (force range: 1 to 1 N). Fig. 19 shows the comparison of uncertainty intervals for these

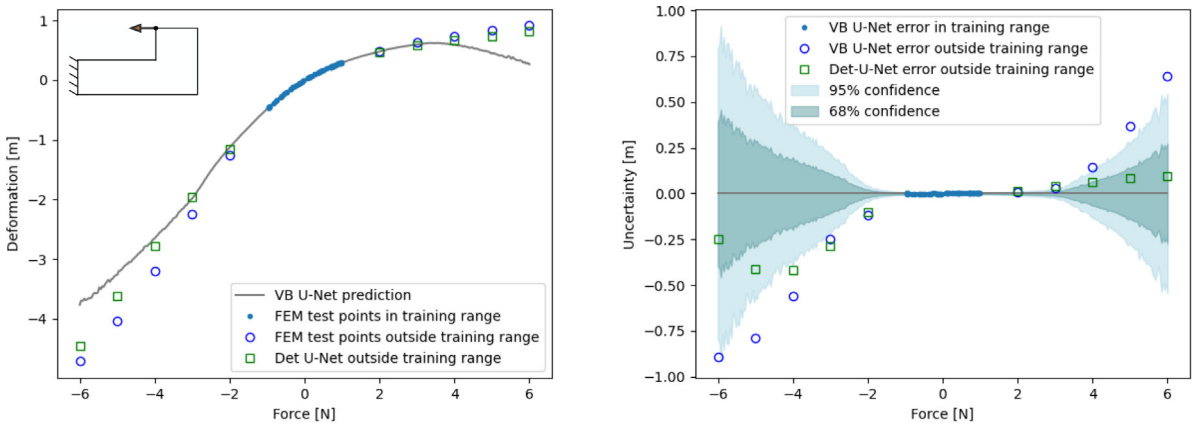


Fig. 20. Outputs of Bayesian U-Net when a range of forces is applied on the inner corner node of the 2D L-shape (see inset). (a) The magnitude of X-displacement of the inner corner node predicted with Variational Bayesian U-Net. (b) The uncertainty associated with the predictions of displacement solutions.

two cases, as the range of input force in the training set is decreasing, Bayesian U-Net tends to get more uncertain about its predictions in higher force ranges, which follows the common intuition.

2D L-shape: This training dataset was created by applying point forces in the range of -1 N to 1 N as shown in Table 1. In order to see how prediction uncertainty varies with the input forces, we apply multiple forces in a horizontal direction varying from -6 N to 6 N on the inner corner of the L-shape and predict its displacements using Bayesian U-Net. Fig. 20(a) shows how displacement magnitude changes with applied force values. As we start to move away from the training region, the Bayesian U-Net solution deviates from the FEM solution. For reference, we have plotted the deterministic U-Net solutions as well. Fig. 20(b) gives the uncertainty associated with the prediction. Again the network is very confident in the training data region. But as the force value goes outside the training range, uncertainty tends to increase, for the reference, errors of deterministic U-Nets are presented as well.

4.3.2. Noisy data case

In all the cases above, the U-Net models have been trained with numerical FEM datasets which can be regarded as noiseless. However, in many practical applications, especially when working with experimental data, the data noises exist and can originate from various sources, such as measurement errors, errors associated with tools, human errors, etc. In this section, we would like to demonstrate that our framework is capable of capturing these data noises. To show that, we add random noises to our existing FEM datasets, and check how MLE- and Variational Bayes U-Nets perform in capturing these noises in terms of the predicted uncertainties.

For both 2D beam and 2D L-shape cases, we modify the existing datasets (of the input force range -1 N to 1 N as shown in Table 1) by incorporating random noises to displacement values. When the magnitude of applied force is less than 0.7 N, we add a random noise (from a continuous uniform distribution) within 20% of the real displacement solutions, i.e., when $\|\boldsymbol{f}\|_2 \leqslant 0.7$ we set $\boldsymbol{u} \rightarrow (1+\gamma)\boldsymbol{u}$, where $\gamma \sim \mathrm{U}(-0.2, 0.2)$. Now, the probabilistic networks (MLE and Variational Bayes) are trained using these noisy datasets. In the prediction phase, we apply forces to a single chosen corner node in a single d$-$4 N to 4 N (see insets in Fig. 21). Then we analyze the predicted uncertainties associated with displacements of respective nodes to which the force has been applied, and how they relate to the level of input force noises.

