Environmental Data Science (2025), 4: e53, 1–26
doi:10.1017/eds.2025.10027

CAMBRIDGE
UNIVERSITY PRESS

METHODS PAPER

Uncertainty quantification for deep learning

Peter Jan van Leeuwen, Jui-Yuan Christine Chiu and Chen-Kuang Kevin Yang

Department of Atmospheric Science, Colorado State University, USA
Corresponding author: Peter Jan van Leeuwen; Email: peter.vanleeuwen@colostate.edu

Received: 18 May 2025; **Revised:** 26 October 2025; **Accepted:** 07 November 2025

Keywords: deep learning; machine learning; uncertainty quantification

Abstract

We present a critical survey on the consistency of uncertainty quantification used in deep learning and highlight partial uncertainty coverage and many inconsistencies. We then provide a comprehensive and statistically consistent framework for uncertainty quantification in deep learning that accounts for all major sources of uncertainty: input data, training and testing data, neural network weights, and machine-learning model imperfections, targeting regression problems. We systematically quantify each source by applying Bayes' theorem and conditional probability densities and introduce a fast, practical implementation method. We demonstrate its effectiveness on a simple regression problem and a real-world application: predicting cloud autoconversion rates using a neural network trained on aircraft measurements from the Azores and guided by a two-moment bin model of the stochastic collection equation. In this application, uncertainty from the training and testing data dominates, followed by input data, neural network model, and weight variability. Finally, we highlight the practical advantages of this methodology, showing that explicitly modeling training data uncertainty improves robustness to new inputs that fall outside the training data, and enhances model reliability in real-world scenarios.

Impact Statement

This survey paper presents a critical overview of uncertainty quantification in deep learning. Key limitations are identified in capturing the full range of uncertainties encountered in real-world applications. A complete and consistent uncertainty quantification methodology is proposed. Through both simple illustrative examples and a complex real-world case study, the paper demonstrates where current methods fall short and how the proposed complete framework offers a more robust and statistically consistent alternative. Comparative results underscore the advantages of the new methodology in effectively addressing uncertainty across multiple sources.

1. Introduction and initial survey

Many processes in the geosciences are highly complex, computationally expensive, or not fully understood. In such cases, machine learning—particularly deep learning—has emerged as a valuable tool to either replace costly numerical models or to learn relationships between variables when governing equations are unknown. For instance, radiative transfer processes are well understood but computationally expensive, whereas cloud microphysics is only partially understood and cannot be explicitly resolved

This research article was awarded Open Data badge for transparent practices. See the Data Availability Statement for details.

at the scales used in weather or climate models. Despite promising applications, the adoption of deep learning in scientific domains has been slow, primarily due to the lack of robust uncertainty quantification, which is essential for model comparison, forecasting, and risk assessment.

Uncertainty can be defined in many ways. A common distinction in statistics is between aleatoric and epistemic uncertainty. Aleatoric uncertainty, also known as stochastic or random uncertainty, refers to inherent variability that arises each time an experiment is repeated, such as measurement noise from a well-calibrated instrument. Epistemic uncertainty, also called systematic or structural uncertainty, stems from limitations in our models or understanding, such as uncertainty related to the design of a neural network. In practice, the boundary between these two types of uncertainty is often ambiguous. Moreover, in the geosciences, repeated experiments are rarely possible due to the non-stationary nature of the Earth system, making it difficult to operationally define aleatoric uncertainty. For this reason, we do not explicitly distinguish between the two. Instead, we adopt a practical Bayesian perspective, interpreting all uncertainties as expressions of rational belief informed by available evidence.

To discuss uncertainty quantification in deep learning, we first introduce some notation. Deep learning networks define a mapping from input vector x to output vector z via a neural network:

$$z = f(x, w), \tag{1}$$

where $f(x, w)$ denotes the neural network function parameterized by the weight vector w, which is learned from a dataset of input–output pairs. The function f comprises nested nonlinear transformations, enabling the model to fit highly complex relationships in the data. The flexibility of neural networks stems from the high dimensionality of w, which often includes thousands of parameters (weights and biases).

Several sources contribute to the uncertainty in the output z. From Eq. (1), we can identify direct contributions from uncertainty in the input vector x, uncertainty in the weight vector w, and uncertainty in the functional form of $f(..)$ because the neural network will not be perfect in mapping input to the output vector. Additionally, since w is inferred from training and testing data, uncertainties in these datasets must also be considered, even if not explicitly shown in Eq. (1). Another critical source of uncertainty arises when a new input x lies outside the range of the training data. However, because this extrapolation uncertainty lacks a principled treatment (see the Discussion), we focus on the first four sources in this study. Once quantified, these uncertainties should be combined to yield a total predictive uncertainty described by a probability density function (pdf). Yet, no existing practical method consistently incorporates all these sources into a single pdf.

To better explain the underlying difficulty and the incompleteness of existing methods for uncertainty quantification, we need to refer to Bayes Theorem. This theorem states that the pdf of a random variable A given knowledge of a variable B, $p(A|B)$, can be expressed as the pdf of random variable A without knowing B, so $p(A)$, and the likelihood $l(B|A)$, as:

$$p(A|B) = \frac{l(B|A)}{p(B)} p(A). \tag{2}$$

We call $p(A)$ the prior pdf of A, and $p(A|B)$ the posterior pdf of A given B. The pdf of B in the denominator is a normalization factor because B is not a random variable, but has a given value. $l(B|A)$ represents the likelihood that A gives rise to B, which is a function of random variable A. It determines how likely it is to find value B for each value of random variable A. For future reference, we also use the theorem in its conditional form:

$$p(A|B, C) = \frac{l(B|A, C)}{p(B|C)} p(A|C), \tag{3}$$

which is just Bayes Theorem with all the factors conditioned on another set of variables C.

In the following, we will discuss a few common methods to provide uncertainty estimates in machine learning, and provide a first critical discussion of their basic assumptions. As we will see, among the main sources of uncertainty mentioned above, most attention has been given to the uncertainty in the weight vector w and the uncertainty in the trained neural network $f(..)$.

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

1.1. Bayesian deep learning

In theory, the uncertainty of w in an neural network can be derived directly from Bayes Theorem, which is the basis of Bayesian Deep Learning (MacKay, 1992; Neal, 1996; Andrieu et al., 2003). From Eq. (2), if we choose $A = w$ and B as the training data, we know that the posterior uncertainty in w given a training dataset is determined by the prior uncertainty in w, and a likelihood that indicates how likely the training data are if the weight vector is equal to a certain w.

Since the function $f(..)$ in deep learning is highly nonlinear, this likelihood, and hence the uncertainty pdf for the weight vector w, will have an extremely complex shape (e.g., many local minima and maxima). For low-dimensional systems, one can use methods such as Markov-Chain Monte Carlo (MCMC), Langevin sampling, and Hybrid or Hamiltonian MC to accurately describe the posterior pdf (Andrieu et al., 2003; Van Leeuwen et al., 2015), but these methods do not scale well for high-dimensional problems, as is often the case in deep learning applications. To alleviate this issue, approximations to the posterior pdf have been proposed, such as the Gaussian or Laplace approximation, or variational Bayes (approximated pdf by a finite set of pdfs of a certain shape, e.g., a Gaussian mixture). These methods need the computation of the Hessian, the second derivative of the loss function, which can be computationally expensive. Apart from the crude Gaussian approximation, all these methods remain very expensive in high-dimensional problems, which pose difficulties in practice (Blundell et al., 2015).

1.2. Bagging and bootstrapping

A popular machine-learning method for uncertainty quantification related to the weights w is bootstrapping or Bagging Breiman (1996), which uses an ensemble of weight vectors to generate an ensemble of outputs. In this method, the weight vector ensemble can be generated in several ways, e.g., by randomly drawing initial weights and by randomly varying the order in which training batches are used during training of the neural network, or by giving each neural network a subset of the training data. When a new input vector is presented to the neural network, each trained weight vector in the resulting ensemble is used to generate one output value and interpreted to have the same probability. This, then, results in an ensemble of output values that is suggested to represent its uncertainty pdf. Unfortunately, while this method gives a spread in the output values, this spread does not represent a proper uncertainty estimate. The reason is that during training, the ensemble of initial weight vectors is transformed into an ensemble of trained weight vectors. Because this transformation is nonlinear, the pdf of the equally probable initial weight vectors is deformed into a pdf with a complicated shape determined by the likelihood of each weight vector to produce the training data (corresponding to the $l(B|A)$ factor above). Hence, the trained weight vectors are not equally likely anymore, and neither are the output values based on the neural network's that use these weights. The simple treatment of equally likely output values, as assumed in Bagging, is inappropriate.

This issue with the likelihood of the weight vectors is more troublesome than might be anticipated. In principle, the likelihood value of each trained weight vector, also called the importance weight, can be properly calculated and included to evaluate the ensemble of weight vectors. However, in practice, one weight vector typically fits the training data significantly better than the others, resulting in a much higher importance weight. This dominance effectively collapses the ensemble to a single neural network, eliminating the diversity needed for meaningful uncertainty estimation and yielding an unrealistically low (often near-zero) uncertainty. While this challenge is acknowledged in Bayesian Deep Learning, where methods have been developed to construct ensembles of approximately equally likely weight vectors, it is overlooked in Bagging and related approaches, essentially making the uncertainty quantification useless. We return to this critical issue in detail in Section 3.3.

1.3. Monte-Carlo Dropout

Another popular method for uncertainty quantification related to the weights w is Monte-Carlo (MC) Dropout (Gal and Ghahramani, 2016). In conventional dropout, a method used to avoid overfitting,

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

the weight of each neuron is set to zero during the training phase with a probability P. After training, all weights are used, but their trained value is multiplied by P to generate one neural network. In Monte-Carlo Dropout, the same dropout procedure used during training is also used during prediction. In other words, the trained weights are also put randomly to zero with probability P during prediction. This prediction procedure is performed several times to generate an ensemble of weight vectors, leading to an ensemble of output values. This output ensemble is then suggested to provide an uncertainty estimate.

The issue with MC Dropout is that during training one realization of a trained weight vector is created. This weight vector is randomly perturbed by dropout to generate an ensemble of perturbed trained weight vectors. What should be the next step is to importance-weigh each of those perturbed weight vectors by their likelihood value, as discussed above for Bagging, but that step is omitted. Hence, similar to Bagging, the uncertainty quantification is inappropriate. Furthermore, it fails a fundamental requirement of Bayesian inference: the posterior uncertainty should decrease as more data becomes available. This behavior has been shown to be absent in MC Dropout (Osband, 2016; Verdoja and Kyrki, 2021, and references in these papers).

1.4. Estimating uncertainty distribution parameters directly

The above discussion focused on uncertainties in the weight vector w. A second category of uncertainty-quantification approaches tries to estimate the pdf of output uncertainty directly. Instead of predicting output z, the neural network predicts:

$$p_\alpha(z) = f_\alpha(x, w_\alpha) \tag{4}$$

where $p_\alpha(z)$ denotes the uncertainty pdf of output z with parameters α, and the subscript α on the weight vector w and on f denotes that the target of estimation are the distribution parameters α, not z itself as in Eq. (1). The difference from the methods above is that the uncertainty pdf in z is not determined via an ensemble of weight vectors w, but instead, one weight vector is trained to provide the pdf parameters directly. Often, a Gaussian assumption is used, in which the mean and variance are estimated; however, other standard distributions can also be used (Nix and Weigend, 1994; Wang et al., 2016).

1.5. Evidential deep learning

Evidential deep learning for deep regression is an extension of the above methodology in which not only the parameters of the pdf of z are learned, but also the uncertainty in those parameters, e.g. Amini et al. (2020). Specifically, a Normal-Inverse-Gamma distribution is used to describe the uncertainty in the mean and variance parameters. In this way, part of the epistemic or structural uncertainty is addressed.

1.6. Quantile regression

While the two methods discussed above assume a certain shape for the uncertainty pdf, typically Gaussian, in quantile regression the pdf is allowed to have any shape, at least in principle. The neural network predicts:

$$P_z = f_P(x, w_P) \tag{5}$$

where P_z denotes predefined quantiles of the uncertainty pdf of output z, and the subscript P on the weight vector w and on f denotes that the target of estimation iare these quantiles (Wang et al., 2016; Pfreundschuh et al., 2018; Sønderby et al., 2020). Quantile regression works well for a low-dimensional output space but becomes impractical in high-dimensional problems, because the number of output variables required to describe histograms increases dramatically with the dimension. To avoid the latter issue, this method is often used to estimate the marginal uncertainty pdfs instead of the full joint pdf.

