



# Non-intrusive reduced-order modeling using uncertainty-aware Deep Neural Networks and Proper Orthogonal Decomposition: Application to flood modeling

Pierre Jacquier, Azzedine Abdedou, Vincent Delmas, Azzeddine Soulaïmani \*

École de Technologie Supérieure, Montreal, QC, Canada

## ARTICLE INFO

### Article history:

Received 4 June 2020

Received in revised form 15 September 2020

Accepted 15 September 2020

Available online 21 September 2020

### Keywords:

Uncertainty quantification

Deep learning

Space-time POD

Flood modeling

## ABSTRACT

Deep Learning research is advancing at a fantastic rate, and there is much to gain from transferring this knowledge to older fields like Computational Fluid Dynamics in practical engineering contexts. This work compares state-of-the-art methods that address uncertainty quantification in Deep Neural Networks, pushing forward the reduced-order modeling approach of Proper Orthogonal Decomposition-Neural Networks (POD-NN) with Deep Ensembles and Variational Inference-based Bayesian Neural Networks on two-dimensional problems in space. These are first tested on benchmark problems, and then applied to a real-life application: flooding predictions in the Mille Îles river in the Montreal, Quebec, Canada metropolitan area. Our setup involves a set of input parameters, with a potentially noisy distribution, and accumulates the simulation data resulting from these parameters. The goal is to build a non-intrusive surrogate model that is able to know when it does not know, which is still an open research area in Neural Networks (and in AI in general). With the help of this model, probabilistic flooding maps are generated, aware of the model uncertainty. These insights on the unknown are also utilized for an uncertainty propagation task, allowing for flooded area predictions that are broader and safer than those made with a regular uncertainty-uninformed surrogate model. Our study of the time-dependent and highly nonlinear case of a dam break is also presented. Both the ensembles and the Bayesian approach lead to reliable results for multiple smooth physical solutions, providing the correct warning when going out-of-distribution. However, the former, referred to as POD-EnsNN, proved much easier to implement and showed greater flexibility than the latter in the case of discontinuities, where standard algorithms may oscillate or fail to converge.

Crown Copyright © 2020 Published by Elsevier Inc. All rights reserved.

## 1. Introduction

Machine Learning and other forms of Artificial Intelligence (AI) have been at the epicenter of massive breakthroughs in the notoriously difficult fields of computer vision, language modeling and content generation, as presented in [1], [2], and [3]. Still, there are many other domains where robust and well-tested methods could be significantly improved by the modern computational tools associated with AI: antibiotic discovery is just one very recent example [4]. In the realm of

\* Corresponding author.

E-mail address: azzeddine.soulaïmani@etsmtl.ca (A. Soulaïmani).

high-fidelity computational mechanics, simulation time is tightly linked to the size of the mesh and the number of time-steps; in other words, to its accuracy, which could make it impractical to be used in real-time contexts for new parameters.

Much research has been performed to address this large-size problem and to create Reduced-Ordered Models (ROM) that can effectively replace their heavier counterpart for tasks like design and optimization, or for real-time predictions. The most common way to build a ROM is to go through a compression phase into a *reduced space*, defined by a set of *reduced basis* (RB) vectors, which is at the root of many methods, according to [5]. For the most part, RB methods involve an *offline-online* paradigm, where the former is more computationally-heavy, and the latter should be fast enough to allow for real-time predictions. The idea is to collect data points called *snapshots* from simulations, or any high-fidelity source, and extract the information that has the most significance on the dynamics of the system, the *modes*, via a reduction method in the *offline* stage.

Proper Orthogonal Decomposition (POD), as introduced in [6,7], coupled with the Singular Value Decomposition (SVD) algorithm [8], is by far the most popular method to reach a *low-rank* approximation. Subsequently, the *online* stage involves recovering the *expansion coefficients*, projecting back into the uncompressed, real-life space. This recovery is where the separation between intrusive and non-intrusive methods appears, where the former use techniques based on the problem's formulation, such as the Galerkin procedure [9–11]. At the same time, the latter (non-intrusive methods) try to statistically infer the mapping by considering the snapshots as a dataset. In this non-intrusive context, the POD-NN framework proposed by [12] and extended for time-dependent problems in [13] aims at training an artificial Neural Network to perform the mapping. These time-dependent problems can also benefit from approaching the POD on a temporal subdomain level, which has proved useful to prevent long-term error propagation, as first detailed in [14] and later assessed in [11].

Conventionally, laws of physics are expressed as well-defined Partial Differential Equations (PDEs), with boundary/initial conditions as constraints. Still, lately, pure data-driven methods have led to new approaches in PDE discovery [15]. The explosive growth of this new field of Deep Learning in Computational Fluid Dynamics was predicted in [16]. Its flexibility allows for multiple applications, such as the recovery of missing CFD data [17], or aerodynamic design optimization [18]. The cost associated with a fine mesh is high, but this has been overcome with a Machine Learning (ML) approach aimed at assessing errors and correcting quantities in a coarser setting [19]. New research in the field of numerical schemes was performed in [20], presenting the Volume of Fluid-Machine Learning (VOF-ML) approach applied in bi-material settings. A review of the vast landscape of possibilities is offered in [21]. The constraints of *small data* also led researchers to try to balance the need for data in AI contexts with expert knowledge, as with governing equations. First presented in [22], this was then extended to neural networks in [23] with applications in Computational Fluid Dynamics, as well as in vibration analysis [24]. When modeling data organized in sequence, Recurrent Neural Networks [25] are often predominant, especially the Long Short Term Memory (LSTM) variant [26]. LSTM neural networks have recently been applied in the context of time-dependent flooding prediction in [27], with the promise of providing real-time results. A recent contribution by [28] even allows for an embedded Bayesian treatment. Finally, an older but thorough study of available Machine Learning methods applied to environmental sciences and hydrology is presented in [29].

While their regression power is impressive, Deep Neural Networks are still, in their standard state, only able to predict a mean value, and do not provide any guidance on how much trust one can put into that value. To address this, recent additions to the Machine Learning landscape include Deep Ensembles [30] which suggest the training of an ensemble of specific, variance-informed deep neural networks, to obtain a complete uncertainty treatment. That work was subsequently extended to sub-ensembles for faster implementation in [31] and later reviewed in [32]. Earlier, other works had successfully encompassed the Bayesian view of probabilities within a Deep Neural Network, with the work of [33], [34], [35], [36] ultimately leading to the backpropagation-compatible Bayesian Neural Networks defined in [37], making use of Variational Inference [38], and paving the way for trainable Bayesian Neural Networks, also reviewed in [32]. Notable applications are surrogate modeling for inverse problems [39], and Bayesian physics-informed neural networks [40].

In this work, we aim at transferring the recent breakthroughs in Deep Learning to Computational Fluid Dynamics, by extending the concept of POD-NN with state-of-the-art methods for uncertainty quantification in Deep Neural Networks. After setting up the POD approach in Section 2, the methodologies of Deep Ensembles and Variational Inference for Bayesian Neural Networks are presented in Sections 3 and 4, respectively. Their performances are assessed according to a benchmark in Section 5. Our context of interest, flood modeling, is addressed in Section 6. A dam break scenario is presented in Section 6.2, first in a 1D Riemann analytically tractable example in order to obtain a reproducible problem in this context and to validate the numerical solver used in higher-dimension problems. The primary engineering aim is the training of a model capable of producing probabilistic flooding maps of the river presented in Section 6.3.1, with its results reported in Section 6.3.2. A contribution to standard uncertainty propagation is offered in 6.3.3, while Section 6.4 uses the same river environment for a fictitious dam break simulation. The Mille Îles river located in the Greater Montreal area is considered for these real-life application examples. We summarize our conclusions on this successful application of Deep Ensembles and Variational Inference for Bayesian Neural Networks in Section 7, along with our recommendations for the most promising future work in this area.

## 2. Reduced basis generation with Proper Orthogonal Decomposition

### 2.1. Objective and setup

We start by defining  $u$ , our  $\mathbb{R}^D$ -valued function of interest

$$u : \mathbb{R}^{n+P} \rightarrow \mathbb{R}^D \quad (1)$$

$$(\mathbf{x}, \mathbf{s}) \mapsto u(\mathbf{x}, \mathbf{s}),$$

with  $\mathbf{x} \in \mathbb{R}^n$  as the spatial parameters and  $\mathbf{s} \in \mathbb{R}^P$  as the additional non-spatial parameters, for anything from a fluid viscosity to the time variable.

Computing this function is costly, so only a finite number  $S$  of solutions called *snapshots* can be realized. These are obtained over a discretized space, which can either be a uniform grid or an unstructured mesh, with  $n$  representing the number of dimensions and  $D$  the number of components of the vector-valued function  $u$ .  $N_s$  is the number of non-spatial parameters sampled, and  $N_t$  counts the considered time-steps, which would be higher than one in a time-dependent setting, leading the total number of snapshots to be  $S = N_s N_t$ .

In our applications, the spatial mesh of  $N_D$  nodes is considered fixed in time, and since it is known and defined upfront, it can be incorporated in (1), removing  $\mathbf{x}$  as a parameter in  $u$ , and making  $H = N_D \times D$  be the total number of degrees of freedom (DOFs) of the mesh

$$u_D : \mathbb{R}^P \rightarrow \mathbb{R}^H \quad (2)$$

$$\mathbf{s} \mapsto u_D(\mathbf{s}).$$

The simulation data, obtained from computing the function  $u$  with  $S$  parameter sets  $\mathbf{s}^{(i)}$ , is stored in a matrix of snapshots  $\mathbf{U} = [u_D(\mathbf{s}^{(1)}) | \dots | u_D(\mathbf{s}^{(S)})] \in \mathbb{R}^{H \times S}$ . Proper Orthogonal Decomposition (POD) is used to build a Reduced-Order Model (ROM) and produce a *low-rank approximation*, which will be much more efficient to compute and use when rapid multi-query simulations are required. With the snapshots method [7], a reduced POD basis can be efficiently extracted in a finite-dimension context. In our case, we begin with the  $\mathbf{U}$  matrix, and use the Singular Value Decomposition algorithm [8] to extract  $\mathbf{W} \in \mathbb{R}^{H \times H}$ ,  $\mathbf{Z} \in \mathbb{R}^{S \times S}$  and the  $r$  descending-ordered positive singular values matrix  $\mathbf{D} = \text{diag}(\xi_1, \xi_2, \dots, \xi_r)$  such that

$$\mathbf{U} = \mathbf{W} \begin{bmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{Z}^\top. \quad (3)$$

For the finite truncation of the first  $L$  modes, the following criterion on the singular values is imposed, with a hyperparameter  $\epsilon$  given as

$$\frac{\sum_{l=L+1}^r \xi_l^2}{\sum_{l=1}^r \xi_l^2} \leq \epsilon, \quad (4)$$

and then each mode vector  $\mathbf{V}_j \in \mathbb{R}^S$  can be found from  $\mathbf{U}$  and the  $j$ -th column of  $\mathbf{Z}$ ,  $\mathbf{Z}_j$ , with

$$\mathbf{V}_j = \frac{1}{\xi_j} \mathbf{U} \mathbf{Z}_j, \quad (5)$$

so that we can finally construct our POD mode matrix

$$\mathbf{V} = [\mathbf{V}_1 | \dots | \mathbf{V}_j | \dots | \mathbf{V}_L] \in \mathbb{R}^{H \times L}. \quad (6)$$

### 2.2. Projections

Projecting to and from the low-rank approximation requires some projection coefficients; those *corresponding* to the matrix of snapshots are obtained by the following

$$\mathbf{v} = \mathbf{V}^\top \mathbf{U}, \quad (7)$$

and then  $\mathbf{U}_{\text{POD}}$ , the approximation of  $\mathbf{U}$ , can be projected back to the expanded space:

$$\mathbf{U}_{\text{POD}} = \mathbf{V} \mathbf{V}^\top \mathbf{U} = \mathbf{V} \mathbf{v}. \quad (8)$$

The following relative projection error can be computed to assess the quality of the compression/expansion procedure,

$$RE_{\text{POD}} = \sum_{j=1}^S \frac{\|(\mathbf{U})_j - (\mathbf{U}_{\text{POD}})_j\|_2}{\|(\mathbf{U})_j\|_2}, \quad (9)$$

with the  $j$  subscript denoting the  $j$ -th column of the targeted matrix, and  $\|\cdot\|_2$  the  $L^2$ -norm.

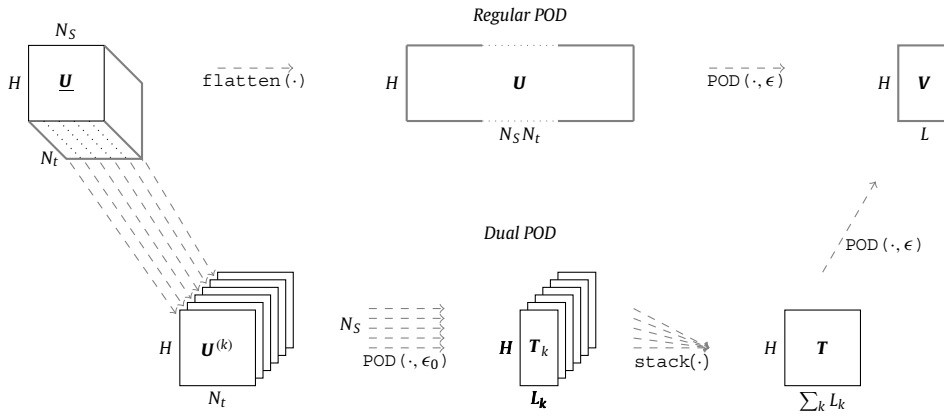


Fig. 1. Representation of the two methods for POD order reduction in time-dependent problems.

### 2.3. Improving POD speed for time-dependent problems

While the SVD algorithm is well-known and widely used, it can quickly get overwhelmed by the dimensionality of the problem, especially in a time-dependent context, such as Burgers' equation and its variations (Euler, Shallow Water, etc.), which will be discussed later in Section 6.2. Indeed, since time is being added as an input parameter, the matrix of snapshots  $\mathbf{U} \in \mathbb{R}^{H \times S}$  can have a considerable width, making it very difficult and time-consuming to manipulate. One way to deal with this is the two-step POD algorithm introduced in [13].

Instead of invoking the algorithm directly on the wide matrix  $\mathbf{U}$ , the idea is to perform the SVD first along the time axis for each parameter, as POD is usually used for standard space-time problems for a single parameter. We therefore consider the structured tensor  $\underline{\mathbf{U}} \in \mathbb{R}^{H \times N_s \times N_t}$  as a starting point.

The workflow is as follows:

1. The “time-trajectory of each parameter value,” quoting directly from [13], is being fed to the SVD algorithm, and the subsequent process of reconstructing a POD basis  $\mathbf{T}_k$  is performed for each time-trajectory  $\mathbf{U}^{(k)}$ , with  $k \in [1, N_s]$ . A specific stopping hyperparameter,  $\epsilon_0$ , is used here.
2. Each basis  $\mathbf{T}_k$  is collected in a new time-compressed matrix  $\mathbf{T}$ , in which the SVD algorithm is performed, along with the regular hyperparameter  $\epsilon$ , so the final POD basis construction to produce  $\mathbf{V}$  can be achieved.

Fig. 1 offers a visual representation of this process, and a pseudo-code implementation is available in Algorithm 1.

