Uncertainty Quantification in Machine Learning for Engineering Design and Health Prognostics: A Tutorial

Venkat Nemani[a], Luca Biggio[b], Xun Huan[c], Zhen Hu[d], Olga Fink[e], Anh Tran[f], Yan Wang[g], Xiaoge Zhang[h,i,*], Chao Hu[j,*]

[a]*Department of Mechanical Engineering, Iowa State University, Ames, IA 50011, USA*
[b]*Data Analytics Lab, ETH, Zürich, Switzerland*
[c]*Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109, USA*
[d]*Department of Industrial and Manufacturing Systems Engineering, University of Michigan-Dearborn, Dearborn, MI 48128, USA*
[e]*Intelligent Maintenance and Operations Systems, EPFL, Lausanne, 12309, Switzerland*
[f]*Scientific Machine Learning, Sandia National Laboratories, Albuquerque, NM 87123, USA*
[g]*George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA*
[h]*Department of Industrial and Systems Engineering, The Hong Kong Polytechnic University, Kowloon, Hong Kong*
[i]*Center for Advances in Reliability and Safety (CAiRS), New Territories, Hong Kong*
[j]*Department of Mechanical Engineering, University of Connecticut, Storrs, CT 06269, USA*

Abstract

On top of machine learning (ML) models, uncertainty quantification (UQ) functions as an essential layer of safety assurance that could lead to more principled decision making by enabling sound risk assessment and management. The safety and reliability improvement of ML models empowered by UQ has the potential to significantly facilitate the broad adoption of ML solutions in high-stakes decision settings, such as healthcare, manufacturing, and aviation, to name a few. In this tutorial, we aim to provide a holistic lens on emerging UQ methods for ML models with a particular focus on neural networks and the applications of these UQ methods in tackling engineering design as well as prognostics and health management problems. Toward this goal, we start with a comprehensive classification of uncertainty types, sources, and causes pertaining to UQ of ML models. Next, we provide a tutorial-style description of several state-of-the-art UQ methods: Gaussian process regression, Bayesian neural network, neural network ensemble, and deterministic UQ methods focusing on spectral-normalized neural Gaussian process. Established upon the mathematical formulations, we subsequently examine the soundness of these UQ methods quantitatively and qualitatively (by a toy regression example) to examine their strengths and shortcomings from different dimensions. Then, we review quantitative metrics commonly used to assess the quality of predictive uncertainty in classification and regression problems. Afterward, we discuss the increasingly important role of UQ of ML models in solving challenging problems in engineering design and health prognostics. Two case studies with source codes available on GitHub are used to demonstrate these UQ methods and compare their performance in the life prediction of lithium-ion batteries at the early stage (case study 1) and the remaining useful life prediction of turbofan engines (case study 2).

Keywords: Machine learning, Uncertainty quantification, Engineering design, Prognostics and

*Correspondence authors.
    Email addresses:* `xiaoge.zhang@polyu.edu.hk` (Xiaoge Zhang), `chao.hu@uconn.edu` (Chao Hu)

Nomenclature

List of acronyms

ARD	Automatic relevance determination
BNN	Bayesian neural network
DL	Deep learning
DNN	Deep neural network
ECE	Expected calibration error
EI	Expected improvement
ELBO	Evidence lower bound
GAN	Generative adversarial network
GPR	Gaussian process regression
HMC	Hamiltonian Monte Carlo
KL	Kullback–Leibler
MC	Monte Carlo
MCMC	Markov chain Monte Carlo
MFVI	Mean-field variational inference
ML	Machine learning
MSE	Mean squared error
NLL	Negative log-likelihood
OOD	Out of distribution
PDF	Probability density function
PHM	Prognostics and health management
RUL	Remaining useful life
SNGP	Spectral-normalized neural Gaussian process
SVGD	Stein variational gradient descent
UQ	Uncertainty quantification
VAE	Variational autoencoder
VI	Variational inference

List of mathematical notations

$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$ Training data

D — Number of features (dimensions) in a single input $\mathbf{x}$

ε — A random noise variable following a zero-mean Gaussian distribution

$\mathbb{E}[\bullet]$ — Expectation of $\bullet$

$k(\mathbf{x}, \mathbf{x}')$ — Covariance function or kernel in GPR depicting the covariance between function outputs at $\mathbf{x}$ and $\mathbf{x}'$

λ — Parameter to be optimized in the variational distribution q

l — Length scale parameter of a kernel

N — Number of training samples

p — Probability density

$p(\boldsymbol{\theta})$ — Prior distribution of $\boldsymbol{\theta}$

$p(\boldsymbol{\theta}|\mathcal{D})$ — Posterior distribution of $\boldsymbol{\theta}$ given the training data $\mathcal{D}$

$p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X})$ — Likelihood function indicating the probability of observing $\mathbf{y}$ given the parameters $\boldsymbol{\theta}$ and inputs $\mathbf{X}$

$q(\boldsymbol{\theta}; \lambda)$ — A variational distribution parameterized by λ to approximate the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$

σ_{f} — Signal amplitude parameter of a kernel

σ_{ε} — Standard deviation of a random noise variable ε

$\boldsymbol{\theta}$ — Set of tunable parameters in an ML model

$\boldsymbol{\theta}^*$ — Set of optimal parameters in an ML model after tuning

$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ Inputs (or input points) in a training dataset for BNN

$\mathbf{X}_{\mathrm{t}}$ — Matrix representation of inputs in training data, i.e., $\mathbf{X}_{\mathrm{t}} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^{\mathrm{T}} \in \mathbb{R}^{N \times D}$

$\mathbf{x}$ — A single input, $\mathbf{x} \in \mathbb{R}^N$

$\mathbf{x}_*$ — A test point

y — A single observation/target, $y \in \mathbb{R}^1$

$\mathbf{y} = \{y_1, y_2, \cdots, y_N\}$ Observations/targets in a training dataset to be predicted by an ML model

$\mathbf{y}_t$ Matrix representation of target output in $\mathbb{R}^N$
training data, that is $\mathbf{y}_t = [y_1, \ldots, y_N]^T \in$

1. Introduction

In recent years, data-driven machine learning (ML) models have become increasingly prevalent across a wide range of engineering fields. Two application domains of interest to this tutorial are engineering design and post-design health prognostics. The ML community has devoted significant efforts toward creating deep learning (DL) models that yield improved prediction accuracy over earlier DL models on publicly available, large, standardized datasets, such as MNIST [1], ImageNet [2], Places [3], and Microsoft COCO [4]. Among these DL models are deep neural networks (DNNs), known for their ability to extract high-level abstracted features from large volumes of data automatically achieved through multiple layers of neurons and activation functions in an end-to-end fashion.