1.7. Deep Ensembles

However, the methods in the second and third categories fail to account for the uncertainty in the weight vector w_p. This omission is partly corrected in so-called Deep Ensembles (Lakshminarayanan et al.,

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

2017). Deep Ensembles generates an ensemble of pdf parameters, or, alternatively, an ensemble of histograms, by training multiple networks, each initialized with different untrained weight vectors and using distinct training batch orders. The resulting ensemble of pdf parameters is further averaged to represent the pdf of z, or the histograms themselves are averaged. Similar to Bagging and MC dropout, the simple average of the ensemble overlooks the fact that the trained weight vectors w_p, and thus the output ensemble members, are not equally likely.

As mentioned above, the literature has concentrated on quantifying uncertainty in the weight vector w or tried to estimate the uncertainty pdf of z directly. None of the machine-learning methods discussed above has considered the uncertainty in the training and testing data, or in the new input data. While uncertainties in the training, testing, and new input data might be small in some fields, they can be significant in the geosciences, where real-world data, such as direct measurements, model outputs, or reanalysis data, often contain substantial errors that cannot be ignored.

1.8. Conformal prediction

A completely different category is conformal prediction Balasubramanian et al. (2014). It is based on the concept of dissimilarity, or nonconformity, of a new prediction compared to previous input data and their predictions from the neural network, so that the prediction interval size is larger for new “unusual” instances and smaller for “typical” instances. First, the neural network model is trained on a dataset. Then, a nonconformity score is defined by the user, which measures the extent to which a data point deviates from the model’s predictions. For instance, one can use the difference between the predicted value and the true value as a score. Given new input data, conformal prediction uses the nonconformity scores from the testing data to calculate a prediction interval. The idea is to find the smallest set or interval such that, with a predefined confidence level (like 90%), the true value falls inside. Hence, the method produces a confidence interval, not an uncertainty estimate as Bayesian methods do, and it ignores uncertainty in the weight vector. Furthermore, in the geosciences, the training, testing, and new input data all contain uncertainties, and especially including uncertainty in output training data is not straightforward with this methodology. In fact, one typically relies on Bayesian methods to do this, resulting in an interesting, but perhaps ad hoc mixture of Bayesian and significance testing methodologies.

1.9. Recent reviews on uncertainty quantification

To conclude, while many methods have been developed for quantifying uncertainty in the output of a deep-learning neural network, they are either incomplete or statistically inappropriate. Recent articles by Abdar et al. (2021), Gawlikowski et al. (2023), and He et al. (2025) provide comprehensive overviews of existing techniques, but they lack critical discussions of the statistical properties of these methodologies. As a result, understanding the true meaning of uncertainty estimates from these approaches remains challenging. The review by Cheng et al. (2023) discusses the connection between data assimilation, machine learning, and uncertainty quantification, but the uncertainty quantification methods mentioned above are quoted without a critical review.

1.10. Our contribution

In this paper, we present a novel approach that accounts for all sources of uncertainty. We derive an expression for the exact pdf and provide an efficient methodology for practical computation. In Section 2, we outline how to systematically incorporate all uncertainty sources. In Section 3, we introduce a practical approach to calculate the relevant uncertainties and combine them effectively. Specifically, we explain how proposal densities, a standard method in statistics, can be used to address the likelihood issue associated with trained weight vectors w. In Section 4, we demonstrate the methodology on a simple toy problem and on an application to predict cloud process rates and offer a physical interpretation of the resulting uncertainty. Finally, we conclude with a discussion in the last section.

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

2. Uncertainty propagation in Deep Learning

The uncertainty in the output z can be described by the pdf $p(z|x,\theta_{tr},\theta_{te})$, which is the pdf of z given an input vector x and training and testing data θ_{tr} and θ_{te}, respectively. The training and testing data consist of input and output pairs (X,Z). The first step in deriving $p(z|x,\theta_{tr},\theta_{te})$ is to incorporate the uncertainty in the input vector x. Following ideas from data assimilation (Van Leeuwen et al., 2015; Van Leeuwen, 2020; Evensen et al., 2022), we can consider x as a random draw from a distribution $p(x)$, centered around a true input vector. This true input vector is not known, but, as we will show later, this is not a concern. Let us consider all possible true input vectors that would lead to us drawing x. We denote these possible true inputs by x^t. We now use the identity:

$$p(A|C) = \int p(A,B|C)p(B)\,dB \tag{6}$$

to incorporate the uncertainty in the new input vector x as follows:

$$\begin{aligned} p(z|x,\theta_{tr},\theta_{te}) &= \int p(z,x^t|x,\theta_{tr},\theta_{te})\,dx^t \\ &= \int p(z|x^t,x,\theta_{tr},\theta_{te})p(x^t|x,\theta_{tr},\theta_{te})\,dx^t \\ &= \int p(z|x^t,\theta_{tr},\theta_{te})p(x^t|x)\,dx^t, \end{aligned} \tag{7}$$

where for the second equality, the identity

$$p(A,B|C) = p(A|B,C)p(B|C) \tag{8}$$

is used. For the third equality, we used that, if the true input was x^t, then x does not provide extra information on z. Furthermore, since input vectors do not depend on the training or testing data, we have $p(x^t|x,\theta_{tr},\theta_{te}) = p(x^t|x)$. This pdf describes the probability of each possible true input x^t given that we have input sample x. This pdf is exactly the uncertainty pdf of x, as that uncertainty pdf describes the probability of the true input by definition.

The second step is to incorporate the uncertainty in the training and testing data. We can consider the training and testing data pairs as a random draw from a distribution $p(\theta_{tr},\theta_{te})$ centered around the true data $\left(\theta^t_{tr},\theta^t_{te}\right)$. Similar to the way of how the uncertainty in the new input vector x is handled, let us consider all possible true training and testing data $\left(\theta^t_{tr},\theta^t_{te}\right)$ that would lead us to draw $(\theta_{tr},\theta_{te})$. This allows us to write:

$$\begin{aligned} p(z|x,\theta_{tr},\theta_{te}) &= \int p(z|x^t,\theta_{tr},\theta_{te})p(x^t|x)\,dx^t \\ &= \int\int\int p\left(z,\theta^t_{tr},\theta^t_{te}|x^t,\theta_{tr},\theta_{te}\right)p(x^t|x)\,dx^t d\theta^t_{tr},d\theta^t_{te} \\ &= \int\int\int p\left(z|x^t,\theta^t_{tr},\theta^t_{te}\right)p\left(\theta^t_{tr}|\theta_{tr}\right)p\left(\theta^t_{te}|\theta_{te}\right)p(x^t|x)\,dx^t d\theta^t_{tr},d\theta^t_{te}, \end{aligned} \tag{9}$$

where we used the identity in Eq. (8) and the same reasoning as for the new input x. We also used that the uncertainties in training and testing data are independent.

We now incorporate the uncertainty of weight vector that defines the neural network. We can write

$$\begin{aligned} p(z|x,\theta_{tr},\theta_{te}) &= \int\int\int p\left(z|x^t,\theta^t_{tr},\theta^t_{te}\right)p\left(\theta^t_{tr}|\theta_{tr}\right)p\left(\theta^t_{te}|\theta_{te}\right)p(x^t|x)\,dx^t d\theta^t_{tr},d\theta^t_{te} \\ &= \int\int\int\int p\left(z,w|x^t,\theta^t_{tr},\theta^t_{te}\right)p\left(\theta^t_{tr}|\theta_{tr}\right)p\left(\theta^t_{te}|\theta_{te}\right)p(x^t|x)\,dx^t d\theta^t_{tr},d\theta^t_{te}dw. \end{aligned} \tag{10}$$

We now evaluate further

$$p\left(z,w|x^t,\theta^t_{tr},\theta^t_{te}\right) = p\left(z,|w,x^t,\theta^t_{tr},\theta^t_{te}\right)p\left(w|x^t,\theta^t_{tr},\theta^t_{te}\right) = p\left(z|w,x^t,\theta^t_{te}\right)p\left(w|\theta^t_{tr}\right) \tag{11}$$

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

where we used that if the weight vector is given, the training data do not provide extra information on z. Furthermore, the weight vector w by definition is only dependent on the training data and does not depend on the testing data and x^t.

Putting Eq. (11) in Eq. (10), we find:

$$p(z|x,\theta_{tr},\theta_{te}) = \int\int\int\int p\left(z|w,x^t,\theta^t_{te}\right)p\left(w|\theta^t_{tr}\right)p\left(\theta^t_{tr}|\theta_{tr}\right)p\left(\theta^t_{te}|\theta_{te}\right)p(x^t|x)\,dx^t d\theta^t_{tr},d\theta^t_{te}dw. \tag{12}$$

The right-hand side of Eq. (12) highlights the sources of the uncertainty in z, namely

1. Uncertainty in the new input, captured by $p(x_t|x)$,
2. Uncertainty in the data pairs in the training and testing sets, described by $p\left(\theta^t_{tr}|\theta_{tr}\right)$ and $p\left(\theta^t_{te}|\theta_{te}\right)$,
3. Uncertainty in the weights, represented by $p\left(w|\theta^t_{tr}\right)$,
4. Intrinsic uncertainty in the neural network, reflecting model imperfection, given by $p\left(z|w,x^t,\theta^t_{te}\right)$.

This equation can be interpreted as the *fundamental equation of machine learning*, as it provides a complete probabilistic characterization of a new output sample. Such characterization represents the ultimate goal of machine learning: not merely to predict a value, but to quantify the uncertainty in that prediction. It shifts the paradigm away from producing a single "best" estimate—an often ill-defined and insufficient approach for real-world applications—towards fully probabilistic descriptions, which are essential for informed decision-making and scientific rigor.

The only uncertainty not included is uncertainty in the neural network architecture. That uncertainty can be included in a manner similar to uncertainty in the weights, but a complication is that the dimension of the weight vector will change with architecture. This is not a fundamental problem, but requires careful consideration of probability spaces beyond the scope of this paper.

The next section discusses practical ways to estimate the four pdfs.

3. A practical method for uncertainty quantification

3.1. Source 1: Uncertainty in the new input

The influence of uncertainty in the new input vector x on the output z is the most straightforward to implement. One can draw N_x samples from the probability density function $p(x_t|x)$,which describes the uncertainty around the observed input vector x. Each of these sampled input vectors is then propagated through the neural network to generate corresponding samples of the output z. This Monte Carlo approach provides an empirical approximation of the distribution of z induced by the uncertainty in x. Mathematically, this means that we draw the samples from the uncertainty pdf, $p(x^t|x)$, as:

$$p(x^t|x) = \frac{1}{N_x}\sum_{i=1}^{N_x}\delta\left(x^t - x^t_i\right), \tag{13}$$

where x^t_i are the drawn samples centered around x. Using this expression, we can rewrite Eq. (12) as:

$$p(z|x,\theta_{tr},\theta_{te}) = \frac{1}{N_x}\sum_{i=1}^{N_x}\int\int\int p\left(z|w,x^t_i,\theta^t_{te}\right)p\left(w|\theta^t_{tr}\right)p\left(\theta^t_{tr}|\theta_{tr}\right)p\left(\theta^t_{te}|\theta_{te}\right)d\theta^t_{tr},d\theta^t_{te}dw. \tag{14}$$

3.2. Source 2: Uncertainty in the training and testing data

The uncertainty in the training and testing data can be treated in the same way as the uncertainty in the new input vector x. Similar to Eq. (13), we draw samples from $p\left(\theta^t_{tr}|\theta_{tr}\right)$ and $p\left(\theta^t_{te}|\theta_{te}\right)$ and write:

$$p\left(\theta^t_{tr}|\theta_{tr}\right) = \frac{1}{N_1}\sum_{j1=1}^{N_1}\delta\left(\theta^t_{tr} - \theta^t_{tr,j1}\right) \tag{15}$$

and similar for $p\left(\theta^t_{te}|\theta_{te}\right)$. This leads to:

$$p(z|x,\theta_{tr},\theta_{te}) = \frac{1}{N_x}\sum_{i=1}^{N_x}\frac{1}{N_{X1}}\sum_{j1=1}^{N_{X1}}\frac{1}{N_{X2}}\sum_{j2=1}^{N_{X2}}\int p(z|w,x^t_i,\theta^t_{te,j2})p(w|\theta^t_{tr,j1})\,dw, \tag{16}$$

where N_{X1} andN_{X2} are the number of samples drawn from the uncertainty pdf of the input–output pairs in the training and testing datasets, respectively. Note that we have N_{X1} samples for each training input–output pair, and N_{X2} samples for each testing input–output pair.