## 3. Learning expansion coefficients distributions using deep ensembles

### 3.1. Regression objective

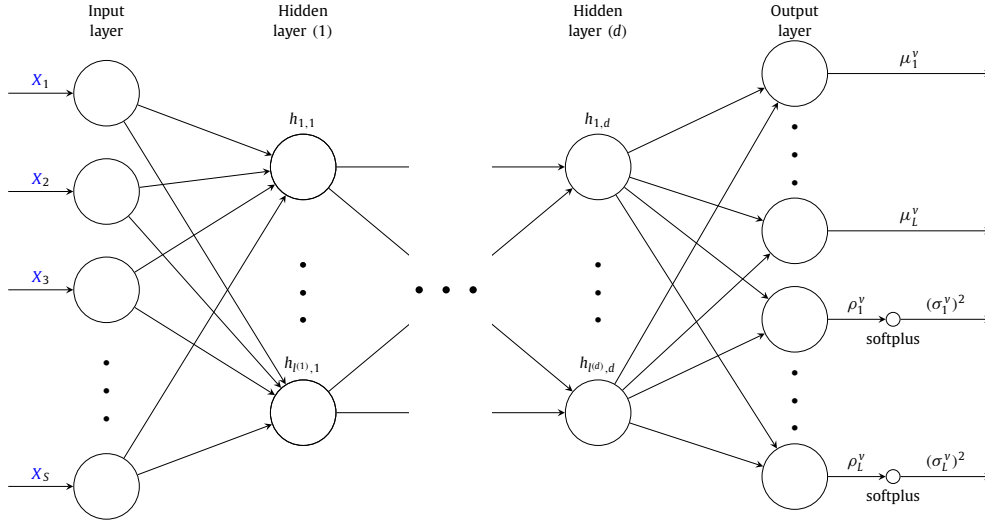
Building a non-intrusive ROM involves a statistical step to construct the function responsible for inferring the expansion parameters  $\mathbf{v}$  from new non-spatial parameters  $\mathbf{s}$ . This regression step is performed offline, and as we have considered the

**Algorithm 1:** Two-step POD that allows for the management of large, time-dependent datasets.

```

1 Function  $\text{POD}(\mathbf{U}, \epsilon)$ :
2    $\mathbf{D}, \mathbf{Z} \leftarrow \text{SVD}(\mathbf{U})$ 
3    $\mathbf{\Lambda} \leftarrow \mathbf{D}^2$ 
4    $\mathbf{\Lambda}_{\text{trunc}} \leftarrow \mathbf{\Lambda} \left[ \frac{\sum_{i=0}^L \Lambda_i}{\sum_i \Lambda_i} \geq (1 - \epsilon) \right]$ 
5    $\mathbf{V} \leftarrow \mathbf{U} \cdot \mathbf{Z} \cdot \mathbf{\Lambda}_{\text{trunc}}^{-1/2}$ 
6 return  $\mathbf{V}$ 
7
8 Function  $\text{DualPOD}(\underline{\mathbf{U}}, \epsilon, \epsilon_0)$ :
9    $\mathbf{T} \leftarrow \mathbf{0}$ 
10  for  $k$  in  $N_s$  do
11     $\mathbf{T}_k \leftarrow \text{POD}(\mathbf{U}^{(k)}, \epsilon_0)$ 
12  end
13   $\mathbf{V} \leftarrow \text{POD}(\mathbf{T}, \epsilon)$ 
14 return  $\mathbf{V}$ 

```



**Fig. 2.**  $\hat{u}_{DB}(\mathbf{X}; \mathbf{w}, \mathbf{b}) \sim \mathcal{N}(\boldsymbol{\mu}^v(\mathbf{X}), \boldsymbol{\sigma}^v(\mathbf{X})^2)$ , a Deep Neural Network regression with a dual mean and variance output.

spatial parameters  $\mathbf{x}$  to be externally handled, it can be represented as a mapping  $u_{DB}$  outputting the projection coefficients  $\mathbf{v}(\mathbf{s})$ , as in

$$\begin{aligned} u_{DB} : \mathbb{R}^P &\rightarrow \mathbb{R}^L \\ \mathbf{s} &\mapsto \mathbf{v}(\mathbf{s}). \end{aligned} \quad (10)$$

### 3.2. Deep Neural Networks with built-in variance

This statistical step is handled in the POD-NN framework by inferring the mapping with a Deep Neural Network,  $\hat{u}_{DB}(\mathbf{s}; \mathbf{w}, \mathbf{b})$ . The *weights* and *biases* of the network,  $\mathbf{w}$  and  $\mathbf{b}$ , respectively, represent the model parameters and are learned during training (*offline* phase), to be later reused to make predictions (*online* phase). The network's number of hidden layers is called the *depth*,  $d$ , which is chosen without accounting for the input and output layers. Each layer has a specific number of neurons that constitutes its *width*,  $l^{(j)}$ .

The main difference here with an ordinary DNN architecture for regression resides in the dual output, first presented in [41] and reused in [30], where the final layer size is twice the number of expansion coefficients to project,  $l^{(d+1)} = 2L$ , since it outputs both a *mean* value  $\boldsymbol{\mu}^v$  and a *raw variance*  $\boldsymbol{\rho}^v$ , which will then be constrained for positiveness through a softplus function, finally outputting  $\boldsymbol{\sigma}^{v^2}$  as

$$\boldsymbol{\sigma}^{v^2} = \text{softplus}(\boldsymbol{\rho}^v) := \log(1 + \exp(\boldsymbol{\rho}^v)). \quad (11)$$

A representation of this DNN is pictured in Fig. 2, with  $d$  hidden layers, and therefore,  $d+2$  layers in total. Each hidden layer state  $\mathbf{h}^{(j)}$  gets computed from its input  $\mathbf{h}^{(j-1)}$  alongside the layer's weights  $\mathbf{w}^{(j)}$  and biases  $\mathbf{b}^{(j)}$ , and finally goes through an activation function  $\phi$

$$\mathbf{h}^{(j)} = \phi(\mathbf{w}^{(j)}\mathbf{h}^{(j-1)} + \mathbf{b}^{(j)}), \quad (12)$$

with  $\mathbf{h}^{(0)} = \mathbf{s}$ , the input of  $\hat{u}_{DB}$ , and  $\mathbf{h}^{(d+1)} = [\boldsymbol{\mu}^v, \boldsymbol{\rho}^v]^\top$ , the output of  $\hat{u}_{DB}$ .

Since this predicted variance reports the spread, or noise, in data (the inputs' data are drawn from a distribution), and so it would not be reduced even if we were to grow our dataset larger, it accounts for the *aleatoric uncertainty*, which is usually separated from *epistemic uncertainty*. This latter form is inherent to the model [42].

One can think about this concept of aleatoric uncertainty as a measurement problem with the goal of measuring a quantity  $u$ . The tool used for measurement has some inherent noise  $n$ , random and dependent upon the parameter  $x$  in the measurable domain, making the measured quantity  $u(x) + n(x)$ . The model presented here, as introduced in [41], is designed to perform the regression on both components, with an estimated variance alongside the regular point-estimate of the mean.

### 3.3. Ensemble training

An  $N$ -sized training dataset  $\mathcal{D} = \{\mathbf{X}_i, \mathbf{v}_i\}$  is considered, with  $\mathbf{X}_i$  denoting the normalized non-spatial parameters  $\mathbf{s}$ , and  $\mathbf{v}_i$  the corresponding expansion coefficients from a training/validation-split of the matrix of snapshots  $\mathbf{U}$ . An *optimizer*

**Algorithm 2:** Deep Ensembles training and predictions.

```

1 Prepare the dataset  $\mathcal{D} = \{\mathbf{X}_i, \mathbf{v}_i\}$ 
2 for each model in the ensemble  $1 \leq m \leq M$  do
3   Train the model  $m$ :
4   for each epoch  $1 \leq e \leq N_e$  do
5     Retrieve the outputs  $(\mu_{\theta_m}^v(\mathbf{X}), \sigma_{\theta_m}^v(\mathbf{X}))$  from the forward pass  $\hat{u}_D(\mathbf{X})$ 
6     Perform the variance treatment,  $\sigma_{\theta_m}^v(\mathbf{X})^2 = \text{softplus}(\rho_{\theta_m}^v(\mathbf{X}))$ 
7     Compute the loss  $\mathcal{L}_{\text{NLL}}$ 
8     Backpropagate the gradients to the parameters  $\theta_m$ 
9   end
10 end
11 for each model in the ensemble  $1 \leq m \leq M$  do
12   Retrieve statistical outputs  $(\mu_{\theta_m}^v(\mathbf{X}_{\text{tst},i}), \sigma_{\theta_m}^v(\mathbf{X}_{\text{tst},i})^2)$  for the model  $m$  for a test dataset  $\mathcal{D}_{\text{tst}} = \{\mathbf{X}_{\text{tst},i}, \mathbf{v}_{\text{tst},i}\}$ 
13 end
14 Approximate the predictions for the reduced space in a Gaussian  $\mathcal{N}(\mu_*^v(\mathbf{X}_{\text{tst},i}), \sigma_*^v(\mathbf{X}_{\text{tst},i})^2)$ 

```

performs several *training epochs*  $N_e$  to minimize the following Negative Log-Likelihood loss function with respect to the network weights and biases parametrized by  $\theta = (\mathbf{w}, \mathbf{b})$

$$\mathcal{L}_{\text{NLL}}(\mathcal{D}, \theta) := \frac{1}{N} \sum_{i=1}^N \left[ \frac{\log \sigma_{\theta}^v(\mathbf{X}_i)^2}{2} + \frac{(\mathbf{v}_i - \mu_{\theta}^v(\mathbf{X}_i))^2}{2\sigma_{\theta}^v(\mathbf{X}_i)^2} \right], \quad (13)$$

with  $\mu_{\theta}^v(\mathbf{X})$  and  $\sigma_{\theta}^v(\mathbf{X})^2$  as the mean and variance, respectively, retrieved from the  $\theta$ -parametrized network.

In practice, this loss gets an L2 regularization as an additional term, commonly known as *weight decay* in Neural Network contexts [43], producing

$$\mathcal{L}_{\text{NLL}}^{\lambda}(\mathcal{D}, \theta) := \mathcal{L}_{\text{NLL}}(\mathcal{D}, \theta) + \lambda \|\mathbf{w}\|^2. \quad (14)$$

Non-convex optimizers, such as Adam [44] or other Stochastic Gradient Descent variants, are needed to handle this loss function, often irregular and non-convex in a Deep Learning context. The derivative of the loss  $\mathcal{L}_{\text{NLL}}$  with respect to the weights  $\mathbf{w}$  and biases  $\mathbf{b}$  is obtained through *automatic differentiation* [45], a technique that relies on monitoring the gradients during the forward pass of the network (12). Using *backpropagation* [46], the updated weights  $\mathbf{w}^{n+1}$  and biases  $\mathbf{b}^{n+1}$  corresponding to the epoch  $n+1$  can be written as

$$(\mathbf{w}^{n+1}, \mathbf{b}^{n+1}) = (\mathbf{w}^n, \mathbf{b}^n) - \tau f \left( \frac{\partial \mathcal{L}_{\text{NLL}}^{\lambda}(\mathcal{D}, (\mathbf{w}^n, \mathbf{b}^n))}{\partial (\mathbf{w}^n, \mathbf{b}^n)} \right), \quad (15)$$

where  $f(\cdot)$  is a function of the loss derivative with respect to weights and biases that is dependent upon the chosen optimizer, and  $\tau$  is the *learning rate*, a hyperparameter defining the step size taken by the optimizer.

The idea behind Deep Ensembles, presented in [30] and recommended in [32], is to randomly initialize  $M$  sets of  $\theta_m = (\mathbf{w}, \mathbf{b})$ , thereby creating  $M$  independent neural networks (NNs). Each NN is then subsequently trained. Overall, the predictions moments in the reduced space  $(\mu_{\theta_m}^v, \sigma_{\theta_m}^v)$  of each NN create a probability mixture, which, as suggested by the original authors, we can approximate in a single Gaussian distribution, leading to a mean expressed as

$$\mu_*^v(\mathbf{X}) = \frac{1}{M} \sum_{m=1}^M \mu_{\theta_m}^v(\mathbf{X}), \quad (16)$$

and a variance subsequently obtained as

$$\sigma_*^v(\mathbf{X})^2 = \frac{1}{M} \sum_{m=1}^M \left[ \sigma_{\theta_m}^v(\mathbf{X})^2 + \mu_{\theta_m}^v(\mathbf{X})^2 \right] - \mu_*^v(\mathbf{X})^2. \quad (17)$$

The model is now accounting for the *epistemic uncertainty* through random initialization and variability in the training step. This uncertainty is directly linked to the model and could be reduced if we had more data. The uncertainty is directly related to the data-fitting capabilities of the model and thus will snowball in the absence of such data since there are no more constraints. In our case, it has the highest value, compared to aleatoric uncertainty, since one of our objectives is to be warned when the model is making predictions that are out-of-distribution.

This model will be referred to as POD-EnsNN, and its training steps are listed in Algorithm 2. Since these networks are independent, parallelizing their training is relatively easy (see Algorithm 3), with only the results needing to be averaged-over.

**Algorithm 3:** Pseudo-code showing parallelization with Horovod [47].

```

1 Function TrainOnOneDevice ( $\mathbf{X}, \mathbf{v}, \lambda, \tau, N_e$ ):
2   Import the TensorFlow library as tf
3   Import the Horovod library as hvd and initialize it with hvd.init()
4   Get the assigned device id  $i = \text{hvd.localRank}()$ 
5   Get local devices  $\mathbf{D} = \text{tf.getVisibleDevices}()$ 
6   Force the device for TensorFlow: tf.setVisibleDevices( $\mathbf{D}_i$ )
7   Init the model:  $\hat{u}_{DB} = \text{NeuralNetwork}(\tau, \lambda)$ 
8   Train it:  $\hat{u}_{DB}.fit(\mathbf{X}, \mathbf{v}, N_e)$ 
9 return  $\hat{u}_{DB}$ 
10 Run the meta-command: horovodrun -np M -H localhost:M TrainOnOneDevice( $\mathbf{X}, \mathbf{v}, \lambda, \tau, N_e$ )

```

**3.4. Predictions in the expanded space**

While embedding uncertainty quantification within Deep Neural Networks helps to obtain a confidence interval on the predicted expansion coefficients  $\mathbf{v}$ , it is still necessary to then perform the expansion step to retrieve the full solution, as presented in (8). It is defined as a dot product with the modes matrix  $\mathbf{V}$ .

While this applies perfectly to the predicted mean  $\mu^v$ , care must be taken when handling the predicted standard deviation  $\sigma^v$ , as there is no theoretical guarantee for the statistical moments on the reduced basis to translate linearly in the expanded space. However, after the mixture approximation, the distribution over the coefficients  $\mathbf{v}$  is known as follows:

$$\hat{\mathbf{v}}(\mathbf{X}) = \hat{u}_{DB}(\mathbf{X}; \mathbf{w}, \mathbf{b}) \sim \mathcal{N}(\mu_*^v(\mathbf{X}), \sigma_*^{v^2}(\mathbf{X})). \quad (18)$$

Therefore, unlimited samples  $\hat{\mathbf{v}}^{(i)}$  can be drawn from this distribution, and individually decompressed into a corresponding full solution  $\hat{u}_D^{(i)} = \mathbf{V} \cdot \hat{\mathbf{v}}^{(i)}$ , from (8). The following Monte-Carlo approximation of the full distribution on  $\hat{u}_D$  is hence proposed, drawing  $N_{\text{ex}}$  samples and using the rapid surrogate model to compute

$$\mu_*(\mathbf{X}) = \frac{1}{N_{\text{ex}}} \sum_{i=1}^{N_{\text{ex}}} \hat{u}_D^{(i)} = \frac{1}{N_{\text{ex}}} \sum_{i=1}^{N_{\text{ex}}} \mathbf{V} \cdot \mathbf{v}^{(i)}, \quad (19)$$

$$\sigma_*^2(\mathbf{X}) = \frac{1}{N_{\text{ex}}} \sum_{i=1}^{N_{\text{ex}}} [\hat{u}_D^{(i)} - \mu_*]^2 = \frac{1}{N_{\text{ex}}} \sum_{i=1}^{N_{\text{ex}}} [\mathbf{V} \cdot \mathbf{v}^{(i)} - \mu_*]^2, \quad (20)$$

which represents the approximated statistical moments of the distribution on the predicted full solution  $\hat{u}_D(\mathbf{X})$ , also referred to as  $\hat{u}_D^\mu$  and  $\hat{u}_D^\sigma$ .

**3.5. Metrics**

In addition to the regularized loss  $\mathcal{L}_{\text{NLL}}^\lambda$ , we define a relative error *RE* on the mean prediction as

$$RE(\mu_*, \mathbf{U}) = \frac{\|\sum_{i=1}^S (\mu_*(\mathbf{X}_i) - \mathbf{U}_i)\|_2}{\|\sum_{i=1}^S \mathbf{U}_i\|_2}, \quad (21)$$

with  $\mathbf{U}_i$  the  $i$ -th column of the snapshots matrix, corresponding to the input  $X_i$ . It can be applied for training, validation, or testing, as defined in Section 2. During the training, we report two metrics: the training loss  $\mathcal{L}_{\text{NLL}}^\lambda$  and the validation relative error  $RE_{\text{val}}$ .

To quantify the uncertainty associated with the model predictions, we define the *mean prediction interval width* (MPIW) [48], aimed at tracking the size of the 95% confidence interval, i.e.,  $\pm 2\sigma_*$ , as follows

$$MPIW(\sigma_*) = \frac{1}{HN} \sum_{j=1}^H \sum_{i=1}^N [\hat{u}_D^{\text{upper}}(\mathbf{X}_i)_j - \hat{u}_D^{\text{lower}}(\mathbf{X}_i)_j] = \frac{1}{HN} \sum_{j=1}^H \sum_{i=1}^N 4\sigma_*^2(\mathbf{X}_i)_j, \quad (22)$$

with the  $j$  subscript denoting the  $j$ -th degree of freedom of a solution.

**3.6. Adversarial training**

First proposed in [49] and studied in [50], the concept of *adversarial training*, not to be confused with Generative Adversarial Networks [51], aims at improving the robustness of Neural Networks when confronted with noisy data, which could potentially be intentionally created.



**Algorithm 4:** Adversarial training within the training loop.

```

1 Function getAdversarialLoss ( $\mathbf{X}, \mathbf{v}, \epsilon$ ):
2    $\mathcal{L}_T \leftarrow \mathcal{L}_{\text{NLL}}(\hat{u}_D(\mathbf{X}), \mathbf{v}), \theta$ 
3    $\mathbf{X}' \leftarrow \mathbf{X} + \zeta \operatorname{sign}(\frac{\partial \mathcal{L}_T}{\partial \mathbf{X}})$ 
4    $\mathcal{L}_T \leftarrow \mathcal{L}_T + \mathcal{L}_{\text{NLL}}(\hat{u}_D(\mathbf{X}'), \mathbf{v}), \theta$ 
5 return  $\mathcal{L}_T$ 

```

In the Deep Ensembles framework, adversarial training is an optional component that, according to [30], can help to smooth out the output. This technique can be particularly useful as shown in the subsequent test case, where the model is struggling with the highly-nonlinear wave being produced by Shallow Water equations (see Section 6.2).