Despite record-breaking prediction accuracy on some fixed sets of test samples (i.e., images in the case of computer vision), these neural networks typically have difficulties in generalizing to data not observed during model training. Suppose test samples come from a distribution substantially different from the training distribution, where most of the training samples are located. These test samples can be called out-of-distribution (OOD) samples. Trained neural network models tend to produce large prediction errors on these OOD samples. Despite considerable efforts, such as domain adaption [5–7], aimed at improving the generalization performance of neural network models, the issue of poor generalizability still persists. Another limitation that adds to the challenge is that complex ML models, such as DNNs, are mostly black-box in nature. It is generally preferred to use simpler models (e.g., linear regression and decision tree) that are easier to interpret unless more complex models can be justified with non-incremental benefits (e.g., substantially improved accuracy). In recent years, the growing availability of large volumes of data has made complex models, which are often significantly more accurate than simple models, the obvious better choice in many ML applications where prediction accuracy is the priority. Consequently, black-box ML models that are hard to understand are increasingly deployed, particularly in big data applications. Some efforts have been made to address the lack of interpretability, with notable explanation algorithms such as SHAP [8] and Grad-CAM [9] and a good review of interpretable ML [10]. Despite these recent efforts, many complex ML models are still implemented as black-box models and cannot explain their predictions to the end user for various reasons. This limitation makes it extremely intricate for the end user to understand the decision mechanism behind a neural network's prediction. Given these two limitations (difficulties in extrapolating to OOD samples and lack of interpretability), it is vital to quantify the predictive uncertainty of a trained ML model and communicate this uncertainty to end users in an easy-to-understand way. To enhance algorithmic transparency and trustworthiness, uncertainty quantification (UQ) and interpretation should ideally be performed together, with UQ providing information on the confidence of complex machine learning models in making predictions. This integration allows for a better understanding of often difficult-to-interpret models and their predictions.

Let us first look at typical ways to express and communicate predictive uncertainty. A simple case is with classification problems, where the probability of the model-predicted class can depict model confidence at a prediction. For example, a fault classification model may predict a bearing to have an inner race fault with a 90% probability/confidence. In regression problems, predictive uncertainty is often communicated as confidence intervals, shown as error bars on graphs visualizing predictions. For instance, we could train a probabilistic ML model to predict the number of weeks a rolling element bearing can be used before failure, i.e., the remaining useful life (RUL). An example prediction may be 120 ± 15, in weeks, which represents a two-sided 95% confidence interval (i.e., ~1.96 standard deviations subtracted from or added to the mean estimate assuming the model-predicted RUL follows a Gaussian distribution). A narrower confidence interval comes from lower predictive uncertainty, which suggests higher model confidence.

One clear advantage of UQ is that it helps end users determine when they can trust predictions made by the model and when they need extra caution while making decisions based on these predictions. This is especially important when incorrect decisions can lead to severe financial losses or even life-threatening outcomes. Towards this end, the integration of UQ in ML models, as well as the sound quantification and calibration of uncertainty in ML model prediction, has a viable potential to tackle a central research question the ML community confronts – safety assurance of ML models [11–14]. In fact, the absence of essential performance characteristics (e.g., model robustness and safety assurance) has emerged as the fundamental roadblock to limiting ML's application scope in risk-insensitive areas, while its adoptions in high-stakes, high-reward decision environments (e.g., healthcare, aviation, and power grid) are still in the infancy stage primarily because of the reluctance of end users to delegate critical decision making to machine intelligence in cases where the safety of patients or critical engineering systems might be put at stake [15–18]. Towards the translation of ML solutions in high-risk domains, UQ offers an additional dimension by extending the traditional discipline of statistical error analysis to capture various uncertainties arising from limited or noisy data, missing variables, incomplete knowledge, etc. This development has wide-ranging implications for supporting quantitative and precise risk management in high-stakes decision-making settings, particularly concerning potential model failures and decision limitations of ML algorithms. However, the evaluation of ML model performance on most benchmarking datasets focuses exclusively on some form of prediction accuracy on a fixed test dataset; it rarely considers the quality of predictive uncertainty. As a result, UQ of ML models is typically pushed to the sidelines, yielding the centerlines to prediction accuracy. In reality, underestimating uncertainty (overconfidence) can create trust issues, while overestimating uncertainty (underconfidence) may result in overly conservative predictions, ultimately diminishing the value of ML.

More recently (approximately since 2015), there has been growing interest in approaches to estimating the predictive uncertainty of deep learning models, for example, in the form of class probability for classification and predicted variance for regression, as discussed earlier. The growing interest can be attributed to failure cases where trained ML models produced unexpectedly incorrect predictions on test samples while communicating high confidence in the predictions [19] and those where models changed their predictions substantially in response to minor, unimportant changes

to samples (or so-called *adversarial samples*) [20]. Two pioneering studies that stimulated many subsequent efforts created two widely used approaches to UQ of neural networks: (1) Monte Carlo (MC) dropout as a computationally efficient alternative to traditional Bayesian neural network [21] and (2) neural network ensemble consisting of multiple independently trained neural networks, each predicting a mean and standard deviation of a Gaussian target [22]. Another notable early study highlighted differences between aleatory and epistemic uncertainty and discussed situations where quantifying aleatory uncertainty is important and where quantifying epistemic uncertainty is important [23]. A common understanding in the ML community towards these two types of uncertainty has been the following: aleatory uncertainty can be considered *data uncertainty* and represents inherent randomness (e.g., measurement noise) in observations of the target that an ML model is tasked with predicting; epistemic uncertainty can be treated as *model uncertainty* and results from having access to only limited training data, which makes it not possible to learn a precise model. As discussed in Sec. 2.1, aleatory and epistemic uncertainty could encompass more sources and causes than the well-known data and model uncertainty.

The engineering design community has a long history of applying Gaussian process regression (GPR) or kriging, an ML method with UQ capability, to build cheap-to-evaluate surrogates of expensive simulation models for simulation-based design, dating back to the early 2000s [24–26]. GPR has an elegant way of quantifying aleatory and epistemic uncertainty and can produce high uncertainty on OOD samples. However, the UQ capability of GPR is typically not used to detect OOD samples or quantify the epistemic uncertainty of a final built surrogate. Rather, it is leveraged in an adaptive sampling scheme to encourage sampling in highly uncertain and critical regions of the input space (exploration) to minimize the number of training samples for either (1) building an accurate surrogate within some lower and upper bounds of input variables (local or global surrogate modeling) [27–29] or (2) finding a globally optimally design for some expensive-to-evaluate black-box objective function [30, 31]. Additionally, little effort is made to evaluate the quality of UQ for a trained GPR model, likely because the model makes predictions on samples within predefined design bounds and does not need to extrapolate much (low epistemic uncertainty). Other classical surrogate modeling methods, such as standard artificial neural networks and support vector machines, are generally less capable of quantifying predictive uncertainty, especially epistemic uncertainty. These methods and GPR are typically used to build surrogates that act as "deterministic" transfer functions and allow propagating aleatory uncertainty in input variables to derive the uncertainty in the model output, known as uncertainty propagation [32]. The recent two years have seen efforts applying DNNs to surrogate modeling for reliability analysis [33–35]. Similarly, these DNNs do not have built-in UQ capability and are typically used as deterministic functions primarily for uncertainty propagation.