3.3. Source 3: uncertainty in the weights

3.3.1. Basic derivation

We now turn to the estimation of the weight distribution conditioned on the training data in Eq. (16), namely $p\left(w|\theta^t_{tr,j1}\right)$. As discussed in the Introduction, this pdf is inherently complex due to the highly nonlinear relationship between the neural network weights and the training data. Consequently, the exact form of this distribution is unknown, and direct sampling from it is not feasible.

Instead, we adopt a pragmatic approach inspired by standard practices in machine learning. In typical training routines, the weights are obtained through a minimization procedure (e.g., stochastic gradient descent), given a fixed neural network architecture and a predefined optimization algorithm. Under these conditions, the final weight vector is determined primarily by two stochastic components: the initialization of the weights and the order in which the training data batches are processed. By varying these components, we can generate multiple plausible realizations of the trained weights. These realizations serve as approximate samples from the unknown distribution $p\left(w|\theta^t_{tr,j1}\right)$, enabling a Monte Carlo approximation of the pdf. However, as explained below, considerable effort is needed to obtain a practical algorithm.

Let us first introduce the random batch order in our formalism and derive an expression for $p\left(w|\theta^t_{tr,j1}\right)$ that takes this uncertainty into account. By denoting the batch order b, we can write:

$$p\left(w|\theta^t_{tr,j1}\right) = \int p\left(w,b|\theta^t_{tr,j1}\right)db = \int p\left(w|b,\theta^t_{tr,j1}\right)p\left(b|\theta^t_{tr,j1}\right)db = \int p\left(w|b,\theta^t_{tr,j1}\right)p(b)\,db, \tag{17}$$

where we used the fact that the batch order is independent of the training data. By further employing Bayes Theorem in the form of Eq. (3), and writing the input–output pair of $\theta^t_{tr,j1}$ as $\left(X^t_{tr,j1},Z^t_{tr,j1}\right)$, we find:

$$p\left(w|b,X^t_{tr,j1},Z^t_{tr,j1}\right) = \frac{l\left(Z^t_{tr,j1}|w,b,X^t_{tr,j1}\right)}{p\left(Z^t_{tr,j1}|b,X^t_{tr,j1}\right)}p\left(w|b,X^t_{tr,j1}\right). \tag{18}$$

We can evaluate $p\left(w|b,X^t_{tr,j1}\right) = p(w)$, since the input data $X^t_{tr,j1}$ and the batch order b alone have no information on the weights w. Therefore, the conditional pdf is equal to the prior distribution over the weights.

The likelihood $l\left(Z^t_{tr,j1}|w,b,X^t_{tr,j1}\right)$ quantifies how well a particular weight vector w, when applied to the input data $X^t_{tr,j1}$ in the batch order b, reproduces the corresponding output data $Z^t_{tr,j1}$ of the training dataset. As explained in the Introduction, the denominator, $p\left(Z^t_{tr,j1}|b,X^t_{tr,j1}\right)$ serves as a normalization constant that ensures the posterior is a valid pdf. Since we are primarily interested in drawing samples from the posterior rather than evaluating it explicitly, this term does not need to be computed in practice.

Inserting Eq. (18) into Eq. (17) we find:

$$p\left(w|\theta^t_{tr,j1}\right) = \int \frac{l\left(Z^t_{tr,j1}|w,b,X^t_{tr,j1}\right)}{p\left(Z^t_{tr,j1}|b,X^t_{tr,j1}\right)}p(w)p(b)\,db \tag{19}$$

In the next section, we focus on determining the likelihood $l\left(Z^t_{tr,j1}|w,b,X^t_{tr,j1}\right)$ and show how to perform the integral.

3.3.2. The likelihood and the integral

To determine the likelihood, we use the relationship between the training pairs and the weight vectors as specified by the neural network:

$$Z^t_{tr,j1}=f\left(X^t_{tr,j1},w,b\right)+\epsilon\left(X^t_{tr,j1}\right), \tag{20}$$

where $\epsilon\left(X^t_{tr,j1}\right)$ is the difference between the predicted $f\left(X^t_{tr,j1},w,b\right)$ and the output of training $Z^t_{tr,j1}$. Because we already incorporated the uncertainty in the input and output training data above in Section 3.2, we can interpret this difference as the uncertainty in the neural network function itself. Rewriting this relation as $f\left(X^t_{tr,j1},w,b\right)=Z^t_{tr,j1}-\epsilon\left(X^t_{tr,j1}\right)$, and since $Z^t_{tr,j1}$ is given, the uncertainty distribution of $f\left(X^t_{tr,j1},w,b\right)$ is equal to that of $\epsilon\left(X^t_{tr,j1}\right)$ shifted by the constant vector $Z^t_{tr,j1}$.

As an example, assume the uncertainty in $\epsilon\left(X^t_{tr,j1}\right)$ to be Gaussian, with zero mean and error variance σ_f^2 for each element m of vector $\epsilon\left(X^t_{tr,j1}\right)$. We then find the likelihood:

$$l\left(Z^t_{tr,j1}|w,b,X^t_{tr,j1}\right)=A\exp\left[-\frac{1}{2}\sum_{m=1}^{N_{tr}}\frac{\left(Z^t_{m,tr,j1}-f\left(X_{m,tr,j1},w,b\right)\right)^2}{\sigma_f^2}\right], \tag{21}$$

where A is a normalization factor, N_{tr} is the total number of the input–output pairs in the training dataset. The term under the exponent can be recognized as a typical machine-learning loss function.

We now turn to the calculation of the integral in Eq. (19). We can draw initial weight vector samples w_k from $p(w)$ and batch order samples b_k from $p(b)$, which leads to:

$$p\left(w|\theta^t_{tr,j1}\right)=\frac{1}{N_w}\sum_{k=1}^{N_w}\frac{l\left(Z^t_{tr,j1}|w_k,b_k,X^t_{tr,j1}\right)}{p\left(Z^t_{tr,j1}|b_k,X^t_{tr,j1}\right)}\delta(w-w_k), \tag{22}$$

where N_w is the total number of drawn weight vector samples. This expression shows that the posterior pdf of the weight vectors is represented by the initial weight vectors w_k, each with a so-called importance weight:

$$\alpha_{j1,k}=\frac{1}{N_w}\frac{l\left(Z^t_{tr,j1}|w_k,b_k,X^t_{tr,j1}\right)}{p\left(Z^t_{tr,j1}|b_k,X^t_{tr,j1}\right)} \tag{23}$$

This implies that any posterior moment of the weight vector w (e.g., the mean or covariance) must be computed using importance weights $\alpha_{j1,k}$. However, in practice, the values of $\alpha_{j1,k}$ tend to vary dramatically between samples and, typically, one of them takes a value close to 1, while the others are nearly zero. This is because the likelihood contains an exponential term, as shown in Eq. (21). The sum in the brackets varies considerably with the weight vector w_k, and the exponent of that sum will enhance that variation substantially. As a result, the posterior distribution becomes sharply peaked, and only a single weight vector significantly contributes to the estimate of $p\left(w|X^t_{tr,j1},Z^t_{tr,j1}\right)$. The effective ensemble size collapses to one, eliminating any representation of uncertainty. This issue is known as filter degeneracy in the particle filter literature (Doucet et al., 2001; Van Leeuwen, 2019; Evensen et al., 2022). It renders the use of Eqs. (22) and (23) impractical, as they fail to maintain a diverse and informative ensemble of weight vectors.

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

3.3.3. The proposal density

Unlike Eq. (22), where samples of the weight vector w are drawn from the prior distribution $p(w)$, we can instead draw samples from a distribution of trained weights. This approach is mathematically formalized by introducing a *proposal density* $q\left(w|X^t_{tr,j1},Z^t_{tr,j1}\right)$ which explicitly depends on the training data (Doucet et al., 2001; Van Leeuwen, 2010; Ades and Van Leeuwen, 2012). To incorporate this, we multiply and divide the prior pdf $p(w)$ by the proposal density $q\left(w|X^t_{tr,j1},Z^t_{tr,j1}\right)$. This procedure will allow us to perform importance sampling using samples drawn from this proposal rather than from the $p(w)$. Using this idea in Eq. (19), we find:

$$\begin{aligned} p\left(w|\theta^t_{tr,j1}\right) &= \int \frac{l\left(Z^t_{tr,j1}|w,b,X^t_{tr,j1}\right)}{p\left(Z^t_{tr,j1}|b,X^t_{tr,j1}\right)} p(w)p(b)\,db \\ &= \int \frac{l\left(Z^t_{tr,j1}|w,b,X^t_{tr,j1}\right)}{p\left(Z^t_{tr,j1}|b,X^t_{tr,j1}\right)} \frac{p(w)}{q\left(w|b,X^t_{tr,j1},Z^t_{tr,j1}\right)} q\left(w|b,X^t_{tr,j1},Z^t_{tr,j1}\right)p(b)\,db \\ &= \frac{1}{N_w}\sum_{k=1}^{N_w} \frac{l\left(Z^t_{tr,j1}|w_{j1,k},b_k,X^t_{tr,j1}\right)}{p\left(Z^t_{tr,j1}|b_k,X^t_{tr,j1}\right)} \frac{p\left(w_{j1,k}\right)}{q\left(w_{j1,k}|b_k,X^t_{tr,j1},Z^t_{tr,j1}\right)} \delta\left(w-w_{j1,k}\right), \end{aligned} \tag{24}$$

where the weight vectors $w_{j1,k}$ are now drawn from the proposal density $q\left(w|b_k,X^t_{tr,j1},Z^t_{tr,j1}\right)$. The conditioning on the training data means that the weight vectors have been trained on the training data. The details on how they are trained are described in the proposal density.

A standard choice in machine learning is to draw the weight vectors by minimizing a loss function, which is the same as maximizing the likelihood. This minimization is combined with a methodology to avoid overfitting. It is not easy to write down the proposal density in this case. No matter these details, from Eq. (24) we see that $p\left(w|\theta^t_{tr,j1}\right)$ is now represented by a set of weights $w_{j1,k}$, each with importance weight given as:

$$\beta_{j1,k} = \frac{1}{N_w} \frac{l\left(Z^t_{tr,j1}|w_{j1,k},b_k,X^t_{tr,j1}\right)}{p\left(Z^t_{tr,j1}|b_k,X^t_{tr,j1}\right)} \frac{p\left(w_{j1,k}\right)}{q\left(w_{j1,k}|X^t_{tr,j1},Z^t_{tr,j1}\right)}. \tag{25}$$

The $\beta_{j1,k}$ consist of three factors that depend on the value of the weight vector, the prior pdf, the proposal density, and the likelihood. The prior $p(w)$ is typically a uniform distribution around zero, i.e., the probability $p\left(w_{j1,k}\right)$ is the same for any drawn weight vector $w_{j1,k}$. The proposal density $q\left(w|b_k,X^t_{tr,j1},Z^t_{tr,j1}\right)$ does depend on the trained weight vector. To see this, denote a prior weight vector sample as u_k and the corresponding trained weight vector as $w_{j1,k}$. The proposal density is related to the prior density via the standard transformation of variables formula from statistics, as:

$$q\left(w_{j1,k}|b_k,X^t_{tr,j1},Z^t_{tr,j1}\right) = \left|\frac{\partial w_{j1,k}}{\partial u_k}\right|_{b_k} p(u_k), \tag{26}$$

where $\left|\partial w_{j1,k}/\partial u_k\right|_{b_k}$ is the Jacobian of this transformation. This Jacobian is a complicated function and depends on u_k and b_k. Although different prior samples u_k will have the same value of $p(u_k)$, the Jacobians are different for different u_k and b_k, and, consequently, the proposal densities of the trained weights will be different from each other. The likelihood values of the trained weights will also be different because some trained weight vectors will fit the training data better than others. This means that the importance weights $\beta_{j1,k}$ will be different for each weight vector. It is well known from the particle filtering literature that if the training data set is large, the differences in the $\beta_{j1,k}$ will be huge, again leading to one importance weight having a value close to 1, while all others have values very close to zero.

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

Since Bagging, Deep Ensembles, and MC dropout use the trained weights in this way, they are of little value for providing information on $p\left(w|\theta^t_{tr,j1}\right)$. However, we can construct a proposal density that does not have this problem, as explained below.