A simple implementation is the *gradient sign* technique, which adds noise in the opposite direction of the gradient descent, scaled by a new hyperparameter  $\zeta$ , at each training epoch, as shown in Algorithm 4. The idea is to perform *data augmentation* at each training epoch. The additional data comes from the generated adversarial samples that will help to train the network more robustly, given that these problematic samples are being inserted in the dataset.

#### 4. Bayesian Neural Networks and variational inference as an alternative

Making a model aware of its associated uncertainties can ultimately be achieved by adopting the Bayesian view. Recently, it has become easier to include a fully Bayesian treatment within Deep Neural Networks [37], designed to be compatible with backpropagation. In this section, we implement this version of Bayesian Neural Networks within the POD-NN framework, which we will refer to as POD-BNN, and compare it to the Deep Ensembles approach.

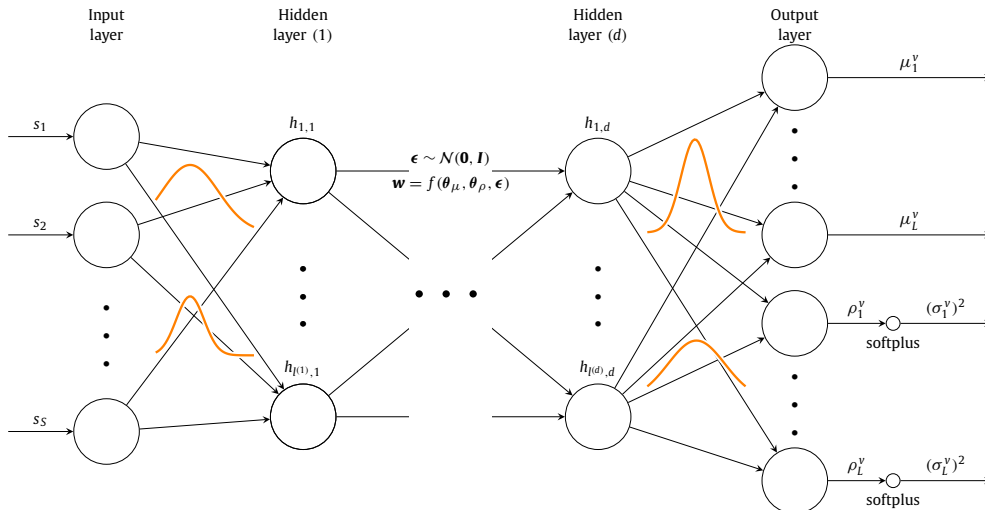
##### 4.1. Overview

To address the *aleatoric uncertainty*, arising from noise in the data, Bayesian Neural Networks can make use of the same dual-output setting as the NNs we used earlier for Deep Ensembles,  $(\mu^v, \rho^v)$  in our context, with the variance  $\sigma^{v^2}$  subsequently retrieved with the softplus function defined in (11).

However, in the *epistemic uncertainty* treatment the process and issues are very different. Earlier, even though the NNs were providing us with a mean and variance, they were still deterministic, and variability was obtained by assembling randomly initialized models. The Bayesian treatment instead aims to assign distributions to the network's weights, and they therefore have a probabilistic output by design (see Fig. 3). In this context, it is necessary to make multiple predictions, instead of numerous (parallel) trainings, in order to obtain data on uncertainties.

Considering a dataset  $\mathcal{D} = \{\mathbf{X}_i, \mathbf{v}_i\}$ , a *likelihood* function  $p(\mathcal{D}|\mathbf{w})$  can be built, with  $\mathbf{w}$  denoting both the weights  $\mathbf{w}$  and the biases  $\mathbf{b}$  for simplicity. The goal is then to construct a *posterior distribution*  $p(\mathbf{w}|\mathcal{D})$  to achieve the following *posterior predictive distribution* on the target  $\mathbf{v}$  for a new input  $\mathbf{X}$

$$p(\mathbf{v}|\mathbf{X}, \mathcal{D}) = \int p(\mathbf{v}|\mathbf{X}, \mathbf{w})p(\mathbf{w}|\mathcal{D})d\mathbf{w}, \quad (23)$$



**Fig. 3.**  $\hat{u}_{DB}(\mathbf{X}; \theta) \sim \mathcal{N}(\mu^v(\mathbf{X}), \sigma^v(\mathbf{X})^2)$ , a probabilistic Bayesian Neural Network regression with a dual mean and variance output, and distributions on the weights.



which cannot be achieved directly in a NN context, due to the infinite possibilities for the weights  $\mathbf{w}$ , leaving the posterior  $p(\mathbf{w}|\mathcal{D})$  intractable as explained in [37]. A few observations can be made on this formula. First, the initial term in the integral,  $p(\mathbf{v}|\mathbf{X}, \mathbf{w})$ , stands for the distribution of the target  $\mathbf{v}$  for the input  $\mathbf{X}$  according to a weight configuration  $\mathbf{w}$ . It directly describes the noise in the data and is handled by the NN's dual-output setting. Second, the posterior distribution  $p(\mathbf{w}|\mathcal{D})$  accounts for the distribution on the weights given the dataset  $\mathcal{D}$ , which bundles the uncertainty on the weights since they are sampled in a finite setting [29]. This decomposition shows the power of the approach. Yet the bottleneck resides in the intractability of the posterior.

While various attempts have been made at approximating this integral in a NN context, such as Markov Chains methods [52,53], the most common way is through *Variational Inference*, first presented by [38], which ultimately led to trainable BNNs in [37]. The idea is to construct a new  $\theta$ -parametrized distribution  $q(\mathbf{w}|\theta)$  as an approximation of  $p(\mathbf{w}|\mathcal{D})$ , by minimizing their Kullback-Leibler divergence, with the goal being computing (23). The KL measures the difference between two distributions and can be defined for two continuous densities  $a(x)$  and  $b(x)$  as

$$\text{KL}(a(x)||b(x)) = \int a(x) \log \frac{a(x)}{b(x)} dx, \quad (24)$$

and has the property of being non-negative. In our case, it writes as  $\text{KL}(q(\mathbf{w}|\theta)||p(\mathbf{w}|\mathcal{D}))$  with respect to the new parameters  $\theta$  called *latent variables*, such as

$$\text{KL}(q(\mathbf{w}|\theta)||p(\mathbf{w}|\mathcal{D})) = \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{p(\mathbf{w}|\mathcal{D})} d\mathbf{w} = \mathbb{E}_{q(\mathbf{w}|\theta)} \left[ \log \frac{q(\mathbf{w}|\theta)}{p(\mathbf{w}|\mathcal{D})} \right]. \quad (25)$$

Applying Bayes rule, the posterior  $p(\mathbf{w}|\mathcal{D})$  can be rewritten as  $p(\mathcal{D}|\mathbf{w})p(\mathbf{w})/p(\mathcal{D})$ , and so

$$\text{KL}(q(\mathbf{w}|\theta)||p(\mathbf{w}|\mathcal{D})) = \mathbb{E}_{q(\mathbf{w}|\theta)} \left[ \log \frac{q(\mathbf{w}|\theta)p(\mathcal{D})}{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})} \right] \quad (26)$$

$$= \mathbb{E}_{q(\mathbf{w}|\theta)} \left[ \log \frac{q(\mathbf{w}|\theta)}{p(\mathbf{w})} - \log p(\mathcal{D}|\mathbf{w}) + \log p(\mathcal{D}) \right]. \quad (27)$$

Recognizing a KL difference between the approximated distribution  $q(\mathbf{w}|\theta)$  and the prior distribution on the weights  $p(\mathbf{w})$ , and the non-dependence on the weights of the *marginal likelihood*  $p(\mathcal{D})$ :

$$\text{KL}(q(\mathbf{w}|\theta)||p(\mathbf{w}|\mathcal{D})) = \text{KL}(q(\mathbf{w}|\theta)||p(\mathbf{w})) - \mathbb{E}_{q(\mathbf{w}|\theta)} [\log p(\mathcal{D}|\mathbf{w})] + \log p(\mathcal{D}) \quad (28)$$

$$=: \mathcal{F}(\mathcal{D}, \theta) + \log p(\mathcal{D}). \quad (29)$$

The term  $\mathcal{F}(\mathcal{D}, \theta)$  is commonly known as the *variational free energy*, and minimizing it with respect to the weights does not involve the last term  $\log p(\mathcal{D})$ , and so it is equivalent to the goal of minimizing  $\text{KL}(q(\mathbf{w}|\theta), ||p(\mathbf{w}|\mathcal{D}))$ . If an appropriate choice of  $q$  is made, (29) can be computationally tractable, and the bottleneck is worked around. In any case, this term acts as a lower bound on the likelihood, tending to an exact inference case where  $\mathcal{F}(\mathcal{D}, \theta)$  would become the log-likelihood  $\log p(\mathcal{D}|\mathbf{w})$ , [54].

By drawing  $N_{\text{mc}}$  samples  $\mathbf{w}^{(i)}$  from the distribution  $q(\mathbf{w}|\theta)$  at the layer level, it is possible to construct a tractable Monte-Carlo approximation of the variational free energy, such as

$$\mathcal{F}(\mathcal{D}, \theta) \approx \sum_{i=1}^{N_{\text{mc}}} \left[ \log q(\mathbf{w}^{(i)}|\theta) - \log p(\mathbf{w}^{(i)}) - \log p(\mathcal{D}|\mathbf{w}^{(i)}) \right], \quad (30)$$

with  $p(\mathbf{w}^{(i)})$  denoting the *prior* on the drawn weight  $\mathbf{w}^{(i)}$ , which is chosen by the user, with an example given in (31). The variational free energy is a sum of two terms, the first being linked to the prior, named *complexity cost*, while the latter is related to the data and referred to in [37] as the *likelihood cost*. The latter shows to be approximated by summing on the  $N_{\text{mc}}$  samples at the output level (for each training input).

Equation (30) defines our new loss function  $\mathcal{L}_{\text{ELBO}}$ . This name comes from the *Evidence Lower Bound* function, commonly known in the literature, and corresponding to the opposite maximizing objective. The third term in (30) may be recognized as a Negative Log-Likelihood, which was used in the training of Deep Ensembles, and will be evaluated from the NN's outputs. The first two are issued from an approximation of the KL divergence at the layer level.

#### 4.2. Choice of prior distributions

The Bayesian view differs that of the frequentists with its ability to reduce the overall uncertainty by observing new data points. The initial shape is described by a prior distribution, representing the previously known information to encode in the model.

In our case, the prior distribution of the NN weights,  $p(\mathbf{w})$ . For simplicity, in this work we start by reusing the *fixed* Gaussian mixture proposed in [37], defined for three positive hyperparameters  $\pi_0$ ,  $\pi_1$ , and  $\pi_2$ , such that

$$p(\mathbf{w}) = \pi_0 \mathcal{N}(\mathbf{w}|0, \pi_1^2) + (1 - \pi_0) \mathcal{N}(\mathbf{w}|0, \pi_2^2). \quad (31)$$

**Algorithm 5:** Epoch training of a BNN via *Bayes by Backprop* [37].

```

1 Feed the model with the dataset  $\mathcal{D}$ 
2 for each variational layer  $1 \leq j \leq d$  do
3    $\epsilon^{(j)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
4    $\mathbf{w}^{(j)} = f(\theta_\mu^{(j)}, \theta_\rho^{(j)}, \epsilon^{(j)})$ 
5    $\theta_\sigma^{(j)} = \text{softplus}(\theta_\rho^{(j)})$ 
6   Sample the variational posterior  $q(\mathbf{w}^{(j)}|\theta^{(j)}) \sim \mathcal{N}(\theta_\mu^{(j)}, \theta_\sigma^{(j)})$ 
7   Sample the prior  $p(\mathbf{w}^{(j)})$ 
8   Contribute the posterior and prior values to the loss,  $\mathcal{L}_{\text{ELBO}} += \log q(\mathbf{w}^{(j)}|\theta^{(j)}) + \log p(\mathbf{w}^{(j)})$ 
9   Perform the forward pass  $\mathbf{h}^{(j)} = \phi(\mathbf{w}^{(j)}\mathbf{h}^{(j-1)} + \mathbf{b}^{(j)})$ 
10 end
11 Retrieve each outputs pair  $\mu^v, \sigma^{v^2}$  from the NN
12 Compute the likelihood from the outputs,  $p(\mathcal{D}|\mathbf{w}) \sim \mathcal{N}(\mu^v, \sigma^{v^2})$ 
13 Contribute the NLL to the loss,  $\mathcal{L}_{\text{ELBO}} += -\log p(\mathcal{D}|\mathbf{w})$ 
14 Backpropagate the gradients  $\frac{\partial \mathcal{L}_{\text{ELBO}}}{\partial \theta}$  to update the latent variables  $\theta$ 

```

### 4.3. Training

The idea behind the work of [37] was to have a fully Bayesian treatment of the weights while providing it in a form compatible to the usual *backpropagation* algorithm, mentioned in Section 3. One of the blockers is the forward pass that requires gradients to be tracked, allowing their derivatives to be backpropagated.

At the  $j$ -th *variational layer*, we consider a Gaussian distribution for the approximated distribution  $q(\mathbf{w}^{(j)}|\theta^{(j)})$ , effectively parameterizing the weights and the biases by a mean  $\theta_\mu^{(j)}$  and raw variance  $\theta_\rho^{(j)}$ , acting as local latent variables. This setting leads the total number of trainable parameters of the network to be twice that in a standard NN, as each  $\mathbf{w}^{(j)}$  is sampled from the approximated two-parameter Gaussian distribution  $q(\mathbf{w}^{(j)}|\theta^{(j)}) \sim \mathcal{N}(\theta_\mu^{(j)}, \theta_\rho^{(j)})$ .

In the forward pass, to keep track of the gradients, each operation must be differentiable. To sample the weights  $\mathbf{w}^{(j)}$ , we construct a function  $f(\theta_\mu^{(j)}, \theta_\rho^{(j)}) = \theta_\mu^{(j)} + \theta_\rho^{(j)} \odot \epsilon^{(j)} =: \mathbf{w}^{(j)}$ , with  $\epsilon^{(j)}$  sampled from a parameter-free normal distribution,  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . This is known as the *reparametrization trick* [55].

The true variance of the weights  $\theta_\sigma^{(j)}$  is not the direct parameter, but as stated earlier, to ensure positivity and numerical stability, it is defined through a softplus function, with  $\theta_\sigma^{(j)} = \log(1 + \exp(\theta_\rho^{(j)}))$ .

Going back to (30), it can be observed that the Monte-Carlo summation is actually going to be two-fold while training. Firstly, at each layer  $j$ , the same number of weights in  $\mathbf{w}^{(j)}$  as the number of neurons  $l^{(j)}$  are going to be produced, creating the summation, and enabling the approximated posterior  $q(\mathbf{w}^{(j)}|\theta^{(j)})$  and the prior  $p(\mathbf{w}^{(j)})$  distributions to be contributed in the logarithm form to the loss  $\mathcal{L}_{\text{ELBO}}$ . Secondly, a full forward pass is required to compute the Negative Log-Likelihood of the outputs  $-\log p(\mathcal{D}|\mathbf{w})$ , and contributes to the loss as well. The practical implementation steps for one training epoch are summarized in Algorithm 5.

The activation function has been chosen to be ReLU by default, as for the ensembles approach in Section 3. However, reaching convergence for some discontinuous time-dependent problems was achieved with the  $\phi: x \mapsto \tanh(x)$  activation, known to perform better in probabilistic models contexts [54].

### 4.4. Predictions

Applying Algorithm 5 for each training epoch produces an optimal value of the variational parameters, referred to as  $\theta_{\text{ELBO}}$ , which minimizes the loss function  $\mathcal{L}_{\text{ELBO}}(\mathcal{D}, \theta)$  and defines the approximated posterior,  $q(\mathbf{w}|\theta_{\text{ELBO}})$ . From this distribution, regular NN weights  $\mathbf{w}$  can be drawn, and sample predictions can be produced by evaluating the network with a forward pass as in (12), for any new input data  $\mathbf{X}$ . If new targets  $\mathbf{v}$  are now considered to be predicted, it is possible to approximate the predictive posterior distribution (23) as

$$p(\mathbf{v}|\mathbf{X}, \mathcal{D}) = \int p(\mathbf{v}|\mathbf{X}, \mathbf{w})q(\mathbf{w}|\theta_{\text{ELBO}}) d\mathbf{w} \quad (32)$$

It can be observed that considering one weights' configuration  $\mathbf{w}_b$  sampled from the inferred distribution  $q(\mathbf{w}|\theta_{\text{ELBO}})$  from the optimal latent variables  $\theta_{\text{ELBO}}$ ,  $p(\mathbf{v}|\mathbf{X}, \mathbf{w}_b) = p(\mathbf{v}|\mathbf{X}, f(\theta_{\text{ELBO}}))$  represents the network output distribution with moments  $(\mu_{\mathbf{w}_b}^v(\mathbf{X}), \sigma_{\mathbf{w}_b}^v(\mathbf{X})^2)$ . Therefore, (32) shows that the posterior predictive distribution is equivalent to averaging predictions from an ensemble of NNs, weighted by the posterior probabilities of their weights,  $\mathbf{w}_b$ . While each output distribution accounts for the variability in the data, or aleatoric uncertainty, (32) tracks the variability in the model configuration, the epistemic uncertainty, via the  $\theta$ -parametrized distribution and the integral. The mean of the predictions is hence given by

$$\mu_{\mathbf{X}} = \int \mathbf{v} p(\mathbf{v}|\mathbf{X}, \mathcal{D}) d\mathbf{X} = \iint \mathbf{v} p(\mathbf{v}|\mathbf{X}, \theta)q(\theta|\mathcal{D}) d\mathbf{X}d\theta = \int q(\theta|\mathcal{D})\mu(\theta) d\theta. \quad (33)$$

By drawing  $B$  samples  $\mathbf{w}_b$  from  $q(\mathbf{w}|\theta_{\text{ELBO}})$ , the mean of the predictions in the reduced space is approximated by

$$\mu_*^v(\mathbf{X}) = \frac{1}{B} \sum_{b=1}^B \mu_{\mathbf{w}_b}^v(\mathbf{X}). \quad (34)$$

As for the ensembles approach in Section 3, we approximate each NN variance in one distribution, with the following, which allows for a fast estimation of the mixture in a single Gaussian,

$$\sigma_*^v(\mathbf{X})^2 = \frac{1}{B} \sum_{b=1}^B [\sigma_{\mathbf{w}_b}^v(\mathbf{X})^2 + \mu_{\mathbf{w}_b}^v(\mathbf{X})^2] - \mu_*^v(\mathbf{X})^2. \quad (35)$$

Expanded space predictions  $(\mu_*(\mathbf{X}), \sigma_*(\mathbf{X})^2)$  are performed then to retrieve the full solution  $\hat{u}_D(\mathbf{s})$ , just as for the ensembles approach, with (19) and (20).