Figs. 21(a) and 21(c) show that the MLE approach is able to capture the noises in the training data region, although it fails to produce reliable uncertainty estimates outside that region (extrapolated region). The network is very confident in predictions even though we move away from the training region, and the prediction errors there are clearly visible. Whereas from Figs. 21(b)–21(d), we can see that the Variational Bayes approach is able to capture both effects: the effect of noises in the data, as well as the desired effect of gradually increasing uncertainty as we move away from the training region.

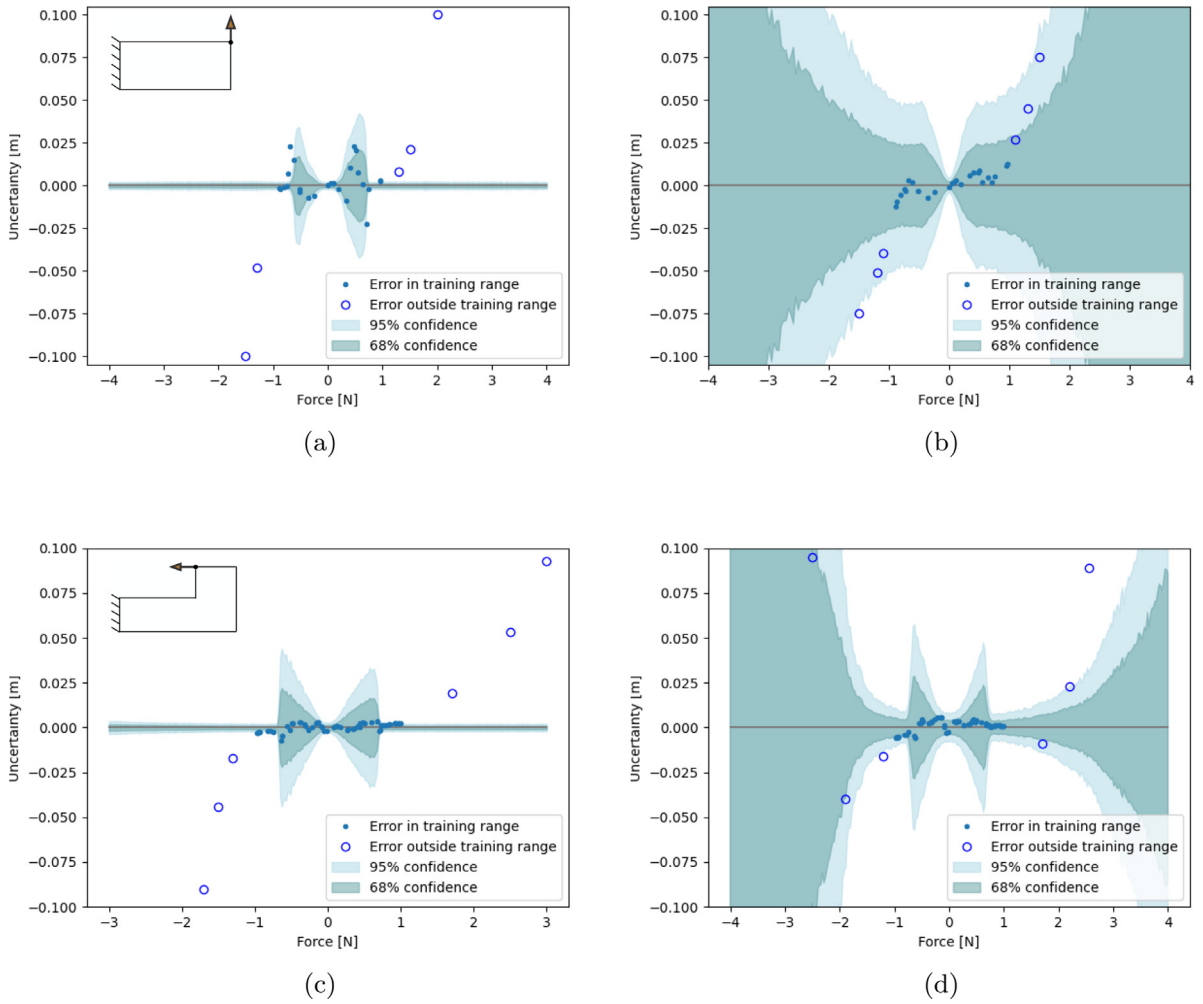


Fig. 21. Uncertainty predictions for noisy data cases using probabilistic U-Nets. For the 2D beam case (upper row), prediction is done for the Y-displacement of the corner node. For 2D L-shape (lower row), prediction is done for the X-displacement of the inner corner node (see insets). (a)&(c) Uncertainty predictions using MLE approach. MLE fails to capture the uncertainty outside the training region. (b)&(d) Uncertainty predictions using Variational Bayes approach. VB is capable to capture the uncertainty outside the training region.

4.4. Prediction and training times

Prediction Times

Although networks are trained on Graphical Processing Units (GPU), predictions are computationally inexpensive on user end Central Processing Units (CPU) as well. Also, since recent years, GPU cloud computing is easily accessible, one can leverage GPU support over the internet. All these factors make our framework easily deployable to the user end. Table 6 gives the comparison of prediction times for different examples on GPU as well on CPU.