For over two decades, the prognostics and health management (PHM) community has used ML methods with built-in UQ capability as part of the health forecasting/RUL prediction process. Early applications include the Bayesian linear regression for aircraft turbofan engine prognostics [36], the relevance vector machine, a probabilistic kernel regression model of an identical function form to the support vector machine [37], for battery prognostics [38–40] and general purpose prognostics [41, 42],

and GPR for battery prognostics [43–45]. UQ of ML models for PHM is perceived to have more significance than that for engineering design, mainly due to (1) the more likely lack of sufficient training data, given an expensive and time-consuming process to collect run-to-failure data for training ML models for health prognostics, (2) the higher need to extrapolate to unseen operating conditions in PHM applications, and (3) the higher criticality of consequences from incorrectly made maintenance decisions. Two representative reviews of UQ work in the field of PHM can be found in [46, 47]. Both reviews seem to focus on identifying uncertainty sources in health prognostics and discussing ways to propagate these sources of uncertainty to derive the probability distribution of RUL.

Sec. 2: Types and sources of uncertainty	2.1: Aleatory and epistemic uncertainty 2.2: Decomposition of predictive uncertainty 2.3: Reduction of epistemic uncertainty
Sec. 3: Methods for UQ of ML models	3.1: Gaussian process regression 3.2: Bayesian neural network 3.3: Neural network ensemble 3.4: Deterministic methods 3.5: Toy example 3.6: Summary
Sec. 4: Evaluation of predictive uncertainty	4.1: Calibration curves and metrics 4.2: Sparsification plots and metrics 4.3: Negative log-likelihood 4.4: Accuracy vs. UQ quality
Sec. 5: UQ of ML models in prognostics	5.1: Uncertainty-aware ML for PHM 5.2: Uncertainty evaluation metrics for prognostics 5.3: Discussion
Sec. 6: Case studies	6.1: Case study 1: Battery early life prediction 6.2: Case study 2: Turbofan engine prognostics
Sec. 7: Other topics related to UQ of ML models	7.1: Physics-informed ML 7.2: Probabilistic Learning on Manifolds 7.3: Interpretability of ML models for dynamic systems 7.4: Polynomial chaos expansion

Sec. 8: Conclusion and outlook

Figure 1: Overview of the organization of the tutorial paper.

Within this paper, we seek to provide a comprehensive overview of emerging approaches for UQ of ML models and a brief review of applications of these approaches to solve engineering design and health prognostics problems. As for the ML models, our tutorial focuses on neural networks due to their increasing popularity amongst academic researchers and industrial practitioners. In essence, we look at methods to quantify the predictive uncertainty of neural networks, i.e., methods for UQ of neural networks. This focus differs from the notion of "ML for UQ" where UQ of engineered

systems or processes becomes the primary task, and ML models are built only to serve the primary purpose of UQ. Figure 1 shows an outline of this tutorial paper. Our tutorial possesses four unique properties that distinguish it from recent reviews on UQ of ML models in the ML community [48–50], computational physics community [51], and PHM community [46, 47], and a recent benchmarking study assessing state-of-the-art UQ methods for neural networks in the PHM community [52].

- First, we give a detailed classification of uncertainty types, sources, and causes (Sec. 2.1) and discuss ways to reduce epistemic uncertainty (Sec. 2.3). Our classification and discussion complement the theoretical and data science-oriented discussions in the ML community and provide more context for researchers and practitioners in the engineering design and PHM communities. Additionally, we provide an easy-to-understand explanation of the process of decomposing the total predictive uncertainty of an ML model into aleatory and epistemic uncertainty, leveraging simple mathematical examples (Sec. 2.2).

- Second, we provide a tutorial-style description and a qualitative and quantitative comparison of emerging UQ approaches developed in the ML community over the past eight years. This tutorial-style description covers both methodologies (Sec. 3) and their implementations on real-world case studies (Sec. 6). The tutorial style also applies to our discussion on methods and metrics for assessing the quality of predictive uncertainty (Sec. 4), an increasingly important exercise in UQ of ML models.

- Third, although our tutorial focuses primarily on UQ methods for ML models, it additionally briefly covers a collection of recent studies that apply some of the emerging UQ approaches to solve challenging problems in engineering design (Appendix B) and health prognostics (Sec. 5). This review is meaningful because as the adoption of ML techniques in design and prognostics rapidly increases, we also expect to see an increasing need for UQ of ML models. Note that deep neural network architectures, originally created for computer vision tasks based on large image datasets, can be readily adopted in engineering design tasks, such as surrogate modeling for reliability analysis [28, 29] and generative designs [53–55], and PHM tasks, such as fault diagnostics [56–60] and RUL prediction [61–63]. We hope to provide observations and insights that can help guide researchers in the engineering design and PHM communities in choosing and implementing the UQ methods suitable for specific applications. This unique and distinct application area distinguishes our tutorial paper from a recent review paper on UQ of ML models [51], which explored the use of ML with UQ for solving partial differential equations and learning neural operators. Additionally, our tutorial paper complements a recent effort in benchmarking UQ methods for neural networks on aircraft engine prognostics [52].

- Fourth, we share, on GitHub, our code for implementing several UQ methods on one toy regression example (Sec. 3.5) and two real-world case studies on health prognostics (Sec. 6). Our implementations have been thoroughly verified to have quality on par with high quality implementations by the ML community. Some of our implementations are directly built on top of code shared by the ML community. We anticipate our code will allow researchers and

practitioners in the engineering design and PHM communities to replicate results, customize existing UQ methods to specific applications, and test new methods. Moving forward, we plan to make continuous improvements to the codebase, e.g., by polishing lines of code and adding new methods as they become available.

Our tutorial paper is concluded in Sec. 8, where we also discuss directions for future research.

2. Types and sources of uncertainty

This section first provides the definitions of different types of uncertainty and a summary of their sources and causes, and then discusses the methods to decompose and reduce the predictive uncertainty of ML models.