## 5. Benchmark with uncertainty quantification

In this section, we assess the uncertainty propagation component of our framework against a steady and two-dimensional benchmark, known as the Ackley Function.

### 5.1. Setup

The library TensorFlow version 2.2.0 [56] is used for all results, while the SVD algorithm and various matrix operations are performed using NumPy, all in Python 3.8. To implement variational layers, we used the new TensorFlow Probability module in version 0.10.0, which allows for greater interoperability with regular networks [57]. Its source code and the corresponding results were validated in-house against a custom adaptation of the code presented in [58]. Documented source code is available at <https://github.com/pierremtb/POD-UQNN>, on both POD-EnsNN and POD-BNN branches.

In almost all benchmarks, the *activation function* on all hidden layers is the default ReLU nonlinearity  $\phi: x \mapsto \max(0, x)$ . At the same time, a linear mapping is applied to the output layer, since in a regression case, real-valued variables are needed as outputs. We perform normalization on all non-spatial parameters  $\mathbf{s}$ , to build the inputs  $\mathbf{X}$  as

$$\mathbf{X} = \frac{\mathbf{s} - \bar{\mathbf{s}}}{\mathbf{s}_{\text{std}}}, \quad (36)$$

with  $\bar{\mathbf{s}}$  and  $\mathbf{s}_{\text{std}}$  respectively the empirical mean and standard deviation over the dataset. The two quantities are computed on each column to maintain the physical meaning, e.g., the time would be normalized against the mean time step and the standard deviation of the time steps, and not against space quantities.

To achieve GPU parallel training, we used the Horovod library [47], which allowed us to efficiently train  $M = 5$  models on  $M = 5$  GPUs at the same time. This number is recommended as a good starting point in [30].

Numba optimizations have also been used for both POD and data generation, which allows for multiple threading and native code compilation within Python, and is especially useful for loop-based computations [59].

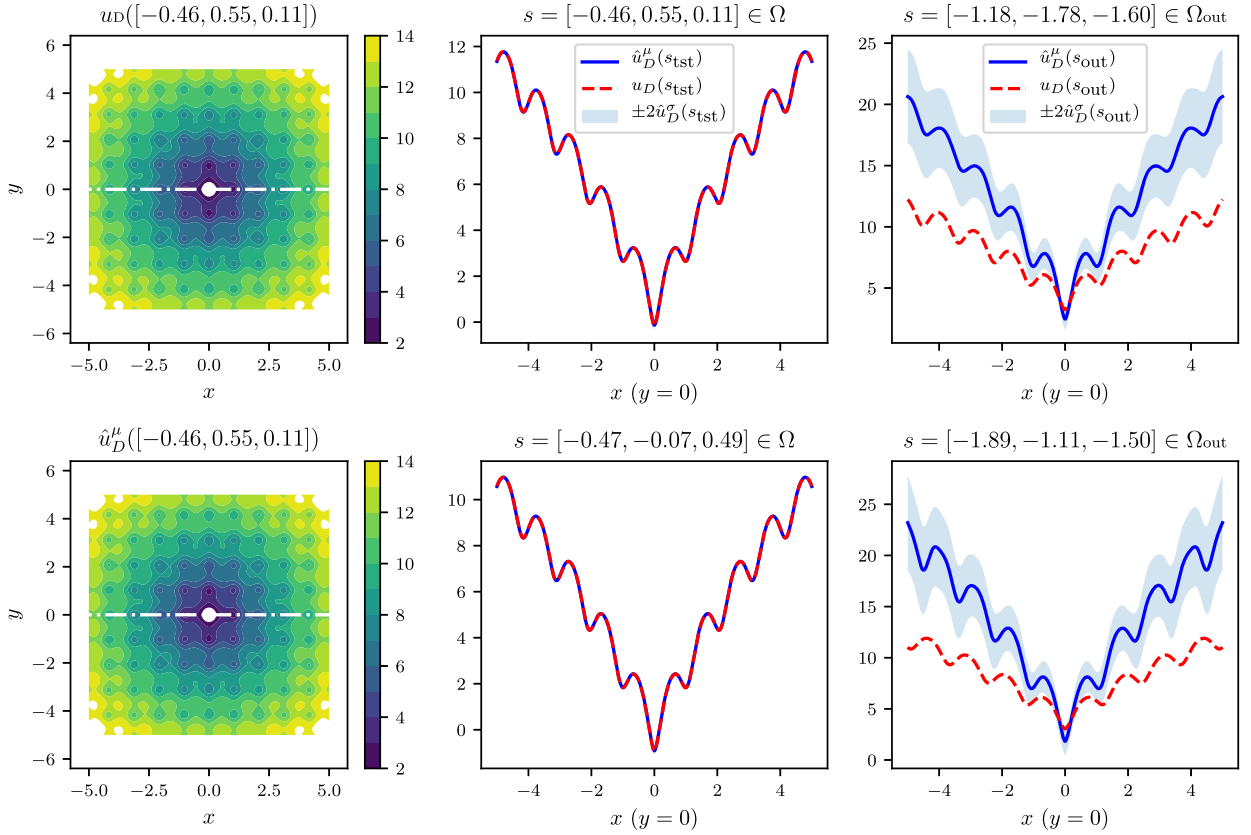
It is also important to note that in practice, a hyperparameter can be added to ensure the stability of the output variance when going through the softplus function for positivity requirements in both approaches (POD-BNN and POD-EnsNN). Denoted as  $\kappa$  with a default value of 1, this hyperparameter is involved in the softplus function calls as

$$\text{softplus}(x) = \log(1 + \exp(\kappa x)). \quad (37)$$

**Remark 1.** For the following benchmarks and the subsequent applications in Section 6, we chose a constant 20% *validation* split  $\mathcal{D}_{\text{val}}$  of the generated dataset  $\mathcal{D}$  from Equations ((3)–(7)). The relative error  $RE$  defined in (21) is computed at each training epoch for both the training set and the validation set. By keeping track of both, we try to avoid *overfitting*. A manual *early stopping* is therefore performed, in case the validation error might increase at some epoch  $N_e$  in the training. No mini-batch split is performed, as our dataset is small enough to be fully handled in memory, and no improvement for using a mini-batch split was shown in our experiments. The final results are reported as well on a testing set generated for  $N_{\text{tst}}$  different points in the domain  $\Omega$ .

### 5.2. Stochastic Ackley function

As a first test case, we introduce a stochastic version of the Ackley function, a highly irregular baseline with multiple extrema presented in [60], which takes  $P = 3$  parameters. Being real-valued ( $D = 1$ ) and two-dimensional in space ( $n = 2$ ), it is defined as



**Fig. 4.** Ackley Function (2D). The first column shows the contour plots of a random test sample with the predicted mean  $\hat{u}_D^\mu$  on the bottom, and the true solution  $u_D$  on top. The second column shows the predicted mean  $\hat{u}_D^\mu$  and standard deviation  $\hat{u}_D^\sigma$ , and the true data  $u_D$  for two random samples inside the training bounds and within the test set (top/bottom). The third column shows the results for two samples  $s_{\text{out}}$ , that are taken outside the dataset bounds and thus have more substantial uncertainties. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$$\begin{aligned}
 u : \mathbb{R}^{2+P} &\rightarrow \mathbb{R} \\
 (x, y; \mathbf{s}) &\mapsto -20(1 + 0.1s_3) \exp \left( -0.2(1 + 0.1s_2) \sqrt{0.5(x^2 + y^2)} \right) \\
 &\quad - \exp(0.5(\cos(2\pi(1 + 0.1s_1)x) + \cos(2\pi(1 + 0.1s_1)y))) \\
 &\quad + 21,
 \end{aligned} \tag{38}$$

with the non-spatial parameters vector  $\mathbf{s}$  of size  $P = 3$ , and each element  $s_i$  randomly sampled over  $\Omega = [-1, 1]$ , as in [60].

The 2D space domain  $\Omega_{xy} = [-5, 5] \times [-5, 5]$  is uniformly discretized with 400 grid points per dimension, leading to  $H = 160,000$  DOFs. With  $S = N_S = 500$  as our default number of samples of the parameters  $\mathbf{s}$ , we use a Latin Hypercube Sampling (LHS) strategy to sample each non-spatial parameter on their domain  $\Omega = [-1, 1]$  and generate the matrix of snapshots  $\mathbf{U} \in \mathbb{R}^{H \times S}$ , as well as  $N_{\text{tst}} = 100$  testing points to make a separate  $\mathbf{U}_{\text{tst}}$ . Selecting  $\epsilon = 10^{-10}$ ,  $L = 14$  coefficients are produced and matched by half of the final layer. The rest of the NN topology is chosen to include  $d = 3$  hidden layers, of widths  $l^{(1)} = l^{(2)} = l^{(3)} = 128$ . A fixed learning rate of  $\tau = 0.001$  is set for the Adam optimizer, as well as an L2 regularization with the coefficient  $\lambda = 0.01$ . The training epochs count is  $N_e = 120,000$ , and a softplus coefficient of  $\kappa = 0.01$  is used.

The training of each model in the ensemble took on average 4 minutes and 5 seconds on each of the 5 GPUs, and the total, real-time of the parallel process was 6 minutes and 23 seconds. This experiment and the following were performed on Compute Canada's Graham cluster (V100 GPUs), as well as Calcul Québec's Hélios cluster (K80 GPUs), depending on their respective availability. To picture the random initialization of each model in the ensemble, the training losses were:  $\mathcal{L} = 3.8457 \times 10^{-3}$ ,  $3.1084 \times 10^{-3}$ ,  $2.6536 \times 10^{-3}$ ,  $4.1969 \times 10^{-3}$ , and  $2.2483 \times 10^{-3}$ , down from the initial losses:  $\mathcal{L}_0 = 7.0313 \times 10^9$ ,  $9.9997 \times 10^0$ ,  $1.0000 \times 10^0$ ,  $9.9999 \times 10^{-1}$ , and  $9.9997 \times 10^{-1}$ . The overall relative errors reached were  $RE_{\text{val}} = 0.17\%$  and  $RE_{\text{tst}} = 0.16\%$ , for validation and testing, respectively. For the following experiments, these low-level details will be gathered in Appendix A.

The first column of Fig. 4 shows two contour plots of the predicted mean across the testing set as well as the analytical solution, making it easy to quickly visualize the Ackley function, its irregularity and its various local extrema. The second

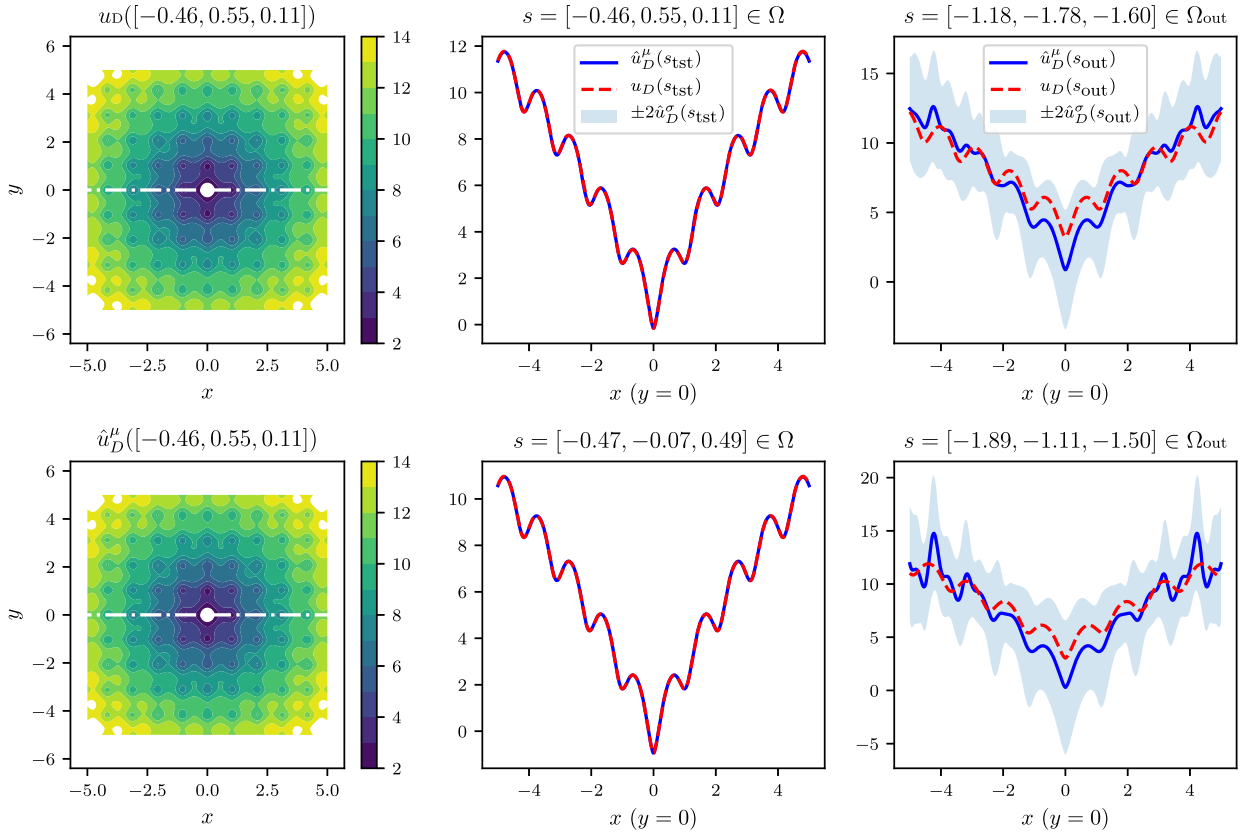


Fig. 5. Identical setup as in Fig. 4 but for the Bayesian Neural Network regression.

column shows two different random samples within the same testing set with predicted and analytical values, while the third column contains *out-of-distribution* cases, sampled in  $\Omega_{\text{out}}$ , defined as

$$\Omega_{\text{out}} = [-2, -1] \cup [1, 2]. \quad (39)$$

The most important information revealed in the last column of Fig. 4 is that the two parameters are sampled *out-of-distribution*, meaning they are outside of the dataset bounds. We can see that the predicted mean, represented by the continuous blue line, is performing poorly compared to the red dashed line, which represents the true value. This predicted mean should be approximately the same as the point estimate prediction of a regular Deep Neural Network. And, even though our out-of-distribution mean prediction is indeed “off”, thanks to the wide confidence zone defined by two times the standard deviations of the prediction, we get a warning that the model *does not know*, and therefore it does not try to make a precise claim. To picture the difference in confidence between in- and out-of-scope predictions quantitatively, we computed  $MPIW_{\text{tst}} = 0.15$  and  $MPIW_{\text{out}} = 10.0$ .

A similar experiment was then performed with the POD-BNN approach on the same dataset; the results are shown in Fig. 5. Two hidden variational layers of sizes  $l^{(1)} = l^{(2)} = 40$  were set up, with a number of epochs  $N_e = 120,000$  and a fixed learning rate of  $\tau = 0.01$ , as well as the softplus coefficient  $\kappa = 0.01$ . The prior distribution was chosen to have the standard parameters  $\pi_0 = 0.5$  and  $\pi_2 = 0.1$ , and we selected  $\pi_1 = 4.0$ . The trainable parameters  $\theta^{(j)}$  (weight or bias) of the  $j$ -th layer were randomly initialized, with

$$\theta^{(j)} = (\theta_{\mu}^{(j)}, \theta_{\sigma}^{(j)}) \sim \mathcal{N}\left(\mathbf{0}, \sqrt{\pi_0 \pi_1^2 + (1 - \pi) \pi_2^2} \mathbf{I}\right). \quad (40)$$

The training time for the BNN approach on a single GPU was 5 minutes and 5 seconds, to reach overall relative errors of  $RE_{\text{val}} = 0.68\%$  and  $RE_{\text{tst}} = 1.11\%$ , for validation and testing, respectively.