For some of the force values in the testing set, the FEM solution took more than 3s. Hence under identical computational resources, deterministic U-Net gave 31 times speedup. Another important point to mention here is, both deterministic and Bayesian U-Net, individually take the same time for prediction irrespective of the value of the input (applied force). In the case of the FEM, solution time evolves with the value of applied force. This is because we use an iterative solver and adaptive load-stepping scheme to avoid convergence issues for large load cases. Hence on local, we can expect much more speedup than 31 times when we go towards the higher input force

Table 6
Prediction times of deterministic U-Net on CPU, GPU. Under similar computational resources on CPU, U-Net shows 31 times speedup, which can be even more depending on the boundary conditions. Whereas GPU shows nearly 350 times speedup.

Type	dof	$t\_\text{fem}_{\text{CPU}}$ [s]	t_{CPU} [s]	t_{GPU} [s]	$\frac{t\_\text{fem}_{\text{CPU}}}{t_{\text{CPU}}}$	$\frac{t\_\text{fem}_{\text{CPU}}}{t_{\text{GPU}}}$
2D Beam	128	0.123	0.005	0.001	25	123
2D L-shape	256	0.120	0.007	0.001	17	120
3D Beam	12 096	3.1	0.1	0.009	31	345

Table 7
U-Net training times, t_{train}. D = Deterministic, VB = Variational Bayes.

Example	Dataset size, N	t_{train} [min]	N. of trainable parameters
2D Beam (D)	5700	131	7.5 E+6
2D Beam (VB)	5700	226	15.1 E+6
2D L-Shaped (D)	3800	78	7.5 E+6
2D L-Shaped (VB)	3800	143	14.6 E+6
3D Beam (D)	33 688	1060	94.1 E+6

values. Deterministic U-Nets gave nearly 350 times speed up when predictions were done using GPU. Even with the high dimensional 3D examples, U-Net did not take more than 10 ms, thus satisfying the real-time constraint.

The time of prediction of Bayesian U-Net is the time of sampling over output distribution, which is as long as 300 stochastic forward passes in our case. For the above example (for both 2D beam and 2D L-shape), 300 forward passes for a single test example took 0.1 secs. Compared to the deterministic case, the average time for the prediction of single-pass is less (0.3 ms) because of the efficient utilization of batch prediction. Hence even Bayesian inference takes very little time in the prediction phase.