3.3.4. A new proposal density

To overcome the filter degeneracy issue, we design the proposal density $q\left(w|X^t_{tr,j1}, Z^t_{tr,j1}\right)$ such that all importance weights $\beta_{j1,k}$ are equal, or at least nearly so. One intuitive approach is to enforce a fixed likelihood value across all sampled weights. Specifically, we define a target likelihood value l_0 and seek intermediate weight vectors $\tilde{w}_{j1,k}$ that satisfy the identity:

$$l\left(Z^t_{tr,j1}|\tilde{w}_{j1,k}, b_k, X^t_{tr,j1}\right) = l_0. \tag{27}$$

To ensure that the $\beta_{j1,k}$ are equal, all values $q\left(\tilde{w}|X^t_{tr,j1}, Z^t_{tr,j1}\right)$ need to be equal as well. According to Eq. (26) this implies that the Jacobian of the transformation from the untrained weights u_k to the trained weights $\tilde{w}_{j1,k}$ must not depend on k. However, variations in u_k that change $\tilde{w}_{j1,k}$ such that $l\left(Z^t_{tr,j1}|\tilde{w}_{j1,k}, b_k, X^t_{tr,j1}\right)$ increases or decreases are not allowed, because the likelihood has to be fixed at l_0. This means that the matrix $\left(\partial \tilde{w}_{j1,k}/\partial u_k\right)$ is not full rank, and its determinant vanishes:

$$\left|\frac{\partial \tilde{w}_{j1,k}}{\partial u_k}\right| = 0 \tag{28}$$

resulting in a proposal density $q\left(\tilde{w}|X^t_{tr,j1}, Z^t_{tr,j1}\right)$ that is not well defined. As a result, while this method is conceptually appealing, it is not mathematically viable for practical use.

To avoid this issue, we define the final weight vectors in the proposal density by adding a small perturbation of size δ to each intermediate weight vector $\tilde{w}_{j1,k}$, as:

$$w_{j1,k} = \tilde{w}_{j1,k} + \delta r_{j1,k}(u_k), \tag{29}$$

where $r_{j1,k}(u_k)$ is a function of the untrained weight vector u_k. The Jacobian can now be evaluated via a Taylor-series expansion around $\left|\partial w_{j1,k}/\partial u_k\right|$, and, since δ is small, we only need to retain up to the first term in the expansion, leading to:

$$\left|\frac{\partial w_{j1,k}}{\partial u_k}\right| = \left|\frac{\partial \tilde{w}_{j1,k}}{\partial u_k} + \delta \frac{\partial r_{j1,k}}{\partial u_k}\right| \approx \left|\frac{\partial \tilde{w}_{j1,k}}{\partial u_k}\right| + \delta Tr\left[Adj\left(\frac{\partial \tilde{w}_{j1,k}}{\partial u_k}\right)\left(\frac{\partial r_{j1,k}}{\partial u_k}\right)\right] + O\left(\delta^2\right), \tag{30}$$

where Tr denotes the trace, and Adj denotes the adjugate. If we now choose $\left(\partial r_{j1,k}/\partial u_k\right)$ as the pseudoinverse $\left(\partial r_{j1,k}/\partial u_k\right) = Adj\left(\frac{\partial \tilde{w}_{j1,k}}{\partial u_k}\right)^{\dagger}$ we find:

$$\left|\frac{\partial w_{j1,k}}{\partial u_k}\right| = \delta n + O\left(\delta^2\right), \tag{31}$$

where $n = \dim\left(w_{j1,k}\right)$, and we used that $\left|\partial \tilde{w}_{j1,k}/\partial u_k\right| = 0$. This shows that, up to order δ, the Jacobian is a constant, independent of the weight vector $w_{j1,k}$. This, then, means that the proposal density $q\left(w_{j1,k}|X^t_{tr,j1}, Z^t_{tr,j1}\right)$ in Eq. (26) has the same value for each weight vector $w_{j1,k}$.

However, by adding a small perturbation to the trained weight vectors, the likelihood is no longer exactly equal to l_0. To assess the impact of this perturbation, we perform a Taylor series expansion on the likelihood around the trained weights. This yields:

$$l\left(Z^t_{tr,j1}|w_{j1,k}, b_k, X^t_{tr,j1}\right) = l_0 + \delta r_{j1,k} \nabla_{w_{j1,k}} l + O\left(\delta^2\right), \tag{32}$$

in which $\delta r_{j1,k}$ is the pertubation. Since the weights $\tilde{w}_{j1,k}$ have been obtained through training, the likelihood is already near its maximum, implying that the gradient $\nabla_{w_{j1,k}} l$ is small. By choosing the perturbation

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

magnitude δ to be sufficiently small, the first-order term becomes negligible, and the perturbed likelihood remains very close to l_0. This approximation restores the practical viability of the method.

The target value for the likelihood l_0 is determined as follows. First, we train the neural network with the unperturbed training data, resulting in a weight vector denoted by w_0. Next, we use the N_{X1} perturbed versions of the training data and pass each of them through the neural network using this fixed weight vector w_0. This process yields N_{X1} loss function values $J_{j1}\left(Z^t_{tr,j1}, X^t_{tr,j1}, w_0\right)$. We average these as $J_0 = 1/N_{X1}\sum_{j1=1}^{N_{X1}} J_{j1}\left(Z^t_{tr,j1}, X^t_{tr,j1}, w_0\right)$ and take as target likelihood value $l_0 = \exp(-J_0)$. By defining l_0 this way, we obtain a representative likelihood value for trained weight vectors. Consequently, the set of trained weight vectors obtained from the perturbed training datasets can be interpreted as an unbiased sample from the posterior distribution of the weights, $p\left(w|X^t_{tr,j}, Z^t_{tr,j1}\right)$.

With the above construction, where each weight vector $w_{j1,k} \approx \tilde{w}_{j1,k}$, we have ensured that the proposal densities $q\left(w_{j1,k}|X^t_{tr,j1}, Z^t_{tr,j1}\right)$ are identical across all samples. Additionally, the corresponding likelihoods $l\left(Z^t_{tr,j1}|w_{j1,k}, b_k, X^t_{tr,j1}\right)$ are the same for each weight vector. As a result, all importance weights $\beta_{j1,k}$ are equal, implying that each weight vector $w_{j1,k}$ is equally probable in the posterior distribution $p\left(w|X^t_{tr,j}, Z^t_{tr,j1}\right)$.

With these equally probable samples from $p\left(w_{j1,k}|X^t_{tr,j1}, Z^t_{tr,j1}\right)$, Eq. (16) can be rewritten as:

$$p(z|x,\theta_{tr},\theta_{te}) = \frac{1}{N_x}\sum_{i=1}^{N_x}\frac{1}{N_{X1}}\sum_{j1=1}^{N_{X1}}\frac{1}{N_{X2}}\sum_{j2=1}^{N_{X2}}\frac{1}{N_w}\sum_{k=1}^{N_w} p\left(z|w_{j1,k}, x^t_i, X^t_{te,j2}, Z^t_{te,j2}\right). \tag{33}$$

3.4. Uncertainty in the trained neural network

As shown in Eq. (33), the final step is to estimate $p\left(z|w_{j1,k}, x^t_i, X^t_{te,j2}, Z^t_{te,j2}\right)$, which captures the uncertainty due to the neural network's imperfection. This imperfection is evaluated using the testing data $\left(X^t_{te,j2}, Z^t_{te,j2}\right)$ for each index $j2$. Since the evaluation is based on the statistical relation described by the input–output pairs of testing data, we rewrite

$$p\left(z|w_{j1,k}, x^t_i, X^t_{te,j2}, Z^t_{te,j2}\right) = p_{\theta_{te,j2,i}}\left(z|w_{j1,k}, x^t_i\right), \tag{34}$$

i.e., the pdf of z given the input x^t_i and weight vector $w_{j1,k}$, determined using the input–output pairs in the testing data $j2$. The extra index i in the testing data denotes that we only use testing data for which the input data is close to x^t_i. Specifically, we calculate the pdf of the output testing data $Z_{te,j2}$ for which the input testing data is within a region around the new input, so $X^t_{tr,j2} \in D_{x^t_i}$. This pdf can be approximated by known functions (e.g., a Gaussian or a Gaussian mixture), by a histogram, by the set of samples $Z_{te,j2,m}$, or by new samples generated from $Z_{te,j2,m}$ (e.g., via a so-called diffusion process (Sohl-Dickstein et al., 2015)).

The size of $D_{x^t_i}$ is governed by the uncertainty in the input testing data. In our experience, setting the radius of $D_{x^t_i}$ between 0.5 and 2 times the uncertainty standard deviation in the input testing data yields robust and consistent output pdf estimates.

3.5. The resulting algorithm and practical considerations

Using Eq. (34) into Eq. (33), we have a complete uncertainty quantification methodology that can be written as:

$$p(z|x,\theta_{tr},\theta_{te}) = \frac{1}{N_x}\sum_{i=1}^{N_x}\frac{1}{N_{X1}}\sum_{j1=1}^{N_{X1}}\frac{1}{N_{X2}}\sum_{j2=1}^{N_{X2}}\frac{1}{N_w}\sum_{k=1}^{N_w} p_{\theta_{te,j2,i}}\left(z|w_{j1,k}, x^t_i\right). \tag{35}$$

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

This equation shows that we average the pdfs $p_{\theta_{te,j2,i}}\left(z|w_{j1,k},x_i^t\right)$ over the input samples i, the trained weight vector samples k, the training data samples $j1$, and the testing data samples $j2$, to find the full uncertainty pdf of the output of the deep learning procedure. The following provides the set of steps needed in this calculation:

1) **Initial training**: Train a neural network using the original, unperturbed training data. Apply any standard technique to avoid overfitting, but do not use dropout. Store the resulting weight vector as w_0.
2) **Generate perturbed training data**: Create an ensemble of N_{X1} training datasets drawn from the uncertainty distributions of the input and output training datasets. This results in N_{X1} perturbed versions of the full training dataset.
3) **Determine the target likelihood value** l_0: Use the perturbed input training datasets from step 2 to calculate their loss function values using the corresponding perturbed training output dataset and the fixed weight vector w_0 from step 1. Average the N_{X1} loss function values and store the result in J_0. Calculate the target likelihood value $l_0 = \exp(-J_0)$.
4) **Determine an ensemble of weight vectors**: For each perturbed training dataset, train N_w neural networks with different initial untrained weight vectors and batch orders. This leads to $N_{X1} \times N_w$ different neural networks. These networks should all be trained such that their loss function value is close to the target loss function value J_0. This process results in $N_{X1} \times N_w$ trained weight vectors.
5) **Generate perturbed new input data**: Create N_x perturbed new input values drawn from the uncertainty distribution of the new input value. This results in N_x perturbed versions of the new input value.
6) **Generate perturbed testing data**: Create N_{X2} perturbed testing datasets drawn from the uncertainty distribution of the testing data. This results in N_{X2} perturbed versions of the testing data.
7) **Generate samples for the pdf**: For each perturbed new input value from step 5, find all the perturbed testing input data from step 6 that are close to it. Pass the selected testing input data through the $N_{X1} \times N_w$ neural networks from step 4 and subtract the result from their corresponding testing output data. This results in a large number of output samples for each perturbed new input data.
8) **Determine final pdf**: Calculate histograms for each perturbed new input data based on the samples calculated in step 7. Average the histograms to find the final uncertainty pdf for output z.

3.6. Practical tips

1. Ensemble sizes. We tested our algorithm on several test cases, including a real-world application described below. In our experience, the predictive pdf converged for $N_x = N_{tr} = N_{te} = N_w = 20$ in all applications. However, we expect these numbers to be application dependent.
2. Accuracy of likelihood calculations. The likelihood is calculated from the loss function by multiplying it by minus one and then taking the exponential. When a large amount of training data is used, the loss function values will be very large, and taking the exponent of minus that large number can lead to underflow issues. Since we are only interested in relative likelihood values, we can subtract the same constant value from each loss function before taking the exponent. In our algorithm, we used the J_0 value.
3. Simplifications for Gaussian output uncertainties. The above algorithm can be simplified significantly if $l\left(Z_{tr,j1}^t|w_{j1,k},b_k,X_{tr,j1}^t\right)$ is assumed to be Gaussian as in Eq. (21), and the uncertainty in the output training data Z_{tr} is also Gaussian distributed. In that case, we do not need to introduce perturbed output training data Z_{tr}^t in Step 2. Instead, we can work directly with $p\left(w|X_{tr,j1}^t,Z_{tr}\right)$. Then the likelihood of the weight vectors becomes (see Appendix A):

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

$$l\left(Z^t_{tr,j1}|w,b,X^t_{tr,j1}\right) = A\exp\left[-\frac{1}{2}\sum_{m=1}^{N_{tr}}\frac{\left(Z^t_{tr,m} - f\left(X_{tr,j1,m},w,b\right)\right)^2}{\sigma_f^2 + \sigma_{z,m}^2}\right], \tag{36}$$

in which $\sigma^2_{z,m}$ is the uncertainty variance in the output training data $Z_{tr,m}$. All steps and equations from Eq. (21) to Eq. (33) remain unchanged, with the only modification being that the perturbed output training vector $Z^t_{tr,j1}$ is replaced everywhere with the original, unperturbed output vector Z_{tr}. This is a situation often encountered in the geosciences.