The same behavior can be observed from the Bayesian approach as for the ensembles, with tiny uncertainties predicted for the sample inside the training scope, which is expected because the data is not corrupted by noise. However, when predictions are made out-of-distribution, they are correctly pictured by a significant uncertainty revealed by the shadow around the predicted mean. Quantitatively, we report mean prediction interval widths of  $MPIW_{\text{tst}} = 0.11$  and  $MPIW_{\text{out}} = 3.22$ , which in both cases is in the same order as in the Ensembles case. In this first experiment, the same number of

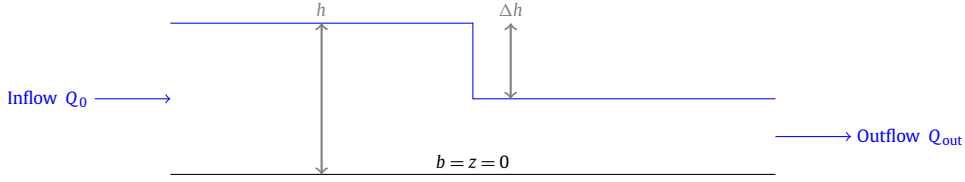


Fig. 6. Simple representation of the water flow and main quantities before a dam break ( $\Delta h > 0$ ).

training epochs has willingly been used in both cases for comparison purpose. We have noticed that POD-EnsNN reached a proper convergence much earlier, and more generally, we may further stress that POD-BNN was computationally heavier.

## 6. Flood modeling application: the Mille Îles river

After assessing how both the Deep Ensembles and the BNN version of the POD-NN model performed on a 2D benchmark problem, here we aim at a real-world engineering problem: flood modeling. The goal is to propose a methodology to predict probabilistic flood maps. Quantification of the uncertainties in the flood zones is assessed through the propagation of the input parameters' aleatoric uncertainties via the numerical solver of the Shallow Water equations.

### 6.1. Background

Just like wildfires or hurricanes, floods are natural phenomena that can be devastating, especially in densely populated areas. Around the globe, floods have become more and more frequent, and ways to predict them should be found in order to deploy safety services and evacuate areas when needed.

The primary physical phenomenon in flooding predictions involves *free surface flows* and is usually described by the Shallow Water equations for rivers and lakes, extensively studied in [61], which, in their inviscid form, are defined as follows

$$\frac{\partial}{\partial t} \int_{\Omega_{xy}} \mathbf{U} d\Omega_{xy} + \int_{\partial\Omega_{xy}} ([\mathbf{G}(\mathbf{U}) \mathbf{H}(\mathbf{U})] \cdot \mathbf{n}) d\Gamma = \int_{\Omega_{xy}} \mathbf{S}(\mathbf{U}) d\Omega_{xy} \quad \text{on } [0, T_s], \quad (41)$$

with  $T_s$  denoting the time duration, and

$$\mathbf{U} = \begin{bmatrix} h \\ hv_x \\ hv_y \end{bmatrix}, \quad \mathbf{G}(\mathbf{U}) = \begin{bmatrix} hv_x \\ hv_x^2 + \frac{1}{2}gh^2 \\ hv_x v_y \end{bmatrix}, \quad \mathbf{H}(\mathbf{U}) = \begin{bmatrix} hv_y \\ hv_x v_y \\ hv_y^2 + \frac{1}{2}gh^2 \end{bmatrix},$$

$$\mathbf{S}(\mathbf{U}) = \begin{bmatrix} 0 \\ gh(S_{0x} - S_{fx}) \\ gh(S_{0y} - S_{fy}) \end{bmatrix}, \quad \begin{bmatrix} S_{0x} \\ S_{0y} \end{bmatrix} = -\nabla b,$$

$$\text{and } \mathbf{S}_f = \begin{bmatrix} S_{fx} \\ S_{fy} \end{bmatrix} = \begin{bmatrix} \frac{m^2 v_x \sqrt{v_x^2 + v_y^2}}{h^{4/3}} \\ \frac{m^2 v_y \sqrt{v_x^2 + v_y^2}}{h^{4/3}} \end{bmatrix}.$$

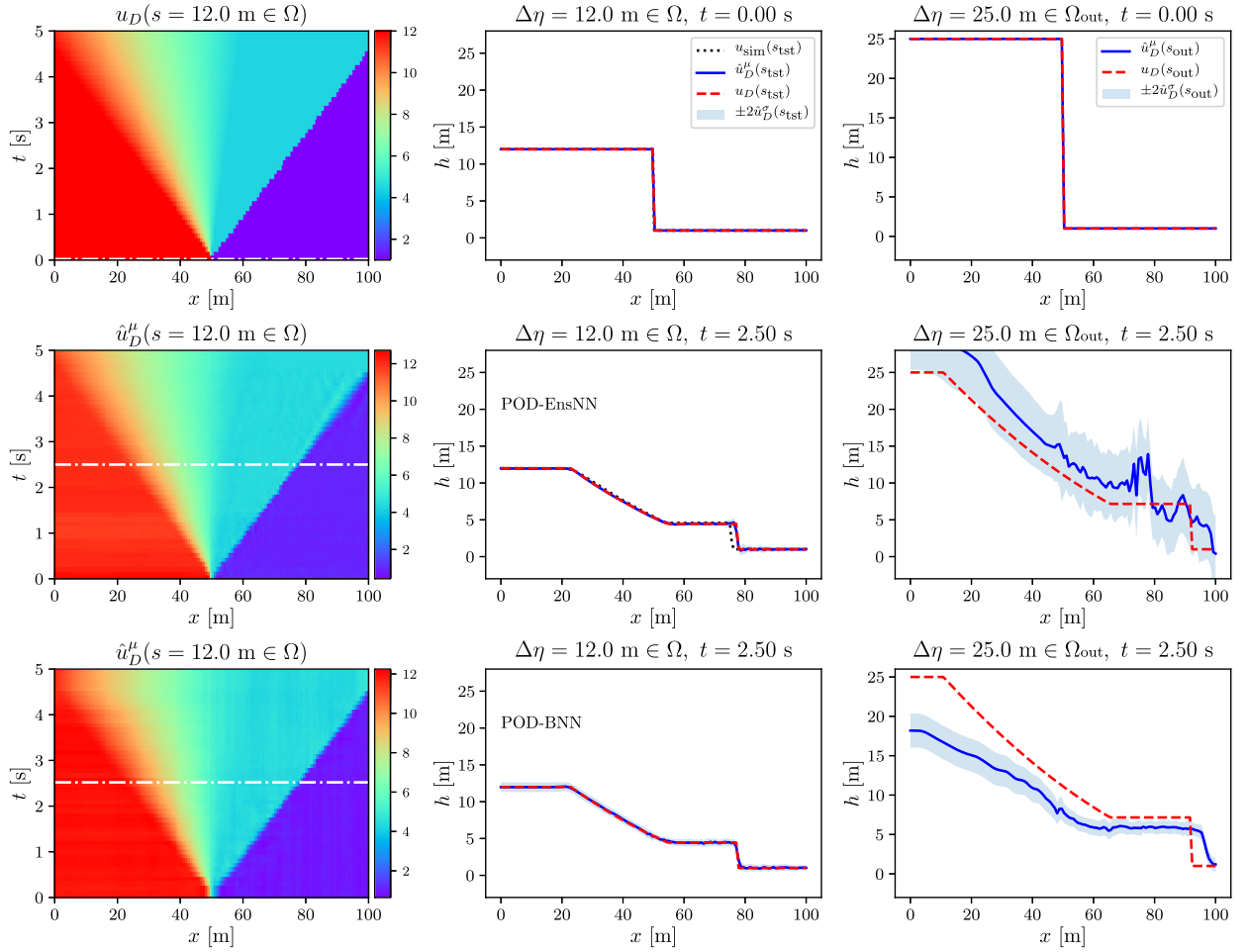
Here  $h = \eta - b$  is the water depth,  $\eta$  the free surface elevation of the water,  $(v_x, v_y)$  the velocity components,  $m$  the Manning roughness,  $g$  the gravity acceleration,  $\mathbf{S}_f$  the friction vector, and  $b$  the bottom depth, or bathymetry, for a reference level.

These equations can be discretized using finite volumes, as detailed in [61] and [10]. And, while we do already have decent numerical simulation programs to make these predictions, with well-validated software like *TELEMAC* [62] or *CuteFlow* [10], these are both computational- and time-expensive for multi-query simulations such as those used in uncertainties propagation. Therefore, it is difficult to run them in real-time, as they depend on various stochastic parameters. The POD-NN model, enriched with uncertainty quantification via Deep Ensembles and BNN, is designed to address this type of problem.

### 6.2. In-context validation with a one-dimensional discontinuous test case

We first put forward a one-dimensional test case in the Shallow Water equations application, with two goals in mind. The first is to have a reproducible benchmark on the same equations that will be used for flood modeling, with an analytically available solution, and, therefore, generable data. The second is to make sure that the solver *CuteFlow* performs correctly with respect to the analytical solution, since in future experiments, it will be our only data source.





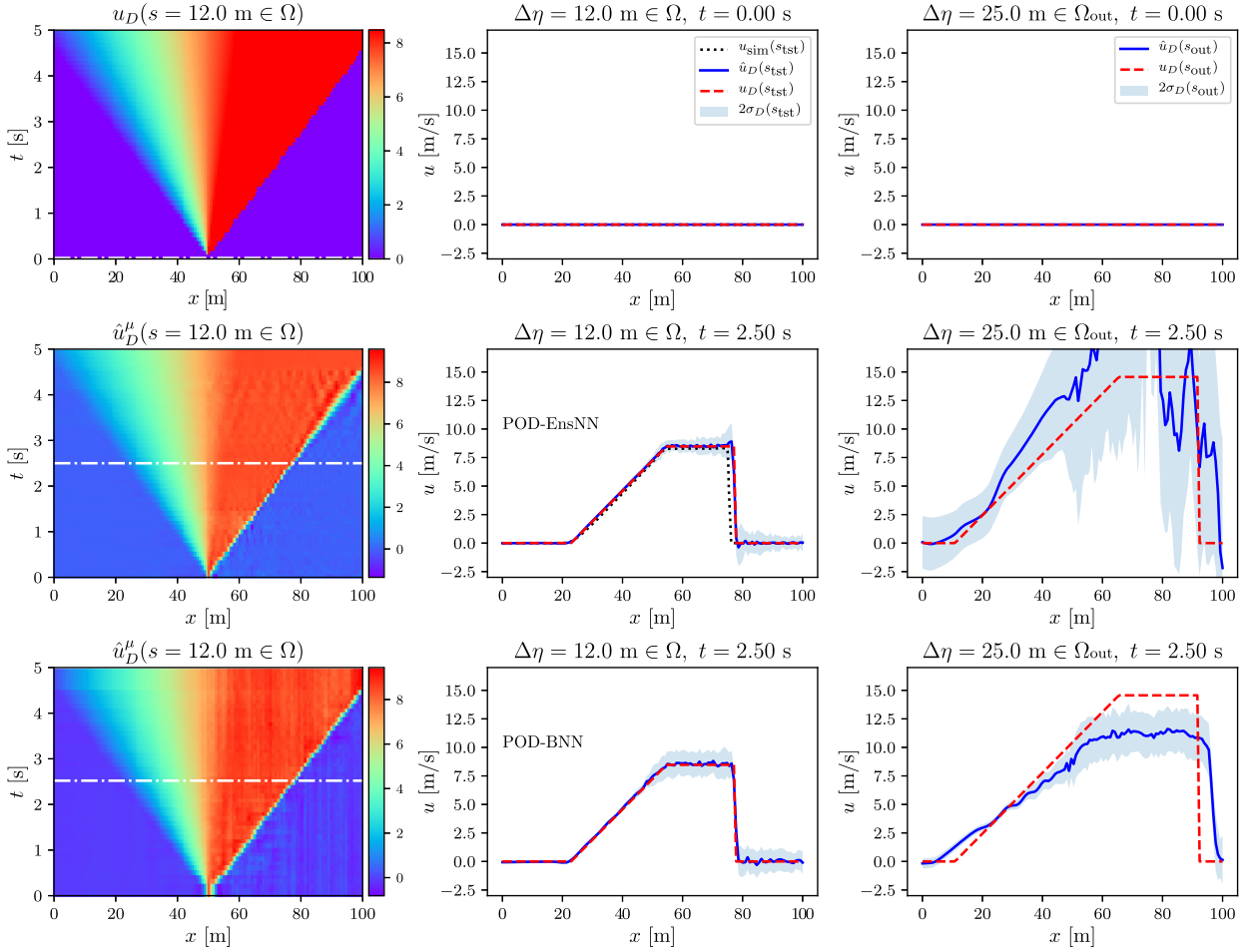
**Fig. 7.** 1D test case for SWE, water elevation results. The first two columns show results for a random sample in the test set, while the last column shows a random sample taken out-of-distribution. The white lines on the color maps denote the time steps of the last two columns. The lines  $u_{\text{sim}}$  are computed numerically by CuteFlow, and compared to the predicted mean  $\hat{u}_D$  as well as the analytical value  $u_D$ . Ensembles are used on the second row, and BNNs on the third.

The 1D domain  $\Omega_x = [0, 100]$  m is considered, with  $N_x = 132$  points, uniformly distributed. An initial condition is set up, with two levels of water depth,  $s = \Delta h$  denoting the difference, that will act as our stochastic parameter in this study, with the water depth in the outflow fixed at  $h = 1$  m. Following the initial discontinuity at  $t = 0$ , we consider  $N_t = 50$  time-steps for snapshots sampling, separated by  $\Delta t = 0.1$  s, in the domain  $\Omega_t = [0, 5]$  s. There are  $D = 2$  DOFs per node, the water depth  $h$  and the velocity  $u$ , leading to the total number of DOFs  $H = 264$ .

The dataset for the training/validation  $\mathcal{D} = \{\mathbf{X}, \mathbf{v}\}$  of size  $N = 40$  was generated from an analytical solution presented in [63], with a uniform sampling  $s$  in  $\Omega = [2, 20]$ . The same process generates a testing dataset  $\mathcal{D}_{\text{tst}} = \{\mathbf{X}_{\text{tst}}, \mathbf{v}_{\text{tst}}\}$  of size  $N_{\text{tst}}$ , with  $\mathbf{s}_{\text{tst}} = [2, 3, \dots, 20]^T$  m. Additionally, the numerical finite volume solver CuteFlow was used to generate corresponding test solutions, from which we also exported  $N_t = 50$  solutions corresponding to the uniform analytical sampling after the initial condition. This solver was run with a 2D dedicated mesh of 25551 nodes and 50000 triangular elements specifically designed to represent this 1D problem in a compatible way for the solver.

This problem is time-dependent, making the matrix of snapshots  $\mathbf{U}$  growing substantially. We therefore make use of the two-step POD algorithm, presented in Section 2.3. The results are comforting: on a dataset of size  $S = 10,000$ , with  $N_t = 100$ , the time to compute the SVD decomposition shrunk from 0.63 seconds to 0.51 by switching from the regular POD to the two-step POD algorithm, which could result in a significant gain on more massive datasets. The overall relative errors were  $RE_{\text{val}} = 3.56\%$  and  $RE_{\text{tst}} = 3.93\%$ , for validation and testing, respectively. The results are displayed in Fig. 7 for the water depth, and in Fig. 8 for the velocity. On both figures, two samples are visible, with one within the testing set, pictured on the first column as a color map for graphic visibility, and plotted for two time-steps on the second column. The first time-step is the initial condition, which is well handled by the POD compression-expansion. A black line in the second column, representing the corresponding solution computed by the numerical solver CuteFlow, is very close to the analytical solution, and thus validates it for later use in more complex cases.





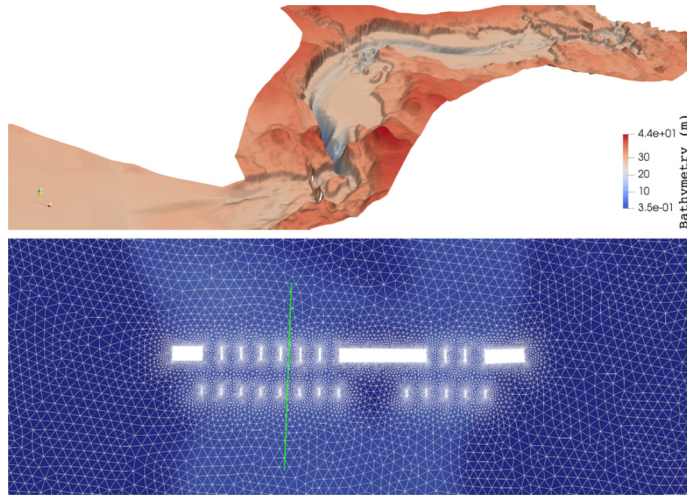
**Fig. 8.** 1D test case for SWE, velocity results. The first two columns show results for a random sample in the test set, while the last column shows a random sample taken out-of-distribution. The white lines on the color maps denote the time steps of the last two columns. The lines  $u_{\text{sim}}$  are computed numerically by CuteFlow, and compared to the predicted mean  $\hat{u}_D$  as well as the analytical value  $u_D$ . Ensembles are used on the second row, and BNNs on the third.