2.1. Aleatory and epistemic uncertainty

Uncertainty, in general, can be classified into two types: aleatory uncertainty and epistemic uncertainty [64]. This classification of uncertainty originated in the engineering domain for risk and reliability analysis [64] and is also applicable to the ML domain [19, 23]. The definitions and sources of these two types of uncertainty are summarized as follows.

i. **Aleatory uncertainty**: It stems from natural variability and is irreducible by nature [64]. This type of uncertainty captures the noise inherent in physical systems [65]. A typical example of aleatory uncertainty is the noise in sensor measurements, which would persist even if more data were collected. In ML, aleatory uncertainty represents the inherently stochastic nature of an input, an output, or the dependency between these two [19]. Example causes of aleatory uncertainty include variability of material properties from one specimen to another, variability of response from different runs of the same experiment, variability in classes for classification problems, and variability of the output for regression problems. This type of uncertainty is usually modeled as a part of the likelihood function in a probabilistic ML model. The predictions of the ML model is also probabilistically distributed [65]. This way of capturing the observation uncertainty (sometimes termed *data uncertainty*) is leveraged by several UQ methods, such as homoscedastic (Eq. (13)) and heteroscedastic (Eq. (30)) GPRs discussed in Sec. 3.1 and neural network ensemble (Eq. (30)) discussed in Sec. 3.3.

ii. **Epistemic uncertainty**: This type of uncertainty is attributed to things one could know in principle but remain unknown in practice due to a lack of knowledge. It is reducible by nature [64]. Common causes of epistemic uncertainty in the engineering domain include model simplification, model-form selection, computational assumptions, lack of information about certain model parameters, and numerical discretization. ML models generally have similar epistemic uncertainty sources as engineering models. In particular, the epistemic uncertainty in ML models can be further classified into the following two categories:

  (a) *Model-form uncertainty* is due to the simplification and approximation procedures involved in ML model construction. It is usually associated with the choices of model types, such

as the architectures and activation functions of neural networks and the model forms of kernel functions in GPR models.

(b) *Parameter uncertainty* is associated with model parameters and arises from the model calibration and training processes. Major causes of parameter uncertainty include a lack of enough training data, inherent bias in the training data due to low data fidelity, and difficulties in converging to optimal solutions faced by training algorithms.

Table 1 summarizes the common sources and associated causes of the above two types of uncertainty in ML. When the test dataset falls outside the training data distribution, the ML model predictions likely have high epistemic uncertainty since the performance of ML models is typically poorer in extrapolation than in interpolation. When the test data in some regions of the input space are associated with higher measurement noise, they can lead to higher aleatory uncertainty. Additionally, data of output used to train an ML model could deviate from the true values of the output. When the error is caused by random noise of measurement, it will lead to aleatory uncertainty in the output. However, when there is also bias in the data, the error causes additional epistemic uncertainty. For instance, when the bias is caused by low data fidelity representing the data's low accuracy, this bias will result in epistemic uncertainty, which is reducible by adding high-fidelity data for training.

Table 1: Types, sources, and causes of uncertainty in ML

Type	Source	Cause(s)
Aleatory uncertainty	Observational uncertainty (model input and output)	Measurement noise (e.g., sensor noise in measuring inputs/outputs of ML models)
	Natural variability (model input)	Variability in material properties, manufacturing tolerance, variability in loading and environmental conditions, etc.
	Lack of predictive power (model input)	Dimension reduction, non-separable classes in input space (classification), etc.
Epistemic uncertainty	Parameter uncertainty	Limited training data, local optima of ML model parameters, low-fidelity training data*, etc.
	Model-form uncertainty	Choices of neural network architectures and activation and other functions, missing input features, etc.

* Data fidelity is the accuracy with which data quantifies and embodies the characteristics of the source [66].

Note that aleatory uncertainty could exist in the input, output, or both of an ML model. A common practice of dealing with aleatory uncertainty in the inputs is propagating the uncertainty to the output after constructing the ML model. The aleatory uncertainty in the output, however, is more challenging to tackle, since it needs to be accounted for during the training of an ML model (see more detailed discussion in Secs. 3.1 and 3.3). Uncertainty propagation of input aleatory

uncertainty to the output is not the focus of this paper. We mainly focus on accounting for aleatory uncertainty in the output during the training of an ML model. Moreover, it is worth mentioning that aleatory uncertainty and epistemic uncertainty often coexist, making it difficult to separate them. Even though some efforts have been made in recent years to separate these two types of uncertainty, for example, by using the variance decomposition method (see Sec. 2.2) that has been extensively studied in the global sensitivity analysis field [67–69], a clean and complete separation of these two types of uncertainty may only be possible for some cases when there are no complicated interactions between aleatory and epistemic uncertainty sources. We are interested in separating these two types of uncertainty often because we are usually concerned about when the "prediction accuracy" of ML models becomes so low that model prediction cannot be trusted. These "break-down" cases are typically associated with high epistemic uncertainty, the quantification of which would help identify low-confidence predictions by the ML models and avoid making sub-optimal or even incorrect decisions whose consequences could be very costly and even life-threatening in safety-critical applications.

Suppose we cannot separate these two types of uncertainty and only look at their combination. In that case, we only have access to the total predictive uncertainty of an ML model, which can be used to measure the model's confidence in predicting at a test point, given both noise sources in the environment and the reducible uncertainty arising from a lack of training data. The total predictive uncertainty is often what commercially available ML solutions produce as ML outputs (e.g., the probability mass function of the predicted health class for health diagnostics and the variance of the remaining useful life estimate for health prognostics).

2.2. Decomposition of predictive uncertainty

From the above discussion, we can intuitively and qualitatively tell the difference between aleatory (irreducible) and epistemic (reducible) uncertainty. Some recent studies also attempted to estimate these two types of uncertainty quantitatively. To this end, it is essential to decompose the total predictive uncertainty into aleatory and epistemic components [70–72]. Let us consider the simplest form of a probabilistic ML model, a linear regression model. This model is parameterized by weights and biases, concatenated into a vector $\boldsymbol{\theta}$. Then, we can express this linear regression model in the following form:

$$\hat{y}(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x} + \varepsilon, \tag{1}$$

where $\varepsilon \sim \mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}\right)$ is the Gaussian noise variable with $\mathbf{I}$ denoting an $D \times D$ identity matrix. Note that applying an activation function to the linear term $\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}$ introduces nonlinearity to the regression model, making it a building block in a neural network.

If we make a Bayesian treatment of Eq. (1), we will start with a prior distribution $p(\boldsymbol{\theta})$ over model parameters $\boldsymbol{\theta}$ and then infer a posterior from a training dataset $\mathcal{D}$, $p(\boldsymbol{\theta}|\mathcal{D})$. Essentially, we build a Bayesian linear regression model, from which we can derive the predictive distribution of y at a given training/validation/test point $\mathbf{x}$ via marginalization:

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta}. \tag{2}$$

To make the discussion more concrete and easier to understand, we further assume that Eq. (1) is a two-dimensional model (i.e., $D = 2$) and the posterior of $\boldsymbol{\theta}$ is jointly Gaussian: $p(\boldsymbol{\theta}|\mathcal{D}) = \mathcal{N}(\mu_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$ with $\mu_{\boldsymbol{\theta}} = [\mu_{\theta_1}, \mu_{\theta_2}]^{\mathrm{T}}$ and a covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \begin{bmatrix} \sigma_{\theta_1}^2 & \rho\sigma_{\theta_1}\sigma_{\theta_2} \\ \rho\sigma_{\theta_1}\sigma_{\theta_2} & \sigma_{\theta_2}^2 \end{bmatrix}$. The predicted y then follows a Gaussian distribution given by:

$$p(y|\mathbf{x}, \mathcal{D}) = \mathcal{N}(\mu_{\theta_1}x_1 + \mu_{\theta_2}x_2, \ \sigma_{\theta_1}^2 x_1^2 + \sigma_{\theta_2}^2 x_2^2 + 2\rho\sigma_{\theta_1}\sigma_{\theta_2}x_1 x_2 + \sigma^2). \tag{3}$$

For classification problems, we typically use *differential entropy* as a measure of uncertainty [73]; for regression problems, a typical choice is *variance* of a Gaussian output [74]. Since we deal with a regression problem, we use variance to measure uncertainty in this example. The total predictive uncertainty is measured as the predicted variance

$$\mathcal{U}_{\text{total}} = Var(y|\mathbf{x}, \mathcal{D}) = \sigma_{\theta_1}^2 x_1^2 + \sigma_{\theta_2}^2 x_2^2 + 2\rho\sigma_{\theta_1}\sigma_{\theta_2}x_1 x_2 + \sigma^2. \tag{4}$$

The aleatory uncertainty can be measured as the variance of the Gaussian noise (intrinsic in the data)

$$\mathcal{U}_{\text{aleatory}} = \sigma^2. \tag{5}$$

Then, the epistemic uncertainty can be estimated by subtracting the aleatory uncertainty from the total predictive uncertainty

$$\mathcal{U}_{\text{epistemic}} = \mathcal{U}_{\text{total}} - \mathcal{U}_{\text{aleatory}} = \sigma_{\theta_1}^2 x_1^2 + \sigma_{\theta_2}^2 x_2^2 + 2\rho\sigma_{\theta_1}\sigma_{\theta_2}x_1 x_2. \tag{6}$$

It can be seen from the above equation that the epistemic uncertainty is dependent on (1) the posterior variances ($\sigma_{\theta_1}^2$ and $\sigma_{\theta_2}^2$) and covariance ($\rho\sigma_{\theta_1}\sigma_{\theta_2}$) of the model parameters $\boldsymbol{\theta}$ and (2) values of the input variables (x_1 and x_2). The noise variance, which measures the intrinsic uncertainty in the data, does not affect and has nothing to do with the epistemic uncertainty.

Using the law of total variance or variance-based sensitivity analysis [75], we can generalize Eqs. (4) through (6) for uncertainty decomposition:

$$\underbrace{Var(y|\mathbf{x}, \mathcal{D})}_{\mathcal{U}_{\text{total}}} = \underbrace{\mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})}[Var(y|\mathbf{x}, \boldsymbol{\theta})]}_{\mathcal{U}_{\text{aleatory}}} + \underbrace{Var_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})}[\mathbb{E}(y|\mathbf{x}, \boldsymbol{\theta})]}_{\mathcal{U}_{\text{epistemic}}}, \tag{7}$$

where $\mathbb{E}(y|\mathbf{x}, \boldsymbol{\theta})$ and $Var(y|\mathbf{x}, \boldsymbol{\theta})$ are the mean and variance of y at $\mathbf{x}$ for a given realization of $\boldsymbol{\theta}$. The first term on the right-hand side of Eq. (7), $\mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})}[Var(y|\mathbf{x}, \boldsymbol{\theta})]$, computes the average of the variance of y, $Var(y|\mathbf{x}, \boldsymbol{\theta})$, over $p(\boldsymbol{\theta}|\mathcal{D})$. This term does not consider any contribution of parameter ($\boldsymbol{\theta}$) uncertainty to the variance of y, as the expectation operation, $\mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})}[\cdot]$, take out the contribution of the variation in $\boldsymbol{\theta}$. It only captures the intrinsic data noise (ε) and therefore represents the aleatory uncertainty. The second term, $Var_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})}[\mathbb{E}(y|\mathbf{x}, \boldsymbol{\theta})]$, computes the variance of $\mathbb{E}(y|\mathbf{x}, \boldsymbol{\theta})$ for $\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})$. The expectation operation, $\mathbb{E}(y|\mathbf{x}, \boldsymbol{\theta})$, essentially takes out the contribution by the data noise (ε). Therefore, this second term measures epistemic uncertainty. For

classification problems, similar expressions can be derived for the uncertainty metric of differential entropy, as demonstrated in some earlier work (see, for example, [70–72]).
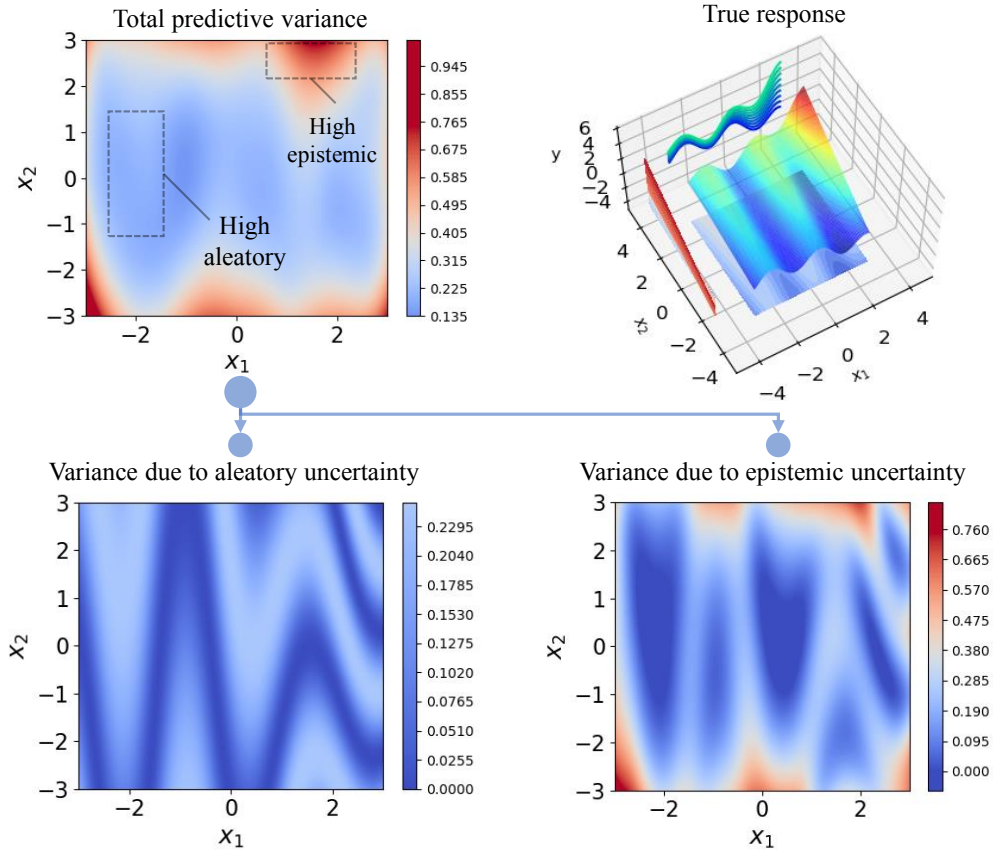

Figure 2: An example of uncertainty decomposition using variance-decomposition based method.