4. Comparison of methodologies on a simple regression problem

To demonstrate how the uncertainty characterized by our method differs from existing approaches, we compare results on a simple regression problem with the Bagging ensemble method by Breiman (1996) and the pdf-based methodology proposed by Wang et al. (2016). Our implementation of the pdf-based method is quantile regression.

The training and testing data sets are generated from the following system

$$z = ax^2 + bx + c \tag{37}$$

in which $a = 0.5$, $b = 2.0$, and $c = 5$. A total of 2000 input–output samples are generated, with inputs drawn from a normal distribution with a mean of 2 and a standard deviation of 2. Both the input and output variables are perturbed by additive Gaussian noise with a standard deviation of 0.3. The full dataset is shown in Figure 1. The 2000 input–output samples are divided into 1600 training and 400 testing samples. All samples are displayed in Figure 1.

Based on the relationship observed in Figure 1, or guided by prior knowledge of the underlying system, one might select the following neural network configuration to represent the system behavior:

$$z = dx^4 + ex + f, \tag{38}$$

In this example, the neural network weight vector is $w = (d, e, f)^T$.

The orange curve in Figure 1 illustrates the machine-learned solution based on the unperturbed training data. While the model fits well for positive input values, it underperforms for inputs smaller than −2. This

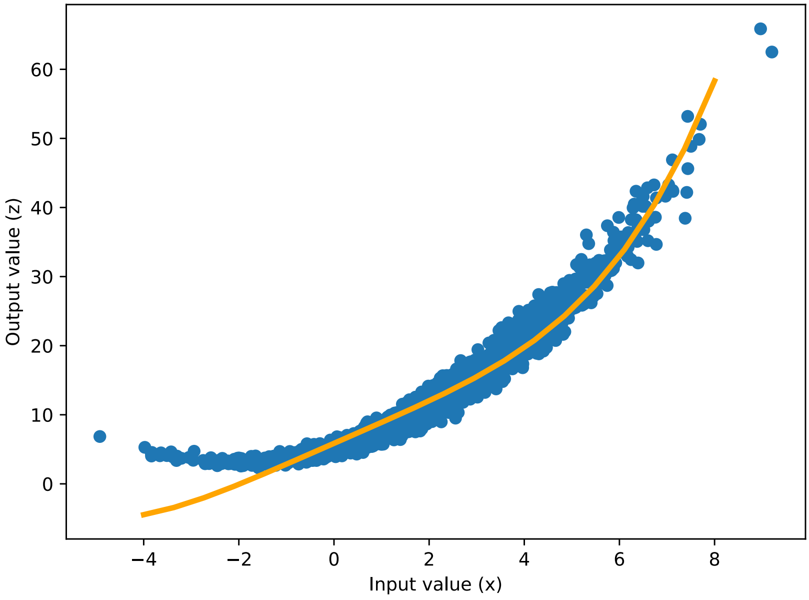

Figure 1. *Training and testing data (blue dots) and example model (orange line).*

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

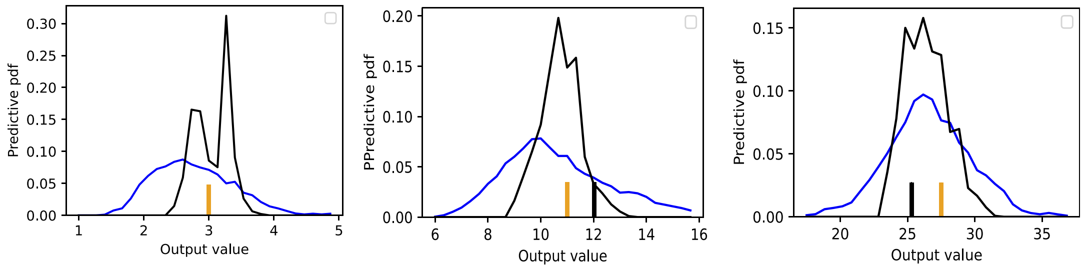

Figure 2. *Posterior pdfs derived from Bagging (black vertical line), quantile regression (black curve), and new methodology (blue line), for input values* -2 *(left), 2 (middle), and 5(right). The pdfs can be compared with the samples in Figure 1. The orange vertical line is the prediction using the true model. (Note that this not the true prediction because the input value has uncertainty.) In the left panel the Bagging vertical lines fall around* -0.07*, outside the plot range. Bagging shows a degenerate pdf, and the quantile regression pdf is too narrow because uncertainties in training, testing, and new input data are ignored.*

outcome illustrates a common issue: even when a model fits the training data adequately, its structural constraints can restrict its ability to generalize across the full input domain.

Figure 2 presents the results for three new input values, specifically $x = -2$, $x = 2$, and $x = 5$. As a reference, using these new inputs in conjunction with the true equation (37), we compute the corresponding 'true' output values. These values are shown as orange vertical bars in Figure 2, under the assumption that the input values are known precisely and have no uncertainty.

The results from the Bagging method are shown by the black bars in Figure 2. In the left panel, the Bagging vertical lines fall around -0.07, outside the plot range. These bars represent the output values from an ensemble of 20 neural networks. Due to the high similarity among the ensemble members, the individual predictions are nearly indistinguishable and collectively appear as a single thick black bar. As discussed in earlier sections, this outcome reflects a fundamental limitation of the Bagging approach. Variations in initial weights or the order of training batches have only a minimal effect on the final trained weights in this simple regression problem. Consequently, all networks converge to similar solutions, leading to a significant underestimation of the associated uncertainty. This limitation is concerning, because in more complex real-world applications, Bagging may similarly fail to capture meaningful uncertainty, potentially giving a false sense of confidence in the predictions. Another fundamental issue is the bias displayed by the Bagging procedure. The values follow the orange curve in Figure 1 quite closely, so a bias in the neural network becomes a bias in Bagging. The reason the new method and quantile regression do not have this bias problem is that they use the testing samples to define the pdfs, not only the weight values. MC dropout would show the same issue as bagging, while Deep Ensembles, based on quantile regression pdfs, would not have this strong bias.

Quantile regression estimates the linear relationship between inputs and outputs at specified quantiles. Given the nonlinear nature of the system, as illustrated in Figure 1, a linear fit over all output values for each quantile is clearly inappropriate. To address this, the quantile regression is restricted to an interval around the new test input value. For the results shown in Figure 2, we used an interval spanning 10 standard deviations of the input error. Alternative interval sizes were tested and yielded comparable outcomes. As shown in Figure 2, quantile regression (black curves) results in uncertainty estimates that are too narrow. This is expected because uncertainties in training, testing, and new input data are neglected.

Compared to the existing methods, the new method provides an uncertainty estimate that is broader than that of quantile regression. To see that it is more realistic, one can add typical input uncertainties of 0.3 to the new input values, and study the output spread in Figure 1. The proof that the new method provides the correct uncertainty estimate follows directly from its formulation: it takes all uncertainty

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

sources properly into account. The only approximation is in the size of the ensembles used to characterize the uncertainties in the individual sources. Increasing these sizes by a factor 10 leads to the same visual pdfs as displayed in Figure 2, strongly suggesting that the method has converged.

We can conclude that none of the existing uncertainty quantification methods provides a proper uncertainty estimate in this simple system. In practical applications, hyperparameters in these existing methods are often tuned to produce visually plausible uncertainty intervals, but such tuning lacks generalizability and must be repeated for each new application. The proposed method avoids this limitation and provides consistent uncertainty estimates.

5. Comparison of methodologies on predictions of cloud process rate

In this section, we implement our methodology to quantify the uncertainty associated with autoconversion rates. Autoconversion is a fundamental microphysical process that governs the formation of raindrops through cloud droplet collision and coalescence. Due to its inherently stochastic and nonlinear nature, autoconversion is well-suited for machine learning based prediction and uncertainty quantification (Chiu et al., 2021; Gettelman et al., 2021).

5.1. Description of the training and testing datasets

In this study, we leverage the training and testing datasets used in Chiu et al. (2021). The input variables are derived from aircraft cloud probe measurements collected during the Aerosol and Cloud Experiments in the Eastern North Atlantic (ACE-ENA) field campaign, operated by the Atmospheric Radiation Measurement (ARM) user facility. These include cloud water content (q_c), cloud droplet number concentration (N_c), drizzle water content (q_r), and drizzle drop number concentration (N_r), with respective uncertainties of 30%, 50%, 30%, and 20% (Glienke and Mei, 2019, 2020; Mei et al., 2020). The output variable, autoconversion rate, is calculated by applying the in-situ cloud droplet size distribution to the stochastic collection equation formulated as a two-moment bin model. The rate spans a wide dynamic range, from 7.7×10^{-24} to $2.8 \times 10^{-6}\ kg\,\mathrm{m}^3\mathrm{s}^{-1}$, capturing diverse stages of cloud evolution and supporting robust training. The uncertainty in the autoconversion rate is approximately 53%, based on the analytical autoconversion equation (6) from Chiu et al. (2021). The full dataset, consisting of approximately 10,612,000 samples, is partitioned into training, validation, and testing subsets, with a 60–20-20% data split. To reduce dynamic range and satisfy the assumption of additive Gaussian noise, a base-10 logarithmic transformation is applied to all input and output variables.

5.2. Training procedure

Building on the neural network architecture used by Chiu et al. (2021), we implement a feedforward network comprising six fully connected hidden layers, each containing 16 nodes. Following the procedure outlined in Section 3.5, we first trained the network using the unperturbed training dataset, with randomly initialized weights and a shuffled batch order. To mitigate overfitting, we applied the early stopping criterion proposed by Prechelt (1998), halting training when the validation loss no longer decreased with additional epochs. Given the high variability of the validation loss across epochs, we smoothed the loss curve using a moving average over a 50-epoch window. We then identified the epoch at which the smoothed loss stabilized, and used the corresponding model weights as the final trained weights, w_0.

Following the initial training with unperturbed data, we then trained separate neural networks for each of the 20 perturbed training datasets $X_{tr,j1}, Z_{tr,j1}$, using 20 different sets of perturbed initial weights and batch orders. This yielded $N_w = 20$ models per perturbed dataset. Training was stopped when the loss function for the training data reached J_0. As a result, we obtained a total of 400 neural networks (20 networks for each of the 20 perturbed datasets) to quantify uncertainty in both the model weights and the training data.

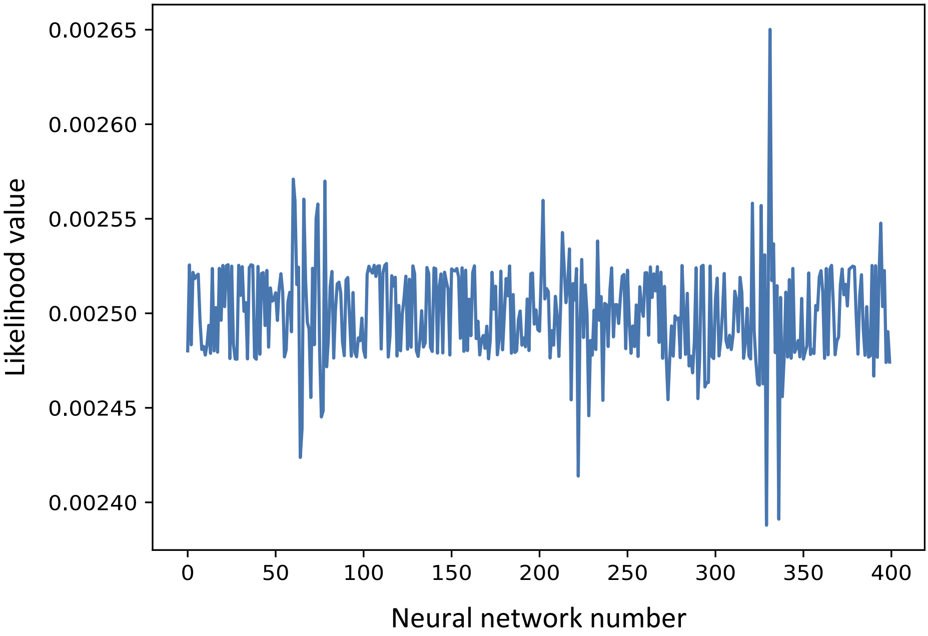

Figure 3. *The normalized likelihood values (i.e., the importance weights) for the* 400 *neural networks.*

In practice, exact convergence to J_0 is complicated by the fact that many machine learning frameworks report loss only at the end of each epoch. To address this, we identified the two epochs where the loss values bracketed J_0, and then applied a bisection method to approximate a loss value within 0.01 of J_0. This threshold ensures that the likelihood values are sufficiently close to $l_0 = \exp(-J_0)$, leading to approximately uniform importance weights across the ensemble. As shown in Figure 3, most importance weights are close to the expected value of $1/400 = 0.0025$, which corresponds to a perfect equal-weight ensemble. A small number of networks did not converge precisely, which is expected due to the high dimensionality of the optimization space. For these networks, training was stopped at slightly less precise loss values, resulting in importance weights deviating by no more than 6%, which is acceptable for our purposes.