A second out-of-distribution sample from  $\Omega_{\text{out}} = [20, 30]$  m is plotted for the same two time-steps on the third column. The model performance within the training set was very good considering the nonlinearities involved, with relatively small uncertainties, and decreases when going out-of-distribution, as expected. We report mean predictive interval width values of  $MPIW_{\text{tst}} = 1.64$  and  $MPIW_{\text{out}} = 3.97$ , aligning with the qualitative comparison.

The Bayesian approach was also applied to this discontinuous problem; the results are indicated in the last row of Fig. 7 and 8. We had to resort to a tanh activation function to reach a decent convergence, but the out-of-distribution warning is not present, as shown in the third column of both figures. The overall relative errors were  $RE_{\text{val}} = 6.30\%$  and  $RE_{\text{tst}} = 5.32\%$ , for validation and testing, respectively. Values for the mean predictive interval width are  $MPIW_{\text{tst}} = 1.23$  and  $MPIW_{\text{out}} = 2.05$ .

Here, the POD-BNN performance is a bit less striking compared to POD-EnsNN, in connection with the computational costs and despite our best tuning efforts. This evaluation helped to reveal the difficulties involved in the Bayesian approach when discontinuities are issued for the underlying physical phenomenon. This approach is general in its essence, yet difficult to implement due to its inherent intractability involving approximations via Variational Inference. In its simplest version with the approximated posterior distribution  $q(\mathbf{w}|\theta)$  considered as a uniform distribution, it corresponds to the ensembles approach, which achieves excellent results. At the same time, the Bayesian approach, as first presented in [37] had more difficulty converging when discontinuities appeared in the physical solutions, which is not a trivial problem for Neural Networks in general, as discussed extensively in [64]. The action taken to overcome this issue was to use the common but less wide-spread hyperbolic tangent activation function.

Nonetheless, this test case allowed for a great benchmark of the numerical simulator and is another example showcasing the flexibility of the ensembles approach. We move on to real-world examples with probabilistic flooding predictions, involving first a steady context.



**Fig. 9.** Setup for the Mille Îles river in Laval, QC, Canada. On top is a representation of the river's bathymetry, given by the *Communauté Métropolitaine de Montréal*, and below, the portion of the triangle-based mesh around the piers of a bridge, which features refinements. The green line indicates a cross-section  $x'$ , studied later.

### 6.3. Probabilistic flooding maps

#### 6.3.1. River model setup

Our domain  $\Omega_{xy}$  is composed of an unstructured mesh of  $N_{xy} = 24361$  nodes, connected in 481930 triangular elements. It is represented in Fig. 9. Each node has in reality 3 degrees of freedom, but only  $N_{val} = 1$  degree of freedom, the water depth  $h$ , will be considered in this study, leading to the global number of DOFs to be  $H = 24361$  for the POD snapshots.

For this first study, we will consider the time-independent case, and have at our disposal a dataset of  $S = 180$  samples for different inflow discharge ( $Q_0$ ) values used for training, the varying parameter here, and another of  $S = 20$  used for testing, with the solution computed numerically with the software CUTEFLOW. Both datasets were uniformly sampled before being split into the domain  $\Omega = [800, 1200] \text{ m}^3\text{s}^{-1}$ . This domain was chosen to be just above the regular flow in the river of  $Q_r = 780 \text{ m}^3\text{s}^{-1}$ , [11].

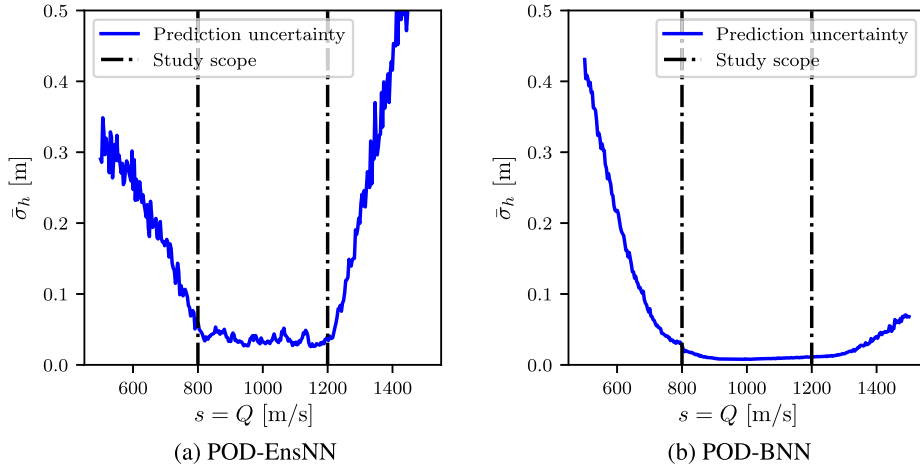
#### 6.3.2. Results

We selected a POD truncating criterion of  $\epsilon = 10^{-10}$ , producing  $L = 81$  coefficients to be matched by half of the final layer. The ReLU activation function is chosen. No mini-batching is performed, i.e., the whole dataset is run through at once for each epoch. The overall relative errors reached were  $RE_{val} = 1.90\%$  and  $RE_{tst} = 1.46\%$ , for validation and testing, respectively.

Fig. 11 shows random test predictions using the open-source visualization software Paraview [65] on two random samples for the water depth  $h$ . We can see that the flooding limits, achieved by slicing at  $h = 0.05 \text{ m}$  of water depth—in place of 0 for stability, are very well predicted when compared to the simulation results from CUTEFLOW (red line). The light blue lines can be retrieved by adding  $\pm 2$  times the standard deviations on top of the mean predictions, depicted by the blue body of water. These additional light blue lines define the confidence interval of the predicted flood lines. We consider that having this probability-distribution outcome instead of the usual point-estimate prediction of a regular network in the POD-NN framework is a step forward for practical engineering.

As in the previous experiments, we settled on a thinner architecture in the POD-BNN, which facilitates training considering the computational burden of POD-BNN. Fig. 12 depicts the same random test predictions as Fig. 11. The flooding limits are also very well predicted when compared to the simulation results from CUTEFLOW (red line). The confidence interval around these predictions is very similar to the one predicted by the POD-EnsNN, and as for the distances measured to verify it: we found a distance between the predicted mean value and the upper confidence bound of  $d_{2\sigma} = 25.36 \text{ m}$  for the POD-EnsNN results compared to  $d_{2\sigma} = 24.58 \text{ m}$  for the POD-BNN results on the first close-up shot (b), and  $d_{2\sigma} = 4.99 \text{ m}$  versus  $d_{2\sigma} = 4.34 \text{ m}$ , respectively, for the second close-up shot (c). While not being exactly equal, we assume that having the same order of magnitude is a solid accomplishment. In this application, no convergence issues for the Bayesian approach have been observed with the default configuration of a mixture prior and ReLU activation function, compared to previous attempts. These earlier efforts were notably performed on highly nonlinear and time-dependent test cases, where the Variational Inference steps were certainly facing harder circumstances.

Finally, to make sure that our *out-of-distribution* predictions were not just coincidences in the previous benchmark (see Section 5.2), we also sampled new parameters from the whole  $\Omega_{out} \cup \Omega$  domain, retrieved the mean across all DOFs of the predicted standard deviation, and rendered it in Fig. 10. We observe that uncertainties snowball as soon as we exited the space where the model *knows*, just as expected. Nonetheless, it is easy to see the difference in the magnitude of increase



**Fig. 10.** Uncertainties on the flooding case. Visualization of the average uncertainties for a range of inputs with the two approaches. The two vertical black lines denote the boundaries of the training and testing scopes.

when leaving the training bounds, which is much higher in the case of the POD-EnsNN when compared to the POD-BNN. The choice of the prior in the latter has shown to have an impact on this matter, and a more thorough choice could further improve the performance of POD-BNN.

### 6.3.3. Contribution to standard uncertainty propagation

Instead of considering the domain of the sampled inflow  $\Omega$  as simply a dataset, in the field it is often used as the source of random inputs around a central, critical point for *uncertainty propagation* tasks, as performed in a similar context in [11]. For this purpose, the use of a surrogate model is mandatory, since we wish to approximate the statistical moments of the output distributions to the model, i.e., the mean  $\mu_{up}$  and the standard deviation  $\sigma_{up}$ .

In the flood modeling problem for the Mille Îles river, the regular inflow is estimated to be on the order of  $Q_r = 780 \text{ m}^3\text{s}^{-1}$ . Our snapshots were sampled uniformly in  $\Omega = [800, 1200] \text{ m}^3\text{s}^{-1}$ , targeting a critical mean value of  $Q_{crit} = 1000 \text{ m}^3\text{s}^{-1}$ , which corresponds to an extreme flood discharge.

After having successfully trained and validated the model in Section 6.1, we now uniformly generate a new set of inputs  $\mathbf{X}_{up}$  of size  $N_{up} = 10^3$  on  $\Omega$ . Running the full POD-EnsNN model, we obtain the outputs  $\mathbf{U}_{up}$ , with the quantity of interest being the water depth  $h$ . Since our model provides a local uncertainty for each sample point, we can approximate the statistical moments using the same mixture formulas as for sample prediction  $(\mu_{*i}, \sigma_{*i})$ ,

$$\mu_{up} = \frac{1}{N_{up}} \sum_{i=1}^{N_{up}} \mu_{*i}, \quad (42)$$

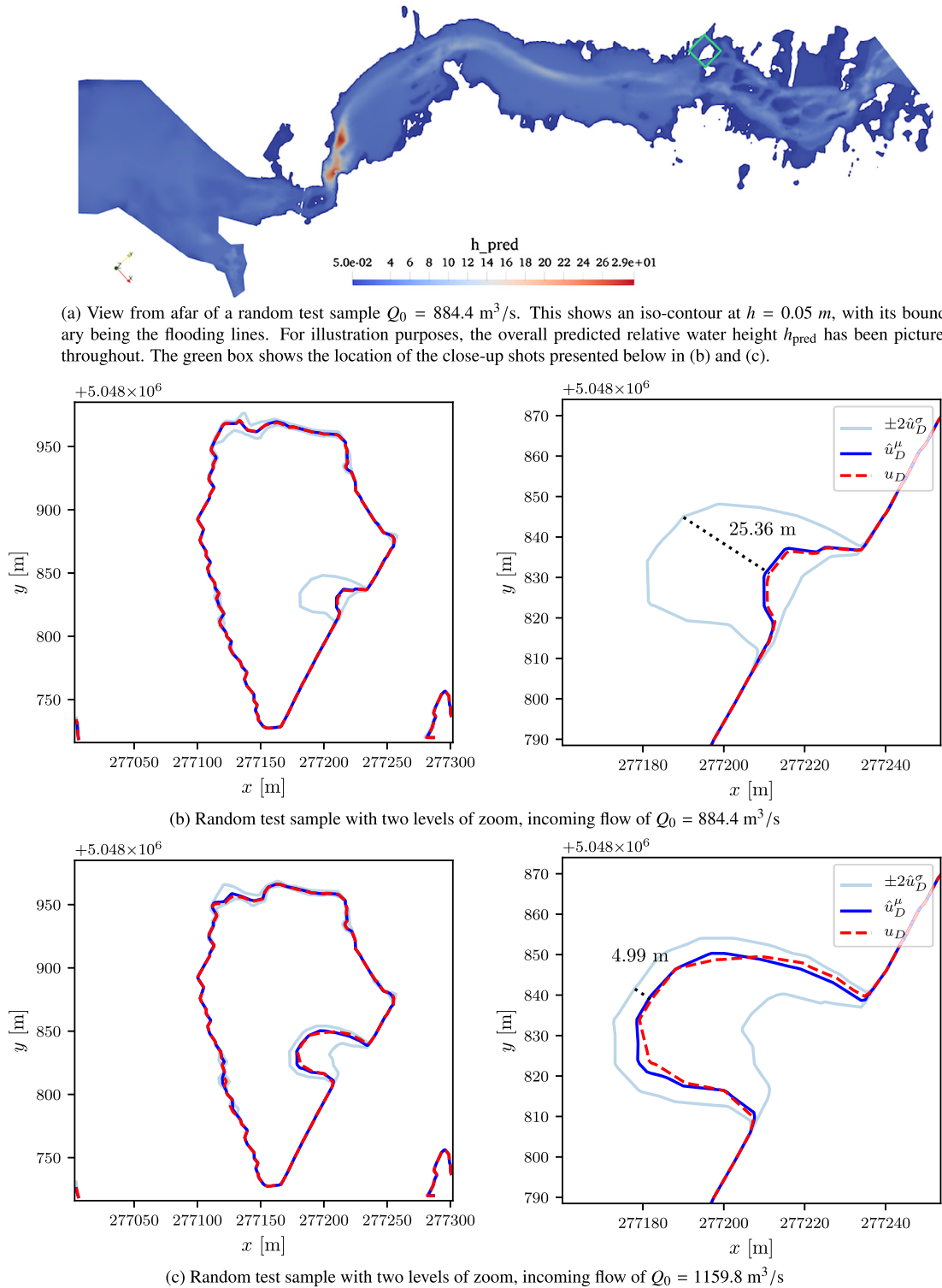
$$\sigma_{up}^2 = \frac{1}{N_{up}} \sum_{i=1}^{N_{up}} (\sigma_{*i}^2 + \mu_{*i}^2) - \mu_{up}^2. \quad (43)$$

Additionally, we monitor the regular statistical standard deviation  $\sigma_{ups}$  on the means, as a point of comparison, defined as

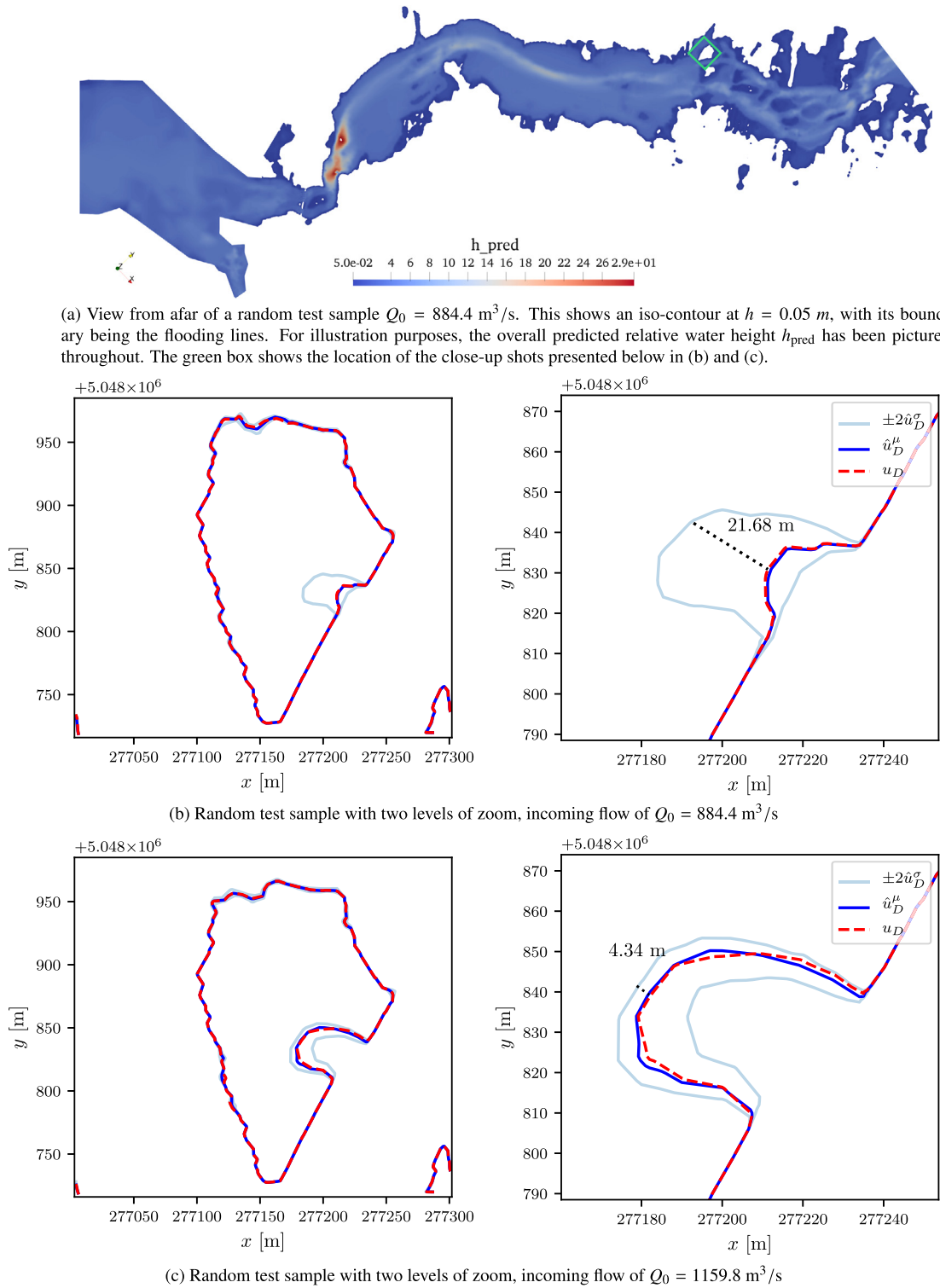
$$\sigma_{ups}^2 = \frac{1}{N_{up}} \sum_{i=1}^{N_{up}} (\mu_{*i} - \mu_{up})^2. \quad (44)$$

As a test case, the trained model of Section 6.3.2 produced two probabilistic flooding maps, depicted in Fig. 13. On the very top, a broad view of the flooding at  $h = 0.05 \text{ m}$  is visible, with the predicted  $h_{mean} = \mu_{up}$  from (42), depicted as a color map throughout, and the green box locating the two close-up shots. These are displayed in the second row for the ensembles approach, and in the third row for the Bayesian approach, for comparison purposes. On both approaches' close-ups, there are four lines on top of the mean blue water level: two green lines, showcasing two bands of the standard deviation over the predicted means only,  $\pm 2\sigma_{ups}$ , and two light blue lines, representing two bands of a standard deviation  $\pm 2\sigma_{up}$  obtained by averaging across each mean and variance predicted locally by either the POD-EnsNN or the POD-BNN framework.