Training Times

For any neural network, the training phase of the model is the most resource-intensive task. Hence modern machine learning open source libraries such as Tensorflow, Keras, PyTorch are optimized to work with GPUs. GPU has a parallel structure that offers faster computing and increased efficiency compared to the user end computer with its CPU. Table 7 gives GPU training times for different datasets for both deterministic and probabilistic U-Nets.

Bayesian U-Nets have more parameters to be trained, additionally, we need to sample over the approximate posterior as described in Section 3.4. Hence training times for Bayesian U-Nets are significantly higher than for the deterministic counterparts. As the size of the problem grows, training time proportionally increases as well. Hence the training time for the 3D beam case is higher compared to 2D cases. Note however, that this time can be reduced by opting alternate topologies of U-Nets, and one way of doing so is keeping a constant number of channels in each U-Net level instead of increasing it (which will be analyzed below).

Effect of number of channels

The training time of U-Net can be reduced by decreasing the number of trainable parameters of the model, and one of the ways to achieve this is to decrease the number of channels at each U-Net level. This can have, however, a side effect on prediction accuracy (intuitively, channels are partially responsible for capturing nonlinearities). We analyze these competing effects by performing a case study for the deterministic 3D-Beam case for architectures with different constant (not variable) number of channels, c.

Table 8 shows a comparison of training times and prediction errors. As we can see, as compared to the architecture used in Section 4.2.2, the use of 64 channels at each level gave comparable error values, while the training time is about three times lower. We can also observe that an excessive increase in the number of channels ($c = 128$) results in deterioration of not only training time but also the prediction accuracy, which can be interpreted as a well-known effect of overfitting. For reference, we have provided GPU prediction times for these networks as well.

5. Conclusions

In this work, we have proposed a deterministic/probabilistic neural network framework that is capable of accurately predicting large deformations in real-time. Although in the present work we only used artificially

Table 8
Error metrics, and training and predicting times for different deterministic U-Net architectures trained on the 3D dataset. A constant number of channels, c, is used in each level of U-Net. The use of 64 channels is optimum to achieve error and computational time trade-off. Standard 3D U-Net architecture took 1060 minutes to train on the identical dataset.

N. of channels, c	$\bar{e}$ [m]	$\sigma(e)$ [m]	t_{train} [min]	t_{GPU} [ms]
16	1.6 E−3	0.9 E−3	272	6
32	1.1 E−3	0.7 E−3	293	6.5
64	**0.8 E−3**	**0.5 E−3**	**348**	7.5
128	3.5 E−3	3.3 E−3	646	9

generated data for training, the framework can naturally assimilate experimental data as well. Because of these factors, our framework has the potential for data-driven applications requiring very fast response rates, such as patient-specific computer-aided surgery of soft human tissues.

In addition to the predictions, the proposed probabilistic framework is also capable of giving reliable uncertainty estimates. Indeed, we showed that the predicted uncertainties correlate with the prediction errors (fitting errors to FEM solution). We also showed that the uncertainties rapidly increase in the extrapolated region, which is the desired property that we expected to achieve. Additionally, we were able to capture the noises present in the data, which has been validated with two probabilistic approaches (Maximum Likelihood Estimation and Variational Bayes). As such, our framework can be seen as a step towards making real-time large-deformation simulations more trustworthy.

To the best of our knowledge, this is the first time the state-of-the-art Bayesian Neural Networks are used in the context of non-linear body deformations. We believe that this work can serve as a reference for further developments in this emerging area of research. Due to its potentially high efficiency and accuracy, as well as due to its unique probabilistic predictive capabilities, we believe that the presented framework will turn out to be useful in a wide scope of novel engineering applications.

Besides showing promising results, we also demonstrated several important limitations of the current framework. Firstly, the convolution operations that are used in our U-Nets' implementation require structured meshes. We showed in the paper possible methods to extend our framework to unstructured meshes, which can be done with a moderate effort in the future. Secondly, we observed that the proposed novel technique to quantify uncertainties in extrapolated regions does not always give reliable predictions. Bayesian U-Nets failed to give reliable credible intervals of predictions when we applied the force on the nodes which were not part of the training procedure (i.e. extrapolated data in the spatial dimensions). As discussed in the paper, it seems to be a more fundamental and challenging problem that needs a dedicated approach, which is left for future research.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 764644. Jakub Lengiewicz would like to acknowledge the support from EU Horizon 2020 Marie Sklodowska Curie Individual Fellowship *MOr-PhEM* under Grant 800150. This paper only contains the author's views and the Research Executive Agency and the Commission are not responsible for any use that may be made of the information it contains.