5.3. Results

To evaluate our methodology on new input data, we selected 50 distinct input vectors spanning a wide range of values. Each input vector was perturbed 100 times using random noise sampled from the input uncertainty distributions described in Section 4.1. This resulted in 40,000 output samples per input vector ($N_x = 100, N_w = 20, N_{X1} = 20$). For each input, we computed the predictive uncertainty distribution using the testing data, as described in Eq. (29), and averaged the resulting histograms.

As mentioned in Section 3.4, the region ($D_{x_i^t}$) around each perturbed new input x_i^t, must be defined to compute the local empirical distribution from the test data. We used all output testing data that had corresponding input testing data close to x_i^t within 0.2 (in $\log 10$ space), for each of the four variables in the input vector. We experimented with thresholds ranging from 0.1 to 0.5 and found that this choice had no significant visual impact on the resulting distributions.

The results for all 50 input vectors are displayed in Appendix B. To highlight the key findings, Figure 4 shows the total uncertainty pdfs as a function of the $\log 10$ autoconversion rate for four representative input vectors. In each panel, the blue curve is the total uncertainty pdf from Eq. (30), and the red bar is the point prediction from the baseline neural network (i.e., using w_0) without uncertainty quantification. The black bars show the predictions from Bagging, obtained by creating an ensemble of 20 networks with different initial weights and batch orders.

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

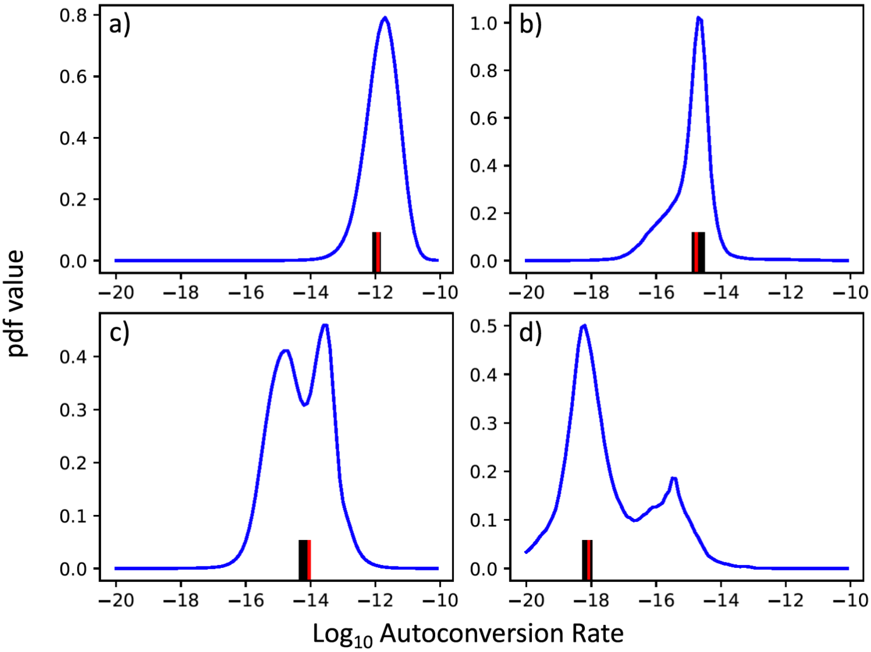

Figure 4. *Examples of total uncertainty pdf in the output autoconversion rate for four input vectors. The blue curves are the total uncertainty pdfs; the red bar is the autoconversion rate calculated by the baseline neural network (w_0) without uncertainty quantification; black bars are the Bagging output samples. Note the wide variety of shapes of the blue uncertainty pdfs and the small spread in the black bars, demonstrating the inadequacy of the Bagging approach to represent uncertainty.*

Figures 4(a) and (b) show that the predictive pdfs span roughly an order of magnitude in autoconversion rate, which is not surprising given the relatively large measurement uncertainty in the input variables (ranging between 20% and 50%). Although the pdf in Figure 4(a) can be approximated by a Gaussian distribution, a non-negligible skewness is observed in Figure 4(b). The pdfs in Figure 4(c) and (d) are broader, showing large uncertainty and bimodality.

To disentangle the contributions of different uncertainty sources, we first isolated the effect of neural network weight uncertainty. This was done by disabling all other uncertainty sources in Eq. (30) and computing outputs using only the ensemble of weight vectors. The resulting distributions were very narrow and closely resembled the Bagging ensemble outputs (black bars), indicating that weight uncertainty is not a major contributor.

Next, we assessed the combined impact of input uncertainty and model uncertainty from the testing data. This involved evaluating the corresponding terms in Eq. (30) while ignoring the other uncertainties. These two sources contributed similar amounts of variability and together accounted for most of the spread in the predictive distribution, as shown by the black curves in Figure 5, which peak at similar values to the full pdfs (blue curves). However, the combined input and model uncertainty distributions were typically unimodal and symmetric (in $\log_{10}$ space), failing to reproduce the skewness or bimodality observed in the full predictive distributions. This suggests that the observed skewness and bimodality are primarily caused by uncertainty in the training and testing datasets.

To further investigate the skewness and bimodality in the predictive pdfs, we analyzed all 50 cases in Appendix B. We found that positive skewness occurred predominantly at high autoconversion values (above $10-12$), while strong negative skewness was associated with intermediate values ($14-17$). Bimodal distributions were mainly observed at low autoconversion values (around -18). Attempts to relate these features to specific patterns in the 4D input space were inconclusive, and even 3D visualizations of input subsets failed to reveal consistent structure.

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

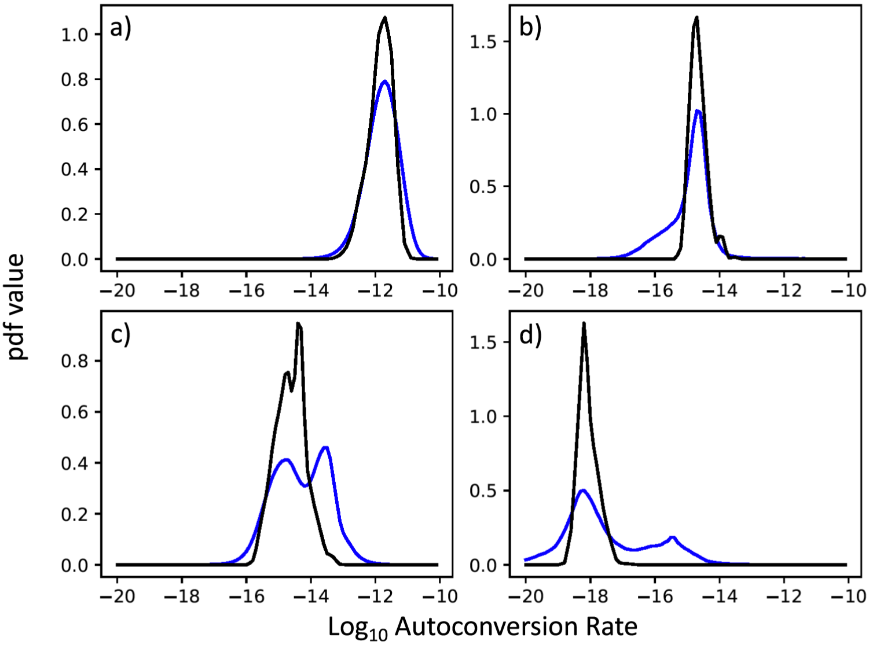

Figure 5. *Same as Figure 4, showing full uncertainty pdfs (blue curves) and the contribution from input uncertainty and model uncertainty (black curves).*

Returning to Figure 4, we observe the limitations of Bagging. The resulting ensemble predictions (black bars) exhibit little spread and fail to capture features such as bimodality, leading to significant underestimation of uncertainty. In fact, the shortcomings of Bagging extend beyond this. As noted in Section 3.3 (following Eq. 21), ensemble methods like Bagging, Deep Ensembles, and MC Dropout implicitly assume equal likelihood for all network realizations. However, in practice, the importance weights of these networks, derived from their likelihood values, vary exponentially.

To illustrate this further, we normalized all likelihoods by the largest value. This resulted in one network receiving an importance weight of 1, while the next highest weight was $\exp(-824)$, effectively zero. All remaining networks contributed negligibly. Thus, only the highest-likelihood network meaningfully affects the predictive distribution, rendering the ensemble effectively a single-model prediction. This outcome stands in stark contrast to our method, which produces near-uniform importance weights (see Figure 3). Accordingly, if standard ensemble-based methods were applied in a statistically consistent manner, they would collapse to a single model and fail to provide meaningful uncertainty quantification.

6. Conclusions and discussion

Quantifying uncertainty in deep learning results is essential for their scientific application and for achieving broader acceptance of machine learning methodologies in the geosciences. In this study, we reviewed the existing literature on uncertainty quantification and highlighted common shortcuts, approximations, and sources of uncertainty that are often overlooked in current approaches. Given the limitations and inconsistencies in these methods, we developed a comprehensive and systematic framework for quantifying output uncertainty that accounts for all major sources:

1. uncertainty in the input variables,
2. uncertainty in the training and testing data,
3. uncertainty in the neural network weight vectors (arising from variations in initial weights and batch order), and

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

4. uncertainty inherent to the neural network itself, as captured by its behavior on the testing dataset.

Our methodology is general and makes no assumptions about the relative importance of these uncertainty sources. It provides a scientifically rigorous foundation for machine learning applications and is expected to support the adoption of such methods in geoscientific research.

We demonstrated the practical implementation of this approach using a simple regression example and a real-world example: predicting cloud autoconversion rates with a deep learning model trained on a two-moment bin formulation of the stochastic collection equation and aircraft cloud probe measurements from the ACE-ENA field campaign, operated by the Atmospheric Radiation Measurement (ARM) user facility. Our analysis revealed that the dominant source of predictive uncertainty in this case stems from the uncertainty in the training and testing datasets, contributing to variations in output of about an order of magnitude. The uncertainty from the new input data and from the neural network itself, as determined by the testing data, also made significant contributions, roughly half that magnitude. In contrast, uncertainty due to initial weight values and training batch order was found to be comparatively minor.

An important question is how our methodology scales with the size of the deep-learning problem. From a practical viewpoint, similar ensemble sizes were needed in the simple and the real-world example, suggesting a scaling independent of system size. However, our real-world example is relatively small, so our findings might not generalize. Nevertheless, we know from numerical weather prediction that ensemble sizes of the order of 20–100 are sufficient to quantify uncertainty in highly nonlinear systems with up to 10^{11} variables, see, e.g., Isaksen et al. (2010) and Van Leeuwen (2019). The reason a relatively small ensemble works is that the focus in the geosciences is on marginal pdfs of one or a few variables, and not the complete joint posterior pdf. As long as the objective remains to estimate (many) marginal pdfs, we see no fundamental reason why this conclusion would not extend to deep learning. This suggests that ensemble sizes of 20–100 should generally suffice, even for large-scale problems.

We further compared our results with widely used ensemble-based uncertainty quantification techniques, including Bagging, Deep Ensembles, and Monte Carlo (MC) Dropout. These methods, in their standard implementations, generate ensembles of model predictions but fail to account for the correct statistical weighting of each ensemble member. When proper importance weighting based on likelihood values is applied, only one member dominates the ensemble, rendering the rest negligible. This effectively collapses the ensemble to a single model, making these methods unsuitable for rigorous uncertainty quantification.

One could argue that existing ensemble-based methods, such as Bagging, Deep Ensembles, or MC Dropout and methods based on them, can be improved by tuning hyperparameters in the neural network to better capture output uncertainty. While it is true that carefully tuned networks can yield more realistic uncertainty estimates, this process is often computationally expensive and lacks generalizability across different applications or datasets. In contrast, a key advantage of our methodology is that it does not require any hyperparameter tuning. The only user-defined parameters involve the sizes of the ensembles, which is a requirement shared with ensemble-based methods. This makes our approach not only more principled but also more practical and broadly applicable.