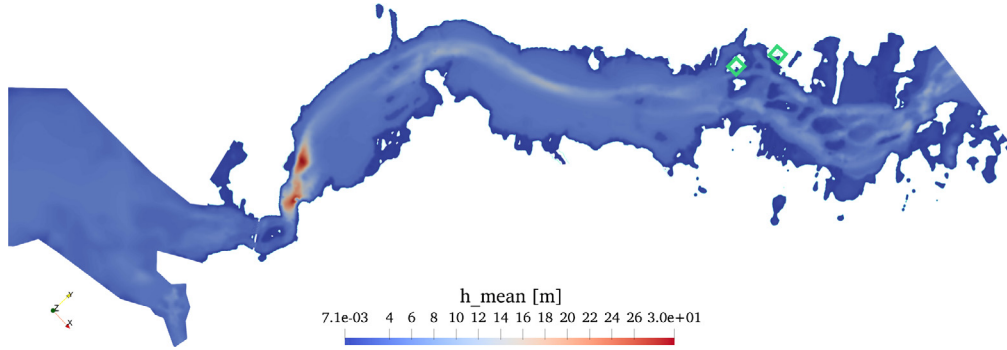
While these lines are very close in both cases, as is well-represented by the first close-up shots (left column) in Fig. 13, where the measured distance is tiny, the gap does increase sometimes, for instance, in the case of the second close-up shots (right column), where the measured difference is somewhat significant. This attests to the potential usefulness of our approach in the realm of uncertainty propagation, as it effectively combines aleatoric (due to the distribution of  $Q_0$ ) and



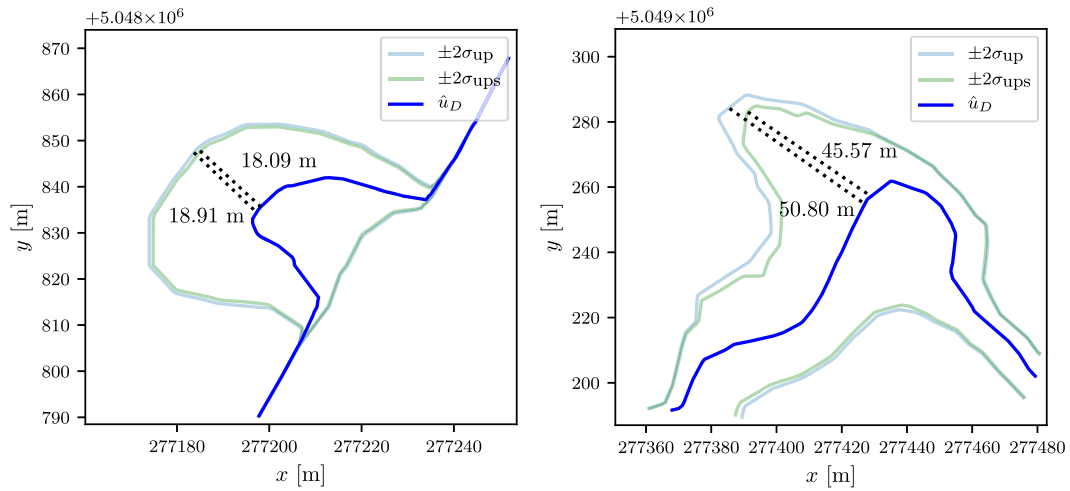
**Fig. 11.** POD-EnsNN application: flood modeling on the Mille Îles river, represented in (a). Flooding lines at  $h = 0.05 \text{ m}$  are shown on the close-up shots (b-c), with the red lines for the CuteFlow solution, and the light blue lines representing the end of the predicted confidence interval  $\pm 2\sigma_D$ . The distance between the simulated value and the upper bound is measured.



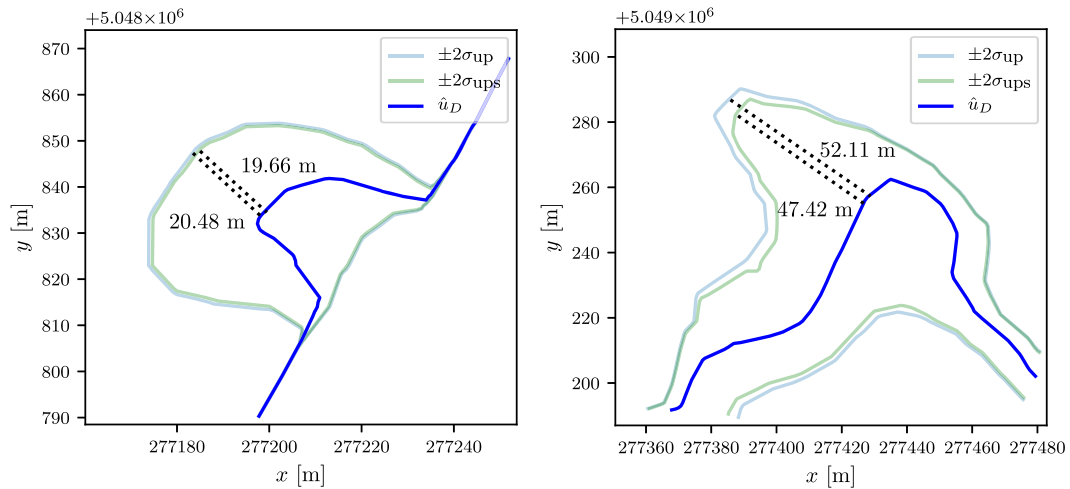
**Fig. 12.** POD-BNN application: flood modeling on the Mille îles river, represented in (a). Flooding lines at  $h = 0.05 \text{ m}$  are shown on the close-up shots (b-c), with the red lines for the CuteFlow solution, and the light blue lines representing the end of the predicted confidence interval  $\pm 2\sigma_D$ . The distance between the simulated value and the upper bound is measured.



(a) View from afar of the mean over the whole predicted domain  $\Omega$ . This perspective shows an iso-contour at  $h = 0.05$  m, with the flooding lines as its boundary. For illustration purposes, the mean predicted relative water height  $h_{\text{mean}}$  has been pictured throughout. The green boxes show the two locations of the close-up shots (shown below in (b) and (c)).



(b) POD-EnsNN: Two close-up shots, showing the differences in the uncertainty around the mean water level (in blue).



(c) POD-BNN: Two close-up shots, showing the differences in the uncertainty around the mean water level (in blue).

**Fig. 13.** Uncertainty propagation on the Mille Îles river. Flooding lines at  $h = 0.05$  m are shown in the close-up shots, with the green lines indicating  $\pm 2\sigma_{\text{ups}}$ , the standard deviation over each predicted mean, and the light blue lines representing  $\pm 2\sigma_{\text{up}}$ , the approximation over each predicted mean and variance. Distances are measured between the mean, represented by the blue lines, and each of these quantities.



epistemic (due to the modeling step) sources of uncertainty. Nonetheless, the epistemic uncertainty remains relatively minor in this case, as averaging over the quite broad domain  $\Omega$  mostly wipes away the predicted local variances.

#### 6.4. An unsteady case: the failure of a fictitious dam

While flooding prediction in the sense of generating flooded/non-flooded limits is a handy tool for public safety, it seemed promising to apply the same framework to a time-dependent case: the results of a fictitious dam break on the same river, whose model was presented in Section 6.3.1, and which is also of interest to dam owners in general.

The setup involves the same Shallow Water equations as described in Section 6.1. The domain of study is a sub-domain of the previous domain  $\Omega_{xy}$ , with only  $N_{xy} = 9734$  nodes and 18412 elements, registering one degree of freedom per node—the water elevation  $\eta$ . However, for this case, we consider  $N_t = 100$  time-steps, after the initial time  $t = 0$  s with a sampling step of  $\Delta t = 0.3$  s — which is different from the adaptive time-steps in the numerical solver.  $N_s = 100$  samples are considered for the non-spatial parameter: the water surface elevation of the inflow cross-section, considering a dried out outflow ( $\eta = b$ ) at the moment of the dam break  $s = \eta_0$ , as pictured in Fig. 6, sampled uniformly on  $\Omega = [30, 31]$  m. These samples comprise the training/validation dataset  $\mathcal{D}$ , while we consider one random test snapshot  $s_{tst}$ . Dual POD was performed with  $\epsilon_0 = 10^{-6}$  and  $\epsilon = 10^{-6}$ , producing  $L = 60$  coefficients to be matched by half of the final layer.

The training of each model in the ensemble took close to 31 minutes on each GPU. The total, real time of the parallel process was 32 minutes. The results are displayed in Fig. 14, in which, from top to bottom, there are representations of four time-steps,  $t = 0$  s,  $t = 1.5$  s,  $t = 6.0$  s, and  $t = 30.0$  s. On the left, a 3D rendering of the blue river on the orange bed is displayed to help understand the problem visually. The subsequent time-steps picture the intense dynamics that follow the initial discontinuity. The investigated cross-section, which was depicted as a green line in Fig. 9, is on the right. There is clearly a decent approximation performed by the model, considering the high nonlinearity of the problem. The uncertainty associated, obtained from (17), is represented by the light blue area around the predicted blue line. The relative errors reached in the POD-EnsNN case were  $RE_{val} = 9.8\%$  and  $RE_{tst} = 2.8\%$ , for validation and testing, respectively.

The relative errors in the POD-BNN case were  $RE_{val} = 11.65\%$  and  $RE_{tst} = 9.35\%$ , for validation and testing, respectively. The BNN training was completed by a single GPU in 1 hour 3 minutes, and the results are displayed in Fig. 15. We can observe comparable results with those of the POD-EnsNN framework, except for a decrease in the curve-fitting performance, as well as more considerable uncertainties, notably near the end of the simulation time.

To illustrate the time efficiency of the method, we measured an average computation time of 91.02 seconds per snapshot running the numerical solver CUTEFLOW on 2 parallel V100 GPUs. Unsurprisingly, evaluating the models in the Uncertainty Quantification context on a standard CPU took 2.72 seconds per snapshot in the POD-EnsNN case, and 6.94 seconds per snapshot in the POD-BNN case, reinforcing a crucial advantage of the offline/online approach.

## 7. Conclusion

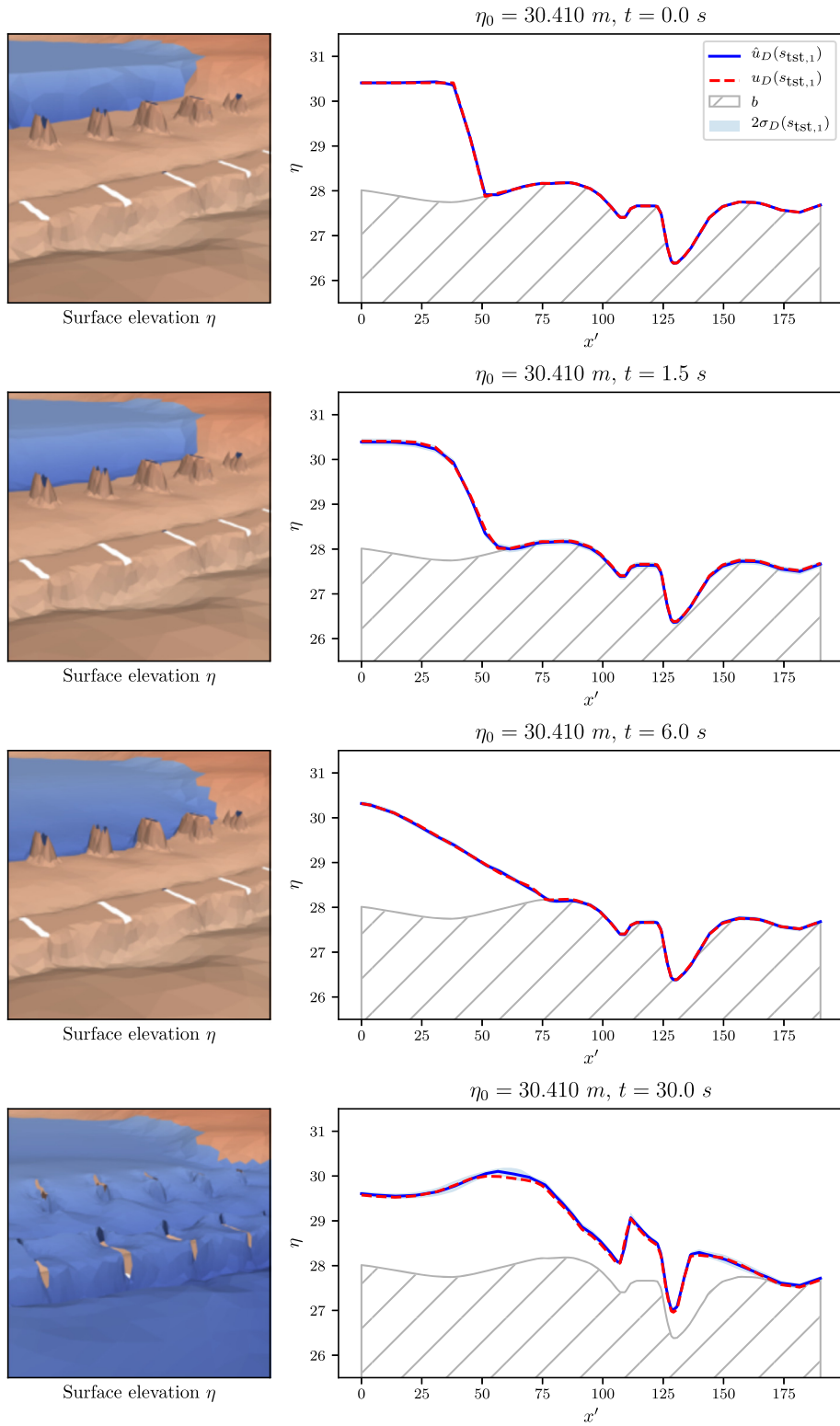
The excellent regression power of Deep Neural Networks has proved to be an asset to deploy along with Proper Orthogonal Decomposition to build reduced-order models. Their advantage is most notable when extended with recent progress in Deep Learning for a Computational Fluid Dynamics application.

Utilizing 1D and 2D benchmarks, we have shown that this approach achieved excellent results in terms of accuracy, and the training times were very reasonable, even on regular computers. It combines several state-of-the-art techniques from the reduced-order modeling and machine learning fields. Deep Ensembles and Bayesian Neural Networks were presented and compared as a way to bundle uncertainty quantification within the model. While Deep Ensembles require multiple training times, which can easily be done in parallel, Bayesian Neural Networks are trained only once, which can be a decisive advantage, especially in terms of the available computational resources. However, one has to consider the time spent finding the right hyperparameters for the Bayesian approach, its computational cost that led to longer training time while often including fewer neurons per layer (or even less layers), and in some cases resulted in less accurate results, notably in time-dependent settings, compared to the relatively *plug-and-play* behavior of ensembles, which we strongly recommend.