Figure 2 shows an example of uncertainty decomposition using the above variance decomposition method for a mathematical problem. The true model is a two-dimensional function as depicted in the top-right graph of Fig. 2 and this function has the following closed form: $y(\mathbf{x}) = \frac{1}{20}((1.5 + x_1)^2 + 4) \times (1.5 + x_2) - \frac{\sin 5 \times (1.5 + x_1)}{2}$. In this example, the true model is assumed to be unknown and needs to be learned from training data using an ML model. Due to inherent sensor noise, observational uncertainty is present in the output of the training data. It is modeled as a random variable following a Gaussian distribution as $\varepsilon(\mathbf{x}) \sim \mathcal{N}(0, \ 0.5|\sin(y(\mathbf{x}))|^2)$. Based on 50 training samples, a GPR model is constructed. The total predicted variance of the resulting ML model is shown in the upper left graph of Fig. 2. This graph shows that the predicted variance is high for some regions and low for others. Since both aleatory and epistemic uncertainty exists and only the total predictive uncertainty is visualized, it is difficult to tell if the uncertainty (the total predicted variance) in a certain region could be further reduced.

Decomposing the total predicted variance into variances due to aleatory uncertainty and epistemic uncertainty, respectively, as shown in the lower half of this figure, allows us to identify regions with high aleatory uncertainty and those with high epistemic uncertainty. If a region with high

epistemic uncertainty is the prediction region of interest, we can reduce the uncertainty to improve the prediction confidence of the ML model (see the uncertainty reduction methods in Sec. 2.3). However, if a region with high aleatory uncertainty and low epistemic uncertainty is the prediction region of interest, it would be difficult to further reduce the total predictive uncertainty. In that case, risk-based decision making needs to be employed to account for the irreducible aleatory uncertainty when deriving optimal decisions (see, for example, decision-making scenarios in engineering design, as discussed in Appendix B, and in PHM, as discussed in Sec. 5).

2.3. Reduction of epistemic uncertainty

As mentioned in Sec. 2.1, epistemic uncertainty is reducible. Suppose an ML model has low prediction accuracy and confidence due to high epistemic uncertainty, resulting in sub-optimal or even incorrect decisions. In that case, it is necessary to reduce the epistemic uncertainty. Commonly used strategies for the reduction of epistemic uncertainty can be roughly divided into the following two groups according to the source of epistemic uncertainty of interest.

(a) **Reducing parameter uncertainty**

    i **Adding more training data**: Having access to limited training data usually leads to uncertainty in ML model parameters. The model-parameter uncertainty is part of epistemic uncertainty. It can be reduced by increasing the training data size, e.g., via data augmentation using physics-based models [76] or simply by collecting and adding more experimental data to the training set. Let us assume the added training data is as clean as the existing data. In that case, the epistemic uncertainty component of the predictive uncertainty becomes smaller, while the aleatory uncertainty is expected to remain at a similar level. Suppose that, in a different case, the added training data contains more noise than the existing data. In that case, we still expect lower epistemic uncertainty in regions of the input space where the added data lie but higher aleatory uncertainty in these regions.

    ii **Adding physics-informed loss or physical constraints for ML model training**: Incorporating physical laws as new loss terms or imposing physical constraints, such as boundedness, monotonicity, and convexity for interpretable latent variables for ML model training, may allow us to obtain a more accurate estimate of ML model parameters. Although this physics-informed/constrained ML approach may not directly reduce epistemic uncertainty in ML predictions, it helps to reduce the training data size required to build a robust ML model that produces accurate predictions across a wide range of input settings. Specifically, enforcing principled physical laws into an ML model considerably prunes the search space of model parameters as parameters violating these constraints are discarded immediately. As a result, physical constraints contribute to reducing parameter uncertainty to some extent by complementing the insufficient training data and narrowing down the feasible region of these parameters. This benefit becomes especially relevant when training data is lacking and has been reported in recent review papers in various

engineering fields, such as computational physics [77], digital twin [78], and reliability engineering [79], and in research papers published in recent special issue collections on health diagnostics/prognostics [80] and the broader topic of reliability and safety [81]. For over-parameterized ML models such as neural networks, it is possible to simultaneously reduce bias and variance in the model parameters [82]. For simpler models such as GPR, utilizing additional information such as gradient information [83], orthogonality [84], and monotonicity [85] as constraints in kernel construction can also improve the prediction accuracy.

iii **Adopting better strategies for ML model training**: If a better starting point can be used when training an ML model, the optimization process may yield a more accurate estimate of the model parameters. Similar to adding physics-informed loss terms, this strategy can also indirectly reduce epistemic uncertainty. A popular example of this strategy is transfer learning, where the model trained in one domain is used as a starting point for training a model in another domain (e.g., transfer of weights and biases in selected neural network layers) [86]. Another strategy is to use better optimization algorithms when the number of parameters to be optimized is large. Global optimization in high-dimensional search spaces is always challenging. Algorithms such as stochastic gradient descent can have better convergence than traditional quasi-Newton methods in training deep neural networks [87]. Reformulating model training with multiple loss terms as minimax problems to adjust the focus of different loss terms can also improve convergence [88].

(b) **Reducing model-form uncertainty**

i **Identifying better input features**: In practical applications, an important step in training ML models is the selection of input features with strong predictive power according to domain knowledge, expert opinions, or exploratory analysis [89, 90]. Identifying input features with higher predictive power and using them as input features allows us to reduce the model-form uncertainty of ML models.

ii **Choosing better model architecture/kernel functions**: All models are wrong, but some are useful [91]. An appropriately chosen model architecture can better approximate the true underlying function than many other model architectures. A commonly used method is, therefore, to choose better model architecture or kernel function through tuning or model validation. It can reduce model-form uncertainty to some degree.

iii **Adding high-fidelity data**: An obvious way to reduce model-form uncertainty caused by bias in the training data is by adding high-fidelity data, thereby reducing the overall epistemic uncertainty. Such strategies have been widely adopted in the ML field in the context of multi-fidelity surrogate modeling/ML [92–95] and transfer learning [96].