Finally, we note that when new input data fall outside the domain of the training and testing sets, deep learning models can produce unreliable or even invalid predictions. This represents an additional source of uncertainty that is not explicitly included in our current framework because it cannot yet be quantified in a principled manner. However, two complementary strategies may help mitigate this issue. The first is to make the trained machine more robust by expanding the training dataset to cover a broader region of the input space. Some methods have been proposed in this direction, e.g., using symmetries in training data (Yu and Ma, 2021). Our methodology automatically generates extra training and testing data, but does this in a principled way by perturbing these datasets within their uncertainty as part of the uncertainty quantification. The second methodology is to include extra prior knowledge in the model training process, for instance, extra constraints that enforce physical consistency. Bayes Theorem provides a systematic way to add such extra prior information. Since our methodology is rooted in Bayes' Theorem, it provides a natural and rigorous foundation for including such constraints as priors, and doing so would be straightforward in practice.

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

In summary, our proposed framework offers a statistically consistent, comprehensive, and flexible solution for uncertainty quantification in deep learning applications. It is particularly well-suited to the geosciences, where complex models and measurement uncertainties demand a careful and principled approach.

Open peer review. To view the open peer review materials for this article, please visit http://doi.org/10.1017/eds.2025.10027.

Acknowledgements. The authors thank the in-depth reviews by Dr. A. Carrassi and an anonymous reviewer, which have led to a much better manuscript.

Author contribution. Conceptualization: PJvL and JCC, Methodology: PJvL, Data curation: JCC and CKY, Data visualisation: CKY and PJvL, Writing original draft: PJvL, All authors approved the final submitted draft.

Competing interests. The authors declare that they have no conflict of interest.

Data availability statement. The training and testing data sets are available freely at https://doi.org/10.5281/zenodo.17623563.

Ethics statement. The research meets all ethical guidelines, including adherence to the legal requirements of the study country.

Funding statement. Van Leeuwen was supported by the National Science Foundation under Grant 2,220,201 and the NOAA CADRE program under Grant NA24OARX459C0002-T1-01. Chiu and Yang were supported by Department of Energy under Grant DE-SC0021167, and National Aeronautics and Space Administration Projects 80NSSC22K1546 and 80NSSC23K1017.

References

Abdar M and **Coauthors** (2021) A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion 76*, 243–297. https://doi.org/10.1016/j.inffus.2021.05.008.

Ades M and Van Leeuwen PJ (2012) An exploration of the equivalent-weights particle filter. *Quarterly Journal of the Royal Meteorological Society*. https://doi.org/10.1002/qj.1995.

Amini A, **Schwarting W**, **Soleimany A and Rus D** (2020) Deep evidential regression. https://doi.org/10.48550/arXiv.1910.02600.

Andrieu C, **de Freitas N**, **Doucet A and Jordan MI** (2003) An introduction to mcmc for machine learning. *Machine Learning 50*, 5–43. https://doi.org/10.1023/A:1020281327116.

Balasubramanian V, **Ho S-S and Vovk V** (eds) (2014) *Conformal Prediction for Reliable Machine Learning, Theory, Adaptations and Applications*. Elsevier.

Blundell C, **Cornebise J**, **Kavukcuoglu K and Wierstra D** (2015) Weight uncertainty in neural network. *International Conference on Machine Learning*, 1613–1622.

Breiman L (1996) Bagging predictors. *Machine Learning 24*. https://doi.org/10.1007/BF00058655.

Cheng S, **Quilodrán-Casas C**, **Ouala S** and **Coauthors** (2023) Machine learning with data assimilation and uncertainty quantification for dynamical systems: A review. *IEEE/CAA Journal of Automatica Sinica 10*(6), 1361–1387. https://doi.org/10.1109/JAS.2023.123537.

Chiu C-C, **Yang C-K**, **Van Leeuwen PJ**, **Feingold G** and **Coauthors** (2021) Observational constraints on warm cloud microphysical processes using machine learning and optimization techniques. *Geophysical Research Letters*. https://doi.org/10.1029/2020GL091236.

Doucet A, **de Freitas N and Gordon N** (eds) (2001) *Sequential Monte Carlo methods in practice*. In *Statistics for Engineering and Information Science*. New York: Springer, pp. 581

Evensen G, **Vossepoel F and van Leeuwen PJ** (2022) *Data Assimilation Fundamentals*. Springer. https://doi.org/10.1007/978-3-030-96709-3

Gal Y and Ghahramani Z (2016) Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *International Conference on Machine Learning 48*, 1050–1059.

Gawlikowski J and **Coauthors** (2023) A survey of uncertainty in deep neural networks. *Artificial Intelligence Review 56*, 1513–1589. https://doi.org/10.1007/s10462-023-10562-9.

Gettelman A, **Gagne DJ**, **Chen C-C**, **Christensen MW**, **Lebo ZJ**, **Morrison H and Gantos G** (2021) Machine learning the warm rain process. *Journal of Advances in Modeling Earth Systems*, *13* (**2**), e2020MS002, 268. https://doi.org/10.1029/2020MS002268.

Glienke S and Mei F (2019) Two-dimensional stereo (2d-s) probe instrument handbook. Technical report, DOE ARM. Available at https://www.arm.gov/publications/tech_reports/handbooks/doe-sc-arm-tr-233.pdf.

Glienke S and Mei F (2020) Fast cloud droplet probe (FCDP) instrument handbook. Technical report, DOE ARM. Available at https://www.arm.gov/publications/tech_reports/handbooks/doe-sc-arm-tr-238.pdf.

He W, **Jiang Z**, **Xiao T**, **Xu Z and Li Y** (2025) *A survey on uncertainty quantification methods for deep learning*. https://arxiv.org/pdf/2008.02627.pdf.

Isaksen L, **Bonavita M**, **Buizza R**, **Fisher M**, **Haseler J**, **Leutbecher M and Raynaud L** (2010) Ensemble of data assimilations at ECMWF. *ECMWF Technical Memoranda 636*. https://doi.org/10.21957/obke4k60.

Lakshminarayanan B, **Pritzel A and Blundell C** (2017) Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in Neural Information Processing Systems 30*.

MacKay DJ (1992) A practical bayesian framework for backpropagation networks. *Neural Computation 4*, 448–472. https://doi.org/10.1162/neco.1992.4.3.448.

Mei F, **Wang J**, **Comstock JM** and **Coauthors** (2020) Comparison of aircraft measurements during goamazon2014/5 and acridicon-chuva. *Atmospheric Measurement Techniques 13*, 661–684. https://doi.org/10.5194/amt-13-661-2020.

Neal RM (1996) *Bayesian Learning for Neural Networks*. Springer. https://doi.org/10.1007/978-1-4612-0745-0

Nix DA and Weigend AS (1994) Estimating the mean and variance of the target probability distribution. *IEEE International Conference on Neural Networks*. https://doi.org/10.1109/ICNN.1994.374138.

Osband I (2016) Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS 2016 Workshop on Bayesian Deep Learning*.

Pfreundschuh S, **Eriksson P**, **Duncan D** and **Coauthors** (2018) A neural network approach to estimating a posteriori distributions of bayesian retrieval problems. *Atmospheric Measurement Techniques 11*, 4627–4643. https://doi.org/10.5194/amt-11-4627-2018.

Prechelt L (1998) Automatic early stopping using cross validation: quantifying the criteria, *Neural Netw.* 761–767. doi: 10.1016/s0893-6080(98)00010-0.

Sohl-Dickstein J, **Weiss E**, **Maheswaranathan N and Ganguli S** (2015) Deep unsupervised learning using nonequilibrium thermodynamics. *Proceedings of Machine Learning Research 37*, 2256–2265.

Sønderby CK, **Espeholt L**, **Heek J** and **Coauthors** (2020) Metnet: A neural weather model for precipitation forecasting. *Google Research*, arXiv:2003.12140v2.

Van Leeuwen PJ (2010) Nonlinear data assimilation in geosciences: An extremely efficient particle filter. *Quarterly Journal of the Royal Meteorological Society 136*, 1991–1996. https://doi.org/10.1002/qj.699.

Van Leeuwen PJ (2019) Nonlocal observations and information transfer in data assimilation. *Frontiers in Applied Mathematics and Statistics 26*. https://doi.org/10.3389/fams.2019.00048.

Van Leeuwen PJ (2020) A consistent interpretation of the stochastic version of the ensemble kalman filter. *Quarterly Journal of the Royal Meteorological Society*. https://doi.org/10.1002/qj.3819.

Van Leeuwen PJ, **Cheng Y and Reich S** (2015) *Nonlinear Data Assimilation*. Springer. https://doi.org/10.1007/978-3-319-18347-3

Verdoja F and Kyrki V (2021) Notes on the behavior of mc dropout. In *Presented at ICML Workshop on Uncertainty and Robustness in Deep Learning*. https://arxiv.org/pdf/2008.02627.pdf.

Wang Z, **Li H**, **Ouyang W and Wang X** (2016) Learnable histogram: Statistical context features for deep neural networks. *European Conference on Computer Vision*, 246–262. https://doi.org/10.1007/978-3-319-46448-0_15.

Yu S and Ma J (2021) Deep learning for geophysics: Current and future trends. *Reviews of Geophysics 59*. https://doi.org/10.1029/2021RG000742.

A. Appendix. The uncertainty in the input training data

In this appendix we provide an extended discussion of how one can treat uncertainty in input training data within the machine-learning likelihood. Let the input training data be denoted by X, and the output training data by Z for ease of notation. We also suppress batch ordering in the neural network function for clarity. Following the main text, we introduce the true (but unknown) input data as a random variable X^t, and write:

$$p(Z|X,w) = \int p(Z,X^t|X,w)\,dX^t = \int p(Z|X^t,X,w)p(X^t|X)\,dX^t = \int p(Z|X^t,w)p(X^t|X)\,dX^t, \tag{A.1}$$

where we used that the input data depends not on the weight vector, and that, with X^t given, X does not provide extra information on Z. The factor $p(X^t|X)$ is the pdf that describes the uncertainty in the input training data.

For the other pdf, $p(Z|X^t,w)$, we use the fact that the weight vector w is given, which means that the output training data Z are assumed to be generated via a neural network with that weight vector, applied to the input training data X^t. In general, there will be uncertainty in the output training data and the neural network will not be perfect, so that we can write for each training pair m:

$$Z_m = f\left(X_m^t, w\right) + \epsilon_m\left(X_m^t\right), \tag{A.2}$$

where f is the neural network and $\epsilon_m\left(X_m^t\right)$ denotes the uncertainty in the output training data Z_m and in the neural network, which can potentially depend on the input data value X_m^t. This equation shows that the uncertainty pdf of output training data given X^t and w, so the pdf $p(Z|X^t,w)$, is the same as the pdf of $\epsilon(X^t)$ but shifted by the value $f(X^t,w)$. We used the same argument in Section 3.3, but note that we are now discussing uncertainty in training data, not the result of a new input value.

As an example, if we assume that the uncertainties in the output data and the neural network are independent Gaussian distributed, with variances $\sigma_{z,m}^2$ and σ_f^2, respectively, we can write:

$$p\left(Z_m|X_m^t,w\right) = p_\epsilon\left(\epsilon_m\left(X_m^t\right)\right) = A\exp\left[-\frac{1}{2}\frac{\left(\epsilon_m^2\left(X_m^t\right)\right)}{\sigma_f^2+\sigma_{z,m}^2}\right] = A\exp\left[-\frac{1}{2}\frac{Z_m - f\left(X_m^t,w\right)\right)^2}{\sigma_f^2+\sigma_{z,m}^2}\right] \tag{41}$$

in which A is a normalization factor and $\sigma_f^2+\sigma_{z,m}^2$ is the variance of the uncertainty variable $\epsilon_m\left(X_m^t\right)$. The independence assumption is not essential, and we refer to e.g. the data assimilation literature, such as Evensen et al. (2022), on how to include dependencies in the uncertainties, but that is not the main point here. If we now also assume that the uncertainty in the input training data $p(X^t|X)$ is independent Gaussian, we find for the pdf for each training pair m, using Eq. (A.1):

$$p(Z_m|X_m,w) = A\int \exp\left[-\frac{1}{2}\frac{Z_m - f\left(X_m^t,w\right))^2}{\sigma_f^2+\sigma_{z,m}^2} - \frac{1}{2}\left(X_m^t - X_m\right)^T C_m(-1)\left(X_m^t - X_m\right)\right] dX_m^t, \tag{A.4}$$

in which A is another normalization factor, and the input X_m is allowed to be a vector with uncertainty covariance C_m. (The independency is between input–output samples, not necessarily between the elements of an input sample.) Evaluating this integral is cumbersome because $f\left(X_m^t,w\right)$ is a highly nonlinear function. However, when the uncertainty in the input data is small, such that we can approximate:

$$f\left(X_m^t,w\right) \approx f(X_m,w) + F_m\left(X_m^t - X_m\right), \tag{A.5}$$

in which F_m is the local derivative of the nonlinear function f at input vector value X_m, we can evaluate the integral because its argument is now the product of two Gaussians in the integration variable X_m^t. We then find:

$$p(Z_m|X_m,w) = A\int \exp\left[-\frac{1}{2}\frac{Z_m - f\left(X_m^t,w\right))^2}{\sigma_f^2+\sigma_{z,m}^2+F_mC_mF_m^T}\right], \tag{A.6}$$

Therefore, for all N_X input data:

$$p(Z|X,w) = \prod_{m=1}^{N_X} p(Z_m|X_m,w) = A\int \exp\left[-\frac{1}{2}\sum_{m=1}^{N_X}\frac{Z_m - f\left(X_m^t,w\right))^2}{\sigma_f^2+\sigma_{z,m}^2+F_mC_mF_m^T}\right], \tag{A.7}$$

in which A is yet another normalization constant. We conclude that, if the uncertainty in the input data is small, such that a linearization of the network around this input value is accurate, we recover the standard deep learning likelihood. However, the variance in that likelihood contains the sum of the output uncertainty, neural network uncertainty, and the input uncertainty transformed to output space via the linearized neural network (the $F_mC_mF_m^T$ term).