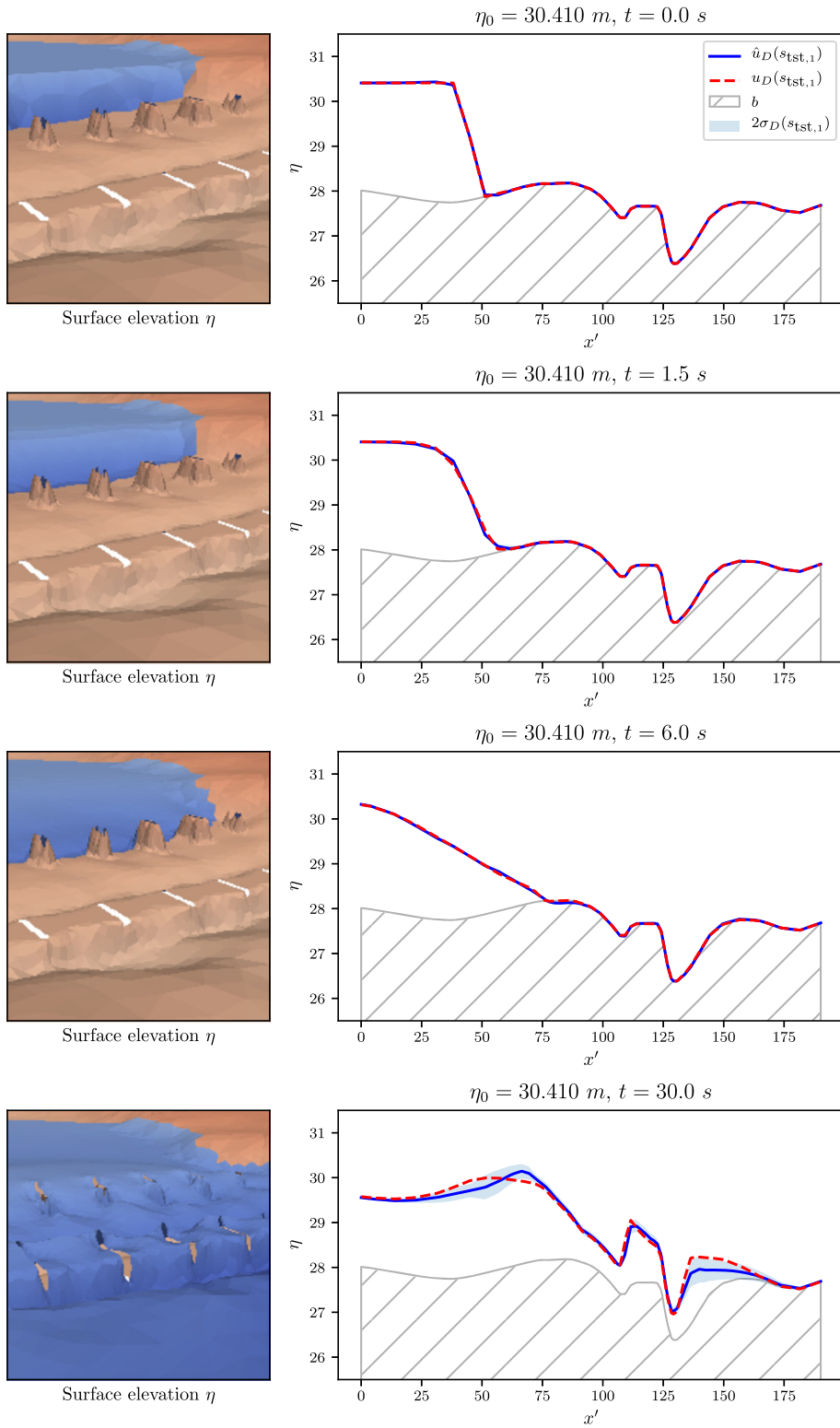
It has also been shown that while standard NNs were rapidly predicting inaccurate quantities when brought out of the training scope, adopting an uncertainty-enabled approach could produce the intended warning. This is where the uncertainty-enabled approach especially shows its worth, since the models are capable of producing flooding lines within a predicted confidence interval, either in a local prediction manner, such as a real-time context where these lines need to be computed for a new parameter, or in a more global, uncertainty propagation case, where there is an unknown extreme and critical inflow, and thus the consequences of profound changes in this quantity need to be assessed. Instead of computing the statistical moments of the output distribution from the point estimates of a surrogate model, such as a standard Neural Network, the model considers the contribution of each local uncertainty and, therefore, produces a more extensive and safer confidence area around the predicted flooding line.

Future work will focus on stabilizing the Bayesian Neural Networks approach, which still requires a much finer tuning compared to the flexibility of Deep Ensembles. Additionally, applying both methods to refined meshes will require the POD step to be performed on a sub-domain basis to avoid memory issues. This will allow to better assess the performance of the uncertainties-aware POD-NN framework in a more complicated engineering problem. While the reduced-basis compression





**Fig. 14.** Dam break with POD-EnsNN. Left: color maps of  $\eta$ . Right: plots of the water elevations in the cross-section from Fig. 9 of a random test snapshot on four time-steps, with the predictions  $\hat{u}_D$ , true values  $u_D$ , and confidence intervals, as well as the bathymetry levels in gray. The water in the river is flowing from left to right.



**Fig. 15.** Dam break with POD-BNN. Left: color maps of  $\eta$ . Right: plots of the water elevations in the cross-section from Fig. 9 of a random test snapshot on four time-steps, with the predictions  $\hat{u}_D$ , true values  $u_D$ , and confidence intervals, as well as the bathymetry levels in gray. The water in the river is flowing from left to right.

helped in the handling of the relatively large space domain of the river, the number of POD modes still has to grow with the problem's size for a fixed tolerance; hence additional research is needed to better understand the impact of the curse of dimensionality on this framework. The Bayesian approach also faced convergence issues for problems showcasing discontinuities in time-dependent settings, and decent results could only be reached by using a different activation function in the test case of Section 6.2. For long time-dependent simulations, error accumulation is known to corrupt results over time in the standard POD. However, using a multiple POD basis can enhance the accuracy and reduce the computing resources needed to apply the SVD algorithm on high-dimensional snapshot matrices [11]. The multi-POD method can easily be implemented in the framework presented in the current paper. Flood modeling offers many future exploration directions, as various other parameters have a direct influence on the results, such as the Manning roughness of the bed, as well as its elevation, and are also complicated by measurement uncertainties.

### CRedit authorship contribution statement

**Pierre Jacquier:** Conceptualization, Methodology, Software, validation, Writing - original draft preparation. **Azedine Abdedou:** Conceptualization, Methodology, Validation. **Vincent Delmas:** Investigation, Validation, Visualization. **Azzeddine Soulaïmani:** Conceptualization, Methodology, Reviewing, Supervision, Project administration, Funding.

### 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.

### Acknowledgements

This research was enabled in part by funding from the Natural Sciences and Engineering Research Council of Canada and Hydro-Québec, by bathymetry data from the [Montreal Metropolitan Community](#) (Communauté métropolitaine de Montréal), and by computational support from [Calcul Québec](#) and [Compute Canada](#).

### Appendix A. Low-level details and results for each use case

		Architecture	$N_e$	$\tau$	$\lambda$	$\zeta$	$\kappa$	$\pi$
2d_ack	EnsNN	[128, 128, 128]	120, 000	0.001	0.005	0	0.01	N/A
	BNN	[40, 40, 40]	120, 000	0.01	N/A	0	0.01	[0.5, 4.0, 0.1]
1dt_sw	EnsNN	[256, 256, 256]	100, 000	0.005	$10^{-4}$	0.001	1.00	N/A
	BNN	[256, 256, 256]	70, 000	0.01	N/A	0.001	0.01	[0.5, 0.2, 0.1]
2d_sw	EnsNN	[128, 128, 128]	120, 000	0.03	$10^{-8}$	0	1.00	N/A
	BNN	[40, 40, 40]	300, 000	0.01	N/A	0	0.01	[0.5, 4.0, 0.1]
2dt_sw	EnsNN	[128, 128, 128]	70, 000	0.01	0.001	0.001	0.01	N/A
	BNN	[128, 128, 128]	150, 000	0.003	N/A	$10^{-5}$	0.01	[0.5, 0.2, 0.1]

Documented code source for these experiments and others at <https://github.com/pierremtb/POD-UQNN>.

### References

- [1] C. Szegedy, S. Ioffe, V. Vanhoucke, A.A. Alemi, Inception-v4, inception-ResNet and the impact of residual connections on learning, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.
- [2] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, NIPS'13, Curran Associates Inc., Red Hook, NY, USA, 2013, pp. 3111–3119.
- [3] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, T. Aila, Analyzing and improving the image quality of StyleGAN, preprint, arXiv:1912.04958, 2019.
- [4] J.M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N.M. Donghia, C.R. MacNair, S. French, L.A. Carfrae, Z. Bloom-Ackerman, V.M. Tran, A. Chiappino-Pepe, A.H. Badran, I.W. Andrews, E.J. Chory, G.M. Church, E.D. Brown, T.S. Jaakkola, R. Barzilay, J.J. Collins, A deep learning approach to antibiotic discovery, *Cell* 180 (2020) 688–702.e13.
- [5] P. Benner, S. Gugercin, K. Willcox, A survey of projection-based model reduction methods for parametric dynamical systems, *SIAM Rev.* 57 (2015) 483–531.
- [6] P.J. Holmes, J.L. Lumley, G. Berkooz, J.C. Mattingly, R.W. Wittenberg, Low-dimensional models of coherent structures in turbulence, *Phys. Rep.* 287 (1997) 337–384.
- [7] L. Sirovich, Turbulence and the dynamics of coherent structures. I. Coherent structures, *Q. Appl. Math.* 45 (1987) 561–571.
- [8] J. Burkardt, M. Gunzburger, H.C. Lee, Centroidal Voronoi tessellation-based reduced-order modeling of complex systems, *SIAM J. Sci. Comput.* 28 (2006) 459–484.
- [9] M. Couplet, C. Basdevant, P. Sagaut, Calibrated reduced-order POD-Galerkin system for fluid flow modelling, *J. Comput. Phys.* 207 (2005) 192–220.
- [10] J.M. Zokagoa, A. Soulaïmani, A POD-based reduced-order model for free surface shallow water flows over real bathymetries for Monte-Carlo-type applications, *Comput. Methods Appl. Mech. Eng.* 221–222 (2012) 1–23.
- [11] J.M. Zokagoa, A. Soulaïmani, A POD-based reduced-order model for uncertainty analyses in shallow water flows, *Int. J. Comput. Fluid Dyn.* (2018) 1–15.
- [12] J. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, *J. Comput. Phys.* 363 (2018) 55–78.

- [13] Q. Wang, J.S. Hesthaven, D. Ray, Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem, *J. Comput. Phys.* 384 (2019) 289–307.
- [14] W. Ijzerman, Signal Representation and Modeling of Spatial Structures in Fluids, 2000.
- [15] S.L. Brunton, J.L. Proctor, J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, *Proc. Natl. Acad. Sci. USA* 113 (2016) 3932–3937.
- [16] J.N. Kutz, Deep learning in fluid dynamics, *J. Fluid Mech.* 814 (2017) 1–4.
- [17] K.T. Carlberg, A. Jameson, M.J. Kochenderfer, J. Morton, L. Peng, F.D. Witherden, Recovering missing CFD data for high-order discretizations using deep neural networks and dynamics learning, *J. Comput. Phys.* 395 (2019) 105–124.
- [18] J. Tao, G. Sun, Application of deep learning based multi-fidelity surrogate model to robust aerodynamic design optimization, *Aerosp. Sci. Technol.* 92 (2019) 722–737.
- [19] B.N. Hanna, N.T. Dinh, R.W. Youngblood, I.A. Bolotnov, Machine-learning based error prediction approach for coarse-grid Computational Fluid Dynamics (CG-CFD), *Prog. Nucl. Energy* 118 (2020) 103140.
- [20] B. Després, H. Jourdain, Machine Learning design of volume of Fluid schemes for compressible flows, *J. Comput. Phys.* 408 (2020) 109275.
- [21] S.L. Brunton, J.N. Kutz, Data-Driven Science and Engineering, Cambridge University Press, 2019.
- [22] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, *J. Comput. Phys.* 348 (2017) 683–693.
- [23] 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.
- [24] M. Raissi, Z. Wang, M.S. Triantafyllou, G.E. Karniadakis, Deep learning of vortex-induced vibrations, *J. Fluid Mech.* 861 (2019) 119–137.
- [25] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, Technical Report, California Univ., San Diego, La Jolla, Inst. for Cognitive Science, 1985.
- [26] S. Hochreiter, J. Schmidhuber, Long short-term memory, *Neural Comput.* 9 (1997) 1735–1780.
- [27] R. Hu, F. Fang, C. Pain, I. Navon, Rapid spatio-temporal flood prediction and uncertainty quantification using a deep learning method, *J. Hydrol.* 575 (2019) 911–920.
- [28] P.L. McDermott, C.K. Wikle, Bayesian recurrent neural network models for forecasting and quantifying uncertainty in spatial-temporal data, *Entropy* 21 (2019).
- [29] W.W. Hsieh, Machine Learning Methods in the Environmental Sciences: Neural Networks and Kernels, Cambridge University Press, 2009, <https://www.xarg.org/ref/a/0521791928/>.
- [30] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, in: *Advances in Neural Information Processing Systems*, 2017, pp. 6402–6413.
- [31] M. Valdenegro-Toro, Deep Sub-Ensembles for Fast Uncertainty Estimation in Image Classification, 2019.
- [32] J. Snoek, Y. Ovadia, E. Fertig, B. Lakshminarayanan, S. Nowozin, D. Sculley, J. Dillon, J. Ren, Z. Nado, Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift, in: *Advances in Neural Information Processing Systems*, 2019, pp. 13969–13980.
- [33] D.J.C. Mackay, Probable networks and plausible predictions – a review of practical Bayesian methods for supervised neural networks, *Netw. Comput. Neural Syst.* 6 (1995) 469–505.
- [34] D. Barber, C. Bishop, Ensemble learning in Bayesian neural networks, *Nato ASI Ser. F, Comput. Syst. Sci.* (1998) 215–237.
- [35] A. Graves, Practical variational inference for neural networks, in: J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, K.Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems*, vol. 24, Curran Associates Inc., 2011, pp. 2348–2356, <http://papers.nips.cc/paper/4329-practical-variational-inference-for-neural-networks.pdf>.
- [36] J.M. Hernandez-Lobato, R. Adams, Probabilistic backpropagation for scalable learning of Bayesian neural networks, in: F. Bach, D. Blei (Eds.), *Proceedings of the 32nd International Conference on Machine Learning, PMLR, Lille, France*, in: *Proceedings of Machine Learning Research*, vol. 37, 2015, pp. 1861–1869, <http://proceedings.mlr.press/v37/hernandez-lobato15.html>.
- [37] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural networks, in: *Proceedings of the 32nd International Conference on Machine Learning - Volume 37, ICML'15, JMLR.org*, 2015, pp. 1613–1622.
- [38] G.E. Hinton, D. van Camp, Keeping the neural networks simple by minimizing the description length of the weights, in: *Proceedings of the Sixth Annual Conference on Computational Learning Theory, COLT '93, COLT '93, Association for Computing Machinery, New York, NY, USA*, 1993, pp. 5–13.
- [39] L. Yan, T. Zhou, An adaptive surrogate modeling based on deep neural networks for large-scale Bayesian inverse problems, *arXiv:1911.08926*, 2019.
- [40] L. Yang, X. Meng, G.E. Karniadakis, B-pinns: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, *arXiv:2003.06097*, 2020.
- [41] D.A. Nix, A.S. Weigend, Estimating the mean and variance of the target probability distribution, in: *Proceedings of 1994 IEEE International Conference on Neural Networks*, vol. 1, ICNN'94, IEEE, 1994, pp. 55–60.
- [42] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision?, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017, pp. 5574–5584, <http://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision.pdf>.
- [43] A. Krogh, J.A. Hertz, A simple weight decay can improve generalization, in: J.E. Moody, S.J. Hanson, R.P. Lippmann (Eds.), *Advances in Neural Information Processing Systems*, vol. 4, Morgan-Kaufmann, 1992, pp. 950–957, <http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.pdf>.
- [44] D.P. Kingma, J. Ba Adam, A method for stochastic optimization, preprint, *arXiv:1412.6980*, 2014.
- [45] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, *Nature* 323 (1986) 533–536.
- [46] S. Linnainmaa, Taylor expansion of the accumulated rounding error, *BIT Numer. Math.* 16 (1976) 146–160.
- [47] A. Sergeev, M. Del Balso, Horovod: Fast and Easy Distributed Deep Learning in TensorFlow, 2018.
- [48] J. Yao, W. Pan, S. Ghosh, F. Doshi-Velez, Quality of Uncertainty Quantification for Bayesian Neural Network Inference, 2019.
- [49] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: *International Conference on Learning Representations*, 2014, <http://arxiv.org/abs/1312.6199>.
- [50] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and Harnessing Adversarial Examples, 2014.
- [51] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Networks, 2014.
- [52] R.M. Neal, Probabilistic Inference Using Markov Chain Monte Carlo Methods, Technical Report, 1993.
- [53] R.M. Neal, Bayesian Learning for Neural Networks, Technical Report, 1995, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.446.9306&rep=rep1&type=pdf>.
- [54] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
- [55] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, in: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings, International Conference on Learning Representations, ICLR, ICLR, 2014*, *arXiv:1312.6114*.
- [56] M. Abadi, TensorFlow: a system for large-scale machine learning, <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>, 2016.

- [57] J.V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, R.A. Saurous, TensorFlow Distributions, 2017.
- [58] M. Krasser, Variational inference in Bayesian neural networks - Martin Krasser's blog, <http://krasserm.github.io/2019/03/14/bayesian-neural-networks/>, 2019.
- [59] S.K. Lam, A. Pitrou, S. Seibert, Numba: a LLVM-based python JIT compiler, in: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, LLVM '15, ACM, New York, NY, USA, 2015, pp. 7:1–7:6.
- [60] X. Sun, X. Pan, J.-I. Choi, A Non-intrusive Reduced-Order Modeling Method Using Polynomial Chaos Expansion, 2019.
- [61] E.F. Toro, Shock-Capturing Methods for Free-Surface Shallow Flows, John Wiley, 2001.
- [62] J.-C. Galland, N. Goutal, J.-M. Hervouet, TELEMAC: a new numerical model for solving shallow water equations, Adv. Water Resour. 14 (1991) 138–148.
- [63] C. Wu, G. Huang, Y. Zheng, Theoretical solution of dam-break shock wave, J. Hydraul. Eng. 125 (1999) 1210–1214.
- [64] B. Llanas, S. Lantarón, F.J. Sáinz, Constructive approximation of discontinuous functions by neural networks, Neural Process. Lett. 27 (2008) 209–226.
- [65] J. Ahrens, B. Geveci, C. Law, Paraview: An End-User Tool for Large Data Visualization, 2005.