Next, we use the two-dimensional example given in Fig. 2 to illustrate the process of reducing epistemic uncertainty. As shown in Fig. 3, a group of training points is first generated from a known
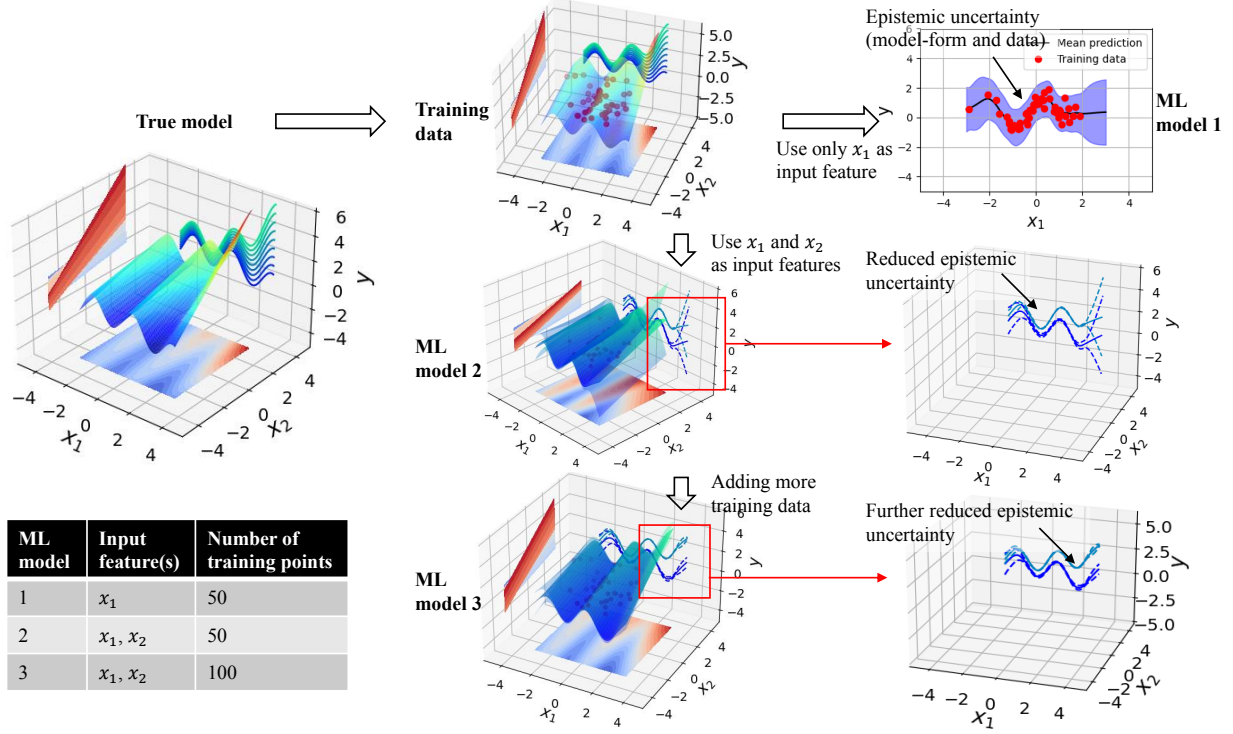
Figure 3: Types of uncertainty sources in ML models and the process of reducing epistemic uncertainty (i.e., methods (b).i and (a).i described in Sec. 2.3).

mathematical function. Then, an ML model with only x_1 as the input feature is constructed based on this group of training data. As shown in this figure, the resulting ML model (i.e., *ML model 1*) has considerable epistemic uncertainty due to the combined effect of model-form uncertainty and model-parameter uncertainty. In particular, the model-form uncertainty is caused by the fact that the underlying model used to generate this dataset has two input variables (x_1 and x_2) while ML model 1 only uses x_1 as its input feature. Model-parameter uncertainty stems from the limited number of training samples (i.e., 50 in this example). In order to reduce the epistemic uncertainty (model-form uncertainty), we then include both x_1 and x_2 as the input features, and another ML model labeled *ML model 2* is constructed using the same group of training data. As illustrated in Fig. 3, adding input feature x_2 (i.e., strategy (b).i as described above) substantially reduces the epistemic uncertainty in regions within the training sample distribution. If we increase the size of the training data to 100 (i.e., strategy (a).i), a third ML model (*ML model 3*) can be built based on this larger training dataset. As expected, the epistemic component of the predictive uncertainty is shown to decrease further due to the reduction of model-parameter uncertainty.

3. Methods for UQ of ML models

Data-driven ML models, most notably neural networks, have demonstrated unprecedented performance in establishing associations and correlations from large volumes of data in high-dimensional space via multiple layers of neurons and activation functions stacked together [97]. While ML has

progressed on a fast track, it is still far away from fulfilling the stringent conditions of mission-critical applications [15, 98], such as medical diagnostics, self-driving, and health prognostics of critical infrastructures, where safety and correctness concerns are salient. In addition to safety and reliability concerns, we are only able to collect a limited amount of data to train an ML model in a broad range of applications due to practical constraints on physical experiments and computational resources. To address some of these challenges, it is of paramount importance to establish principled and formal UQ approaches so that we can quantitatively analyze the uncertainty in ML model predictions arising from scarce and noisy training data as well as model parameters and structures in a sound manner. Accurate quantification of uncertainty in ML model predictions substantially facilitates the risk management of ML models in high-stakes decision-making environments [99–102].

In particular, when dealing with input samples in the region of input space with low signal-to-noise ratios or when handling the so-called OOD samples (input points sampled from a distribution very different from the training distribution), most ML models are prone to produce erroneous predictions [103]. If the uncertainty of an ML model can be quantified appropriately, it could lead to more principled decision making by enabling ML models to automatically detect samples for which there is high uncertainty. In fact, principled ML models are expected to yield high uncertainty (low confidence) in their predictions when the ML model predictions are likely to be wrong [104, 105]. Having uncertainty estimates that appropriately reflect the correctness of predictions is essential to identifying these "difficult-to-predict" samples that need to be examined cautiously, possibly with the eyes of a domain expert. This section provides a detailed, tutorial-style introduction of state-of-the-art methods for estimating the predictive uncertainty of data-driven ML models. As graphically summarized in Fig. 4, these UQ methods are GPR (Sec. 3.1), Bayesian neural network (BNN) (Sec. 3.2), neural network ensemble (Sec. 3.3), and deterministic methods focusing on SNGP (Sec. 3.4).

3.1. Gaussian process regression

GPR can be viewed as a generalized Bayesian inference, extending from an inference about a finite set of random variables to an inference about functions (each being an infinite-dimensional vector of random variables) [106]. This generalized Bayesian inference works with a joint probability distribution of a random function (i.e., an infinite-dimensional random vector) rather than a joint distribution of a finite-dimensional random vector. Comprehensive and critical reviews are provided by Rasmussen [106], Brochu et al. [107], and Shahriari et al. [31]. For complete details about GPR, readers are referred to the seminal textbook by Rasmussen [106].

3.1.1. Basics of Gaussian process regression

a. Introduction to Gaussian process and Gaussian process prior.