Stephane Bordas and Jakub Lengiewicz are grateful for the support of the Fonds National de la Recherche Luxembourg FNR grant QuaC C20/MS/14782078. Stephane Bordas received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 811099 TWINNING Project DRIVEN for the University of Luxembourg.

References

[1] S. Cotin, H. Delingette, N. Ayache, Real-time elastic deformations of soft tissues for surgery simulation, IEEE Trans. Vis. Comput. Graphics 5 (1) (1999) 62–73, http://dx.doi.org/10.1109/2945.764872.

[2] H. Delingette, S. Cotin, N. Ayache, A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation, in: Proceedings Computer Animation 1999, 1999, pp. 70–81, http://dx.doi.org/10.1109/CA.1999.781200.

[3] H. Courtecuisse, J. Allard, P. Kerfriden, S. Bordas, S. Cotin, C. Duriez, Real-time simulation of contact and cutting of heterogeneous soft-tissues, Med. Image Anal. 18 (2) (2014) 394–410, http://dx.doi.org/10.1016/j.media.2013.11.001.

[4] J. Wu, R. Westermann, C. Dick, A survey of physically based simulation of cuts in deformable bodies, Comput. Graph. Forum 34 (2015) http://dx.doi.org/10.1111/cgf.12528.

[5] H.P. Bui, S. Tomar, H. Courtecuisse, S. Cotin, S.P.A. Bordas, Real-time error control for surgical simulation, IEEE Trans. Biomed. Eng. 65 (3) (2018) 596–607, http://dx.doi.org/10.1109/TBME.2017.2695587.

[6] O. Zienkiewicz, R. Taylor, The Finite Element Method, ; Volume 2: Solid and Fluid Mechanics, Dynamics and Non-Linearity, McGraw-Hill, 1991.

[7] C. Farhat, F. Roux, A method of finite element tearing and interconnecting and its parallel solution algorithm, Internat. J. Numer. Methods Engrg. 32 (6) (1991) 1205–1227, http://dx.doi.org/10.1002/nme.1620320604.

[8] D. Marinkovic, M. Zehn, Survey of finite element method-based real-time simulations, Appl. Sci. 9 (14) (2019) http://dx.doi.org/10.3390/app9142775.

[9] P. Kerfriden, P. Gosselet, S. Adhikari, S. Bordas, Bridging proper orthogonal decomposition methods and augmented Newton–Krylov algorithms: An adaptive model order reduction for highly nonlinear mechanical problems, Comput. Methods Appl. Mech. Engrg. 200 (5) (2011) 850–866, http://dx.doi.org/10.1016/j.cma.2010.10.009.

[10] O. Goury, C. Duriez, Fast, generic and reliable control and simulation of soft robots using model order reduction, IEEE Trans. Robot. 34 (6) (2018) 1565–1576, http://dx.doi.org/10.1109/TRO.2018.2861900.

[11] P. Kerfriden, O. Goury, T. Rabczuk, S. Bordas, A partitioned model order reduction approach to rationalise computational expenses in multiscale fracture mechanics, Comput. Methods Appl. Mech. Eng. (2012) 169–188, http://dx.doi.org/10.1016/j.cma.2012.12.004.

[12] S. Bhattacharjee, K. Matouš, A nonlinear manifold-based reduced order model for multiscale analysis of heterogeneous hyperelastic materials, J. Comput. Phys. 313 (2016) 635–653, http://dx.doi.org/10.1016/j.jcp.2016.01.040.

[13] S. Niroomandi, I. Alfaro, E. Cueto, F. Chinesta, Model order reduction for hyperelastic materials, Internat. J. Numer. Methods Engrg. 81 (2009) 1180–1206, http://dx.doi.org/10.1002/nme.2733.