When the uncertainties are not Gaussian or when the linearization of the neural network function is not accurate, analytical solutions quickly become impossible. In that case, we can use Monte-Carlo methods to evaluate the integral in Eq. (A.1). The idea is to draw N_x samples from $p\left(X_m^t|X_m\right)$. This means that we represent $p\left(X_m^t|X_m\right)$ by a set of delta functions, each centered on a sample $X_{m,j}^t$, as:

$$p\left(X_m^t|X_m\right) = \frac{1}{N_X}\sum_{j=1}^{N_X}\delta\left(X_m^t - X_{m,j}^t\right), \tag{A.8}$$

where δ denotes the Dirac delta distribution. Substituting this in Eq. (A.1), together with Eq. (BA.6), we find for each training pair m:

$$p(Z_m|X_m,w) = A\frac{1}{N_X}\sum_{j=1}^{N_X}\exp\left[-\frac{1}{2}\frac{Z_m - f\left(X_{m,j}^t,w\right))^2}{\sigma_f^2+\sigma_{z,m}^2}\right], \tag{A.9}$$

such that

$$p(Z|X,w) = A\frac{1}{N_X}\sum_{j=1}^{N_X}\exp\left[-\frac{1}{2}\sum_{m=1}^{N_X}\frac{Z_m - f\left(X_{m,j}^t,w\right))^2}{\sigma_f^2+\sigma_{z,m}^2}\right], \tag{A.10}$$

which is the formulation used in the main text.

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

B. Appendix. Full results for the autoconversion case

The Figure B.1 shows the full uncertainty pdfs for all 50 new input values (blue curves), the uncertainty due to input uncertainty and the uncertainty of the network as determined from the testing data (black curves), the unperturbed autoconversion estimate (red bar) and the uncertainty estimate from the weight uncertainty only (black bars), but omitting the importance weights (see main text for discussion).

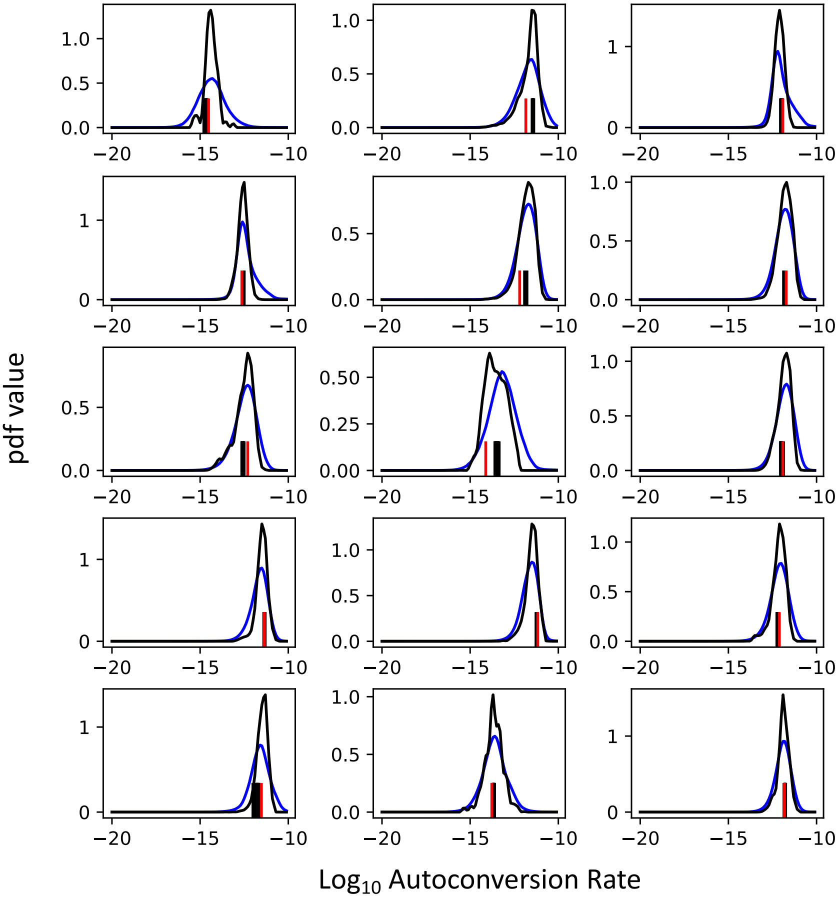

Figure B1. *Examples of total uncertainty pdf in the output autoconversion rate for 50 input vectors. The blue curves are the total uncertainty pdfs, black curves that are derived considering the uncertainty in the new input vector and the uncertainty in the neural network only. The red bar is the autoconversion rate calculated by traditional deep learning without uncertainty quantification. The black bars are the output samples generated using Bagging and used as a measure of uncertainty for the Bagging approach.*

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

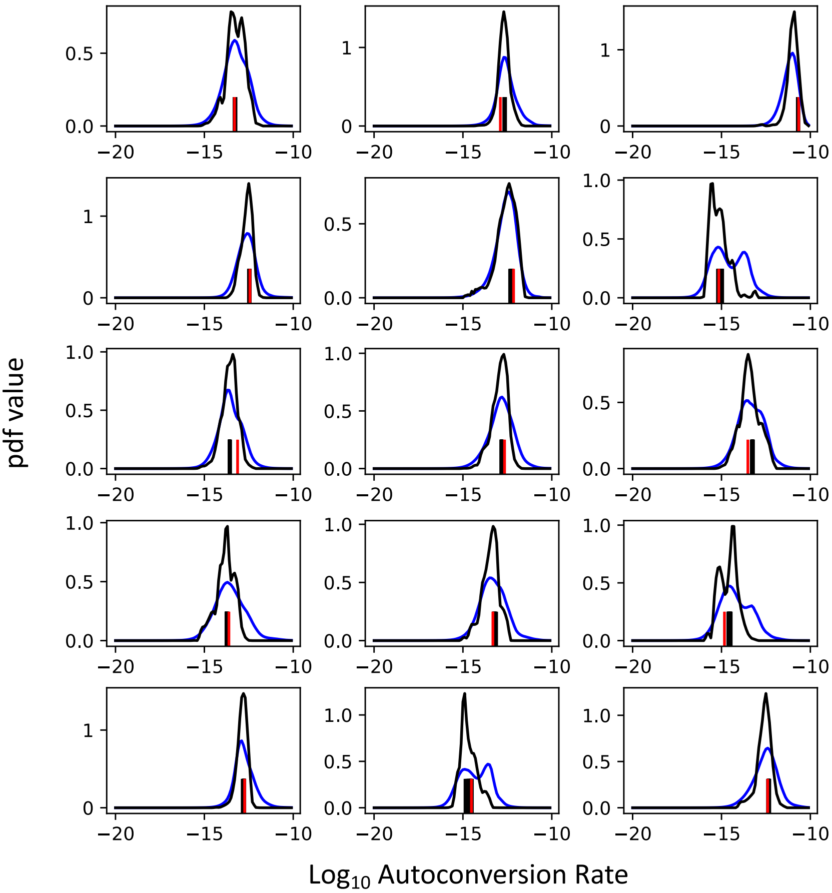

Figure B1. *(Continued)*

https://doi.org/10.1017/eds.2025.10027 Published online by Cambridge University Press

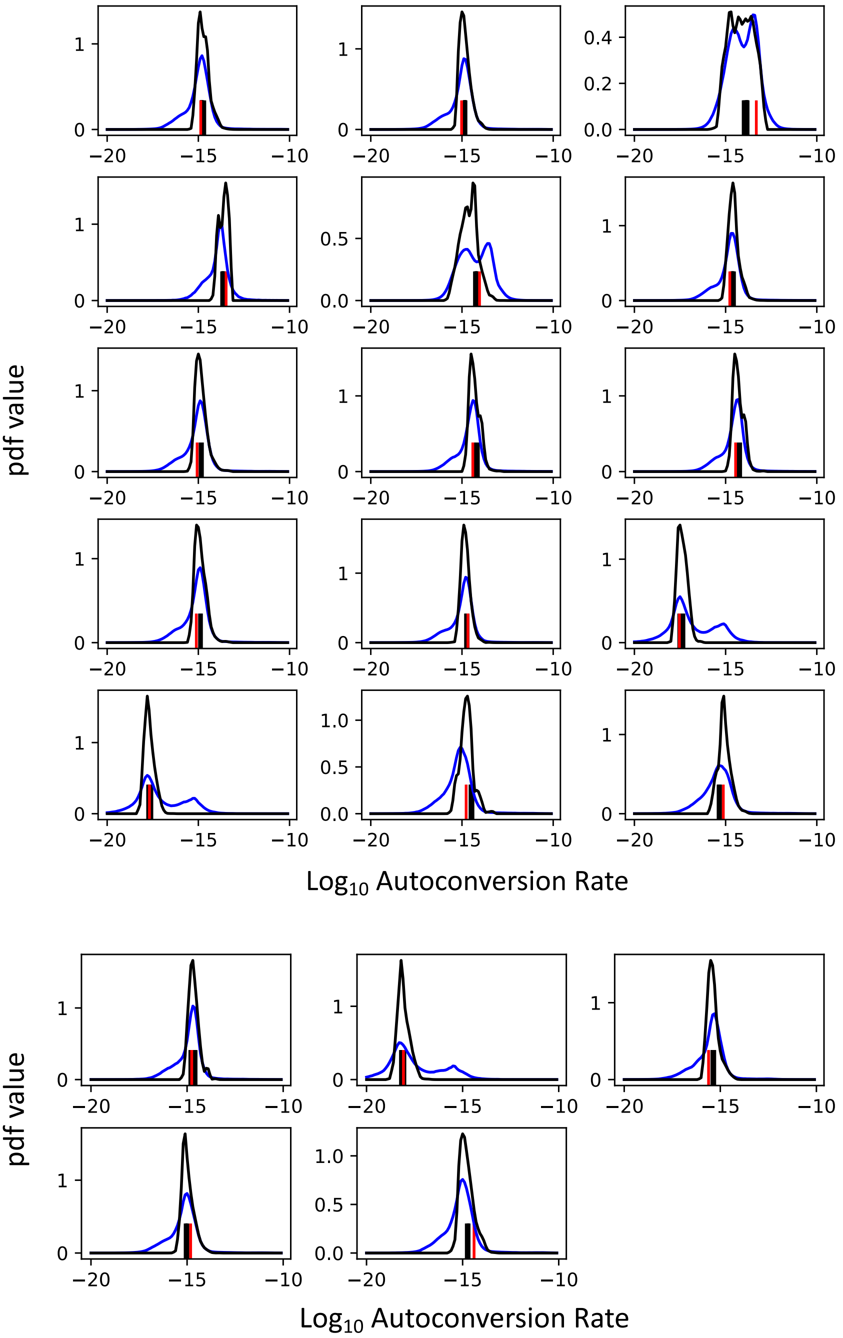

Figure B1. *(Continued)*

Cite this article: van Leeuwen PJ, Chiu J.-Y.C and Yang C.-K.K (2025). Uncertainty quantification for deep learning. *Environmental Data Science*, 4: e53. doi:10.1017/eds.2025.10027