A Gaussian process is a collection of random variables over some domain, where any finite subset of these variables follows a joint (multivariate) Gaussian distribution. Intuitively, the Gaussian process also defines a probability distribution for an unknown function, and this function comprises a collection of (infinitely many) random variables. Let $f(\mathbf{x})$ be the unknown function, where $\mathbf{x} \in \mathbb{R}^D$ is a D-dimensional input, then for any finite set of input ($\mathbf{x}$) points of this function, for example,
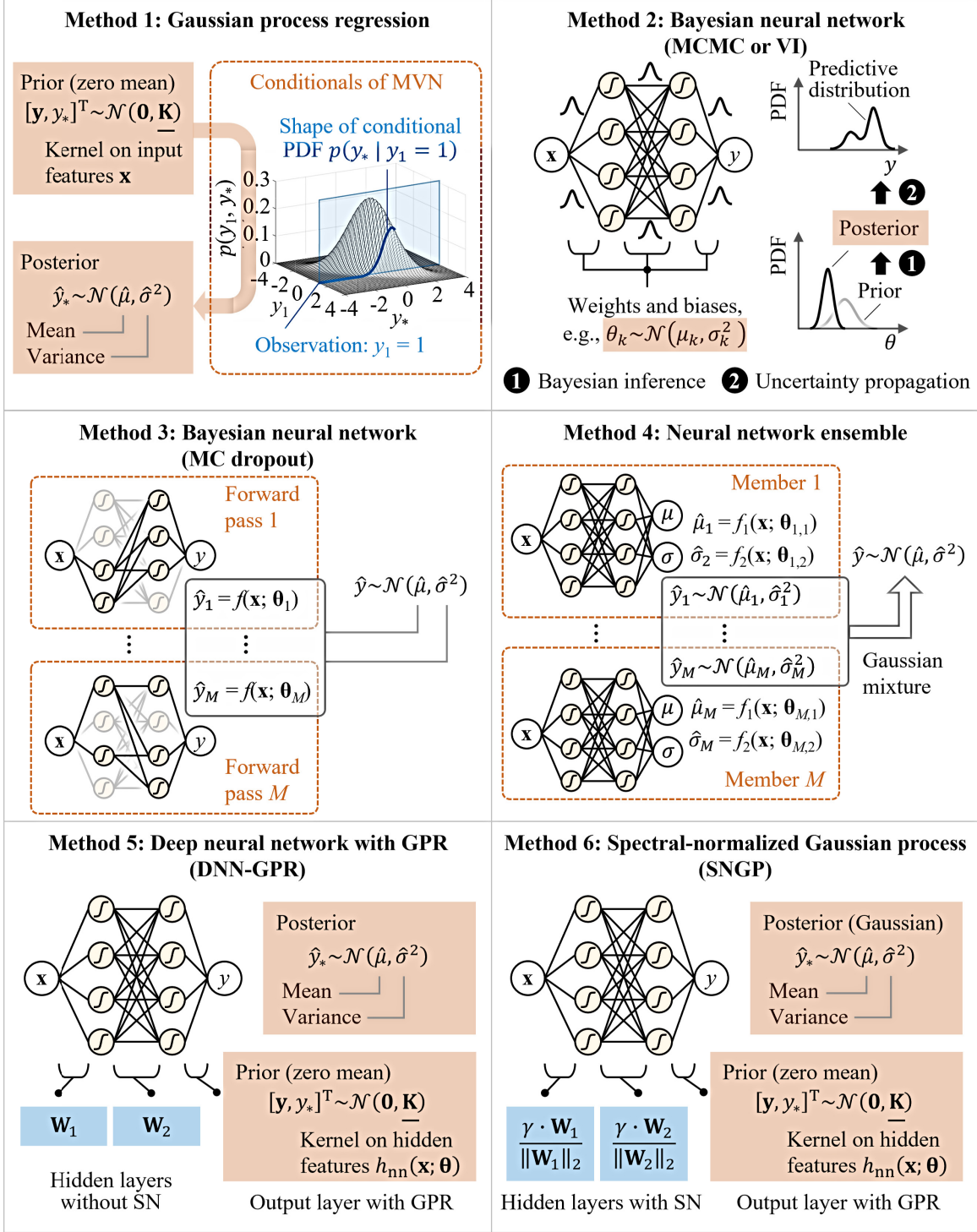
Figure 4: Graphical comparison of six state-of-the-art UQ methods introduced in Sec. 3. These methods are GPR (method 1), BNN via MCMC or VI (method 2), BNN via MC dropout (method 3), neural network ensemble (method 4), DNN with GPR – DNN-GPR (method 5), and SNGP (method 6). In method 1, MVN standards for the multivariate normal distribution, or equivalently, the multivariate Gaussian distribution used in the main text. In methods (5) and (6), SN stands for spectral normalization.

$\mathbf{X}_t = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T \in \mathbb{R}^{N \times D}$, their corresponding outputs $f(\mathbf{X}_t) = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]^T \in \mathbb{R}^N$ follow a joint Gaussian distribution.

GPR starts from a Gaussian process prior for the unknown function: $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ [106]. This Gaussian process prior is fully characterized by a (prior) mean function $m(\mathbf{x}) : \mathbb{R}^D \mapsto \mathbb{R}$ and a (prior) covariance function $k(\mathbf{x}, \mathbf{x}') : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$. The mean function $m(\mathbf{x})$ defines the prior mean of f at any given input point $\mathbf{x}$, i.e.,

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]. \tag{8}$$

The prior mean of the Gaussian process is often set as zero everywhere, $m(\mathbf{x}) = 0$, for the ease of computing the posterior. If the prior mean is a non-zero function, a trick is subtracting the prior means from the observations and function means (which we want to predict), thereby maintaining the "zero-mean" condition. The covariance function $k(\mathbf{x}, \mathbf{x}')$, also called the *kernel* in GPR, captures how the function values at two input points, $\mathbf{x}$ and $\mathbf{x}'$, linearly depend on each other. It takes the following form

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))\left(f(\mathbf{x}') - m(\mathbf{x}')\right)\right]. \tag{9}$$

When the prior mean is zero, the kernel fully defines the shape (e.g., smoothness and patterns) of functions sampled from the prior and posterior.

b. Kernel (covariance function).

Probably the most commonly used kernel is the squared exponential kernel (a.k.a. the radial basis function kernel and the Gaussian kernel), defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2l^2}\right). \tag{10}$$

where the two kernel parameters, or two hyperparameters of the GPR model, are the signal amplitude σ_f (σ_f^2 is called signal variance) and length scale l. σ_f^2 sets the upper limit of the prior variance and covariance and should take a large value if $f(\mathbf{x})$ spans a large range vertically (along the y-axis). It can be observed that the covariance between $f(\mathbf{x})$ and $f(\mathbf{x}')$ decreases as $\mathbf{x