[14] P. Allier, L. Chamoin, P. Ladevèze, Proper generalized decomposition computational methods on a benchmark problem: introducing a new strategy based on constitutive relation error minimization, Adv. Model. Simul. Eng. Sci. 2 (2015) 17, http://dx.doi.org/10.1186/s40323-015-0038-4.

[15] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, Cambridge, MA, USA, 2016, http://www.deeplearningbook.org.

[16] I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000, http://dx.doi.org/10.1109/72.712178.

[17] I. Lagaris, A. Likas, D. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (5) (2000) 1041–1049, http://dx.doi.org/10.1109/72.870037.

[18] K. McFall, J. Mahan, Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions, IEEE Trans. Neural Netw. 20 (8) (2009) 1221—1233, http://dx.doi.org/10.1109/tnn.2009.2020735.

[19] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, http://dx.doi.org/10.1016/j.jcp.2018.10.045.

[20] E. Samaniego, C. Anitescu, S. Goswami, V. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications, Comput. Methods Appl. Mech. Engrg. 362 (2020) 112790, http://dx.doi.org/10.1016/j.cma.2019.112790.

[21] D. Lorente, F. Martínez-Martínez, M. Rupérez, M. Lago, M. Martínez-Sober, P. Escandell-Montero, J. Martínez-Martínez, S. Martínez-Sanchis, A. Serrano-López, C. Monserrat, J. Martín-Guerrero, A framework for modelling the biomechanical behaviour of the human liver during breathing in real time using machine learning, Expert Syst. Appl. 71 (2017) 342–357, http://dx.doi.org/10.1016/j.eswa.2016.11.037.

[22] A. Mendizabal, P. Márquez-Neila, S. Cotin, Simulation of hyperelastic materials in real-time using deep learning, Med. Image Anal. 59 (2019) 101569, http://dx.doi.org/10.1016/j.media.2019.101569.

[23] V. Krokos, V. Bui Xuan, S.P.A. Bordas, P. Young, P. Kerfriden, A Bayesian multiscale CNN framework to predict local stress fields in structures with microscale features, Comput. Mech. 69 (2022) 733–766, http://dx.doi.org/10.1007/s00466-021-02112-3.

[24] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: Medical Image Computing and Computer-Assisted Intervention, MICCAI, in: LNCS, vol. 9351, Springer, 2015, pp. 234–241, (available on cs.CV), URL http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

[25] J. He, J. Xu, MgNet: A unified framework of multigrid and convolutional neural network, Sci. China Math. 62 (7) (2019) 1331–1354.

[26] F. Wang, A. Eljarrat, J. Müller, T. Henninen, R. Erni, C. Koch, Multi-resolution convolutional neural networks for inverse problems, Sci. Rep. 10 (2020) 5730, http://dx.doi.org/10.1038/s41598-020-62484-z.
[27] S.C. Brenner, L.R. Scott, Finite element multigrid methods, in: The Mathematical Theory of Finite Element Methods, Springer New York, New York, NY, 2008, pp. 155–173, http://dx.doi.org/10.1007/978-0-387-75934-0_7.
[28] R. McAllister, Y. Gal, A. Kendall, M. v. d. Wilk, A. Shah, R. Cipolla, A. Weller, Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 4745–4753, http://dx.doi.org/10.24963/ijcai.2017/661.
[29] T. Gawlikowski, C. Tassi, M. Ali, L. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, M. Shahzad, W. Yang, R. Bamler, X.X. Zhu, A survey of uncertainty in deep neural networks, 2021, arXiv:2107.03342.
[30] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision? in: Proceedings of the 31st International Conference on Neural Information Processing Systems, in: NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 5580–5590.
[31] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural network, in: F. Bach, D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 37, PMLR, Lille, France, 2015, pp. 1613–1622.
[32] Y. Gal, Uncertainty in deep learning, 2016.
[33] O. Duerr, B. Sick, E. Murina, Probabilistic Deep Learning: With Python, Keras and TensorFlow Probability, Manning Publications, 2020, URL https://books.google.co.in/books?id=-bYCEAAAQBAJ.
[34] P. Hauseux, J.S. Hale, S. Cotin, S. Bordas, Quantifying the uncertainty in a hyperelastic soft tissue model with stochastic parameters, Appl. Math. Model. 62 (2018) 86–102, http://dx.doi.org/10.1016/j.apm.2018.04.021.
[35] H. Rappel, L.A.A. Beex, J.S. Hale, L. Noels, S.P.A. Bordas, A tutorial on Bayesian inference to identify material parameters in solid mechanics, Arch. Comput. Methods Eng. 27 (2020) http://dx.doi.org/10.1007/s11831-018-09311-x.
[36] M. Zeraatpisheh, S.P. Bordas, L.A. Beex, BayesIan model uncertainty quantification for hyperelastic soft tissue models, Data-Centric Eng. 2 (2021) e9, http://dx.doi.org/10.1017/dce.2021.9.
[37] A. Graves, Practical variational inference for neural networks, in: J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems. Vol. 24, Curran Associates, Inc. 2011, URL https://proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf.
[38] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229, http://dx.doi.org/10.1038/s42256-021-00302-5.
[39] Z. Li, N.B. Kovachki, K. Azizzadenesheli, B. liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, in: International Conference on Learning Representations, 2021, URL https://openreview.net/forum?id=c8P9NQVtmnO.
[40] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, S.M. Benson, U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow, Adv. Water Resour. 163 (2022) 104180, http://dx.doi.org/10.1016/j.advwatres.2022.104180.
[41] R. Neal, Bayesian Learning for Neural Networks, Springer-Verlag, Berlin, Heidelberg, 1996.
[42] M. Vladimirova, J. Verbeek, P. Mesejo, J. Arbel, Understanding priors in Bayesian neural networks at the unit level, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 6458–6467, URL http://proceedings.mlr.press/v97/vladimirova19a.html.
[43] D.P. Kingma, T. Salimans, M. Welling, Variational dropout and the local reparameterization trick, in: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 28, Curran Associates, Inc. 2015, URL https://proceedings.neurips.cc/paper/2015/file/bc7316929fe1545bf0b98d114ee3ecb8-Paper.pdf.
[44] B.P. Carlin, T.A. Louis, Bayes and empirical bayes methods for data analysis, Stat. Comput. (1997) http://dx.doi.org/10.1023/A:1018577817064.
[45] J. Korelc, Multi-language and multi-environment generation of nonlinear finite element codes, Eng. Comput. 18 (2002) 312–327, http://dx.doi.org/10.1007/s003660200028.
[46] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, D. Cohen-Or, MeshCNN: A Network with an Edge, Vol. 38, (4) Association for Computing Machinery, New York, NY, USA, 2019, http://dx.doi.org/10.1145/3306346.3322959.
[47] T. Pfaff, M. Fortunato, A. Gonzalez, P. Battaglia, Learning mesh-based simulation with graph networks, in: International Conference on Learning Representations, 2021, URL https://openreview.net/forum?id=roNqYL0_XP.
[48] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.
[49] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2017, arXiv:1412.6980.
[50] Y. Wen, P. Vicol, J. Ba, D. Tran, R. Grosse, Flipout: Efficient pseudo-independent weight perturbations on mini-batches, 2018, arXiv:1803.04386.
[51] F. Chollet, et al., Keras, 2015, GitHub, https://github.com/fchollet/keras.
[52] J.V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M.D. Hoffman, R.A. Saurous, TensorFlow distributions, 2017, CoRR, arXiv:1711.10604.
[53] S. Varrette, P. Bouvry, H. Cartiaux, F. Georgatos, Management of an academic HPC cluster: The UL experience, in: Proceedings of the 2014 International Conference on High Performance Computing and Simulation, HPCS 2014, 2014, http://dx.doi.org/10.1109/HPCSim.2014.6903792.
[54] C. Geuzaine, J. Remacle, Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, Internat. J. Numer. Methods Engrg. 79 (2009) 1309–1331, http://dx.doi.org/10.1002/nme.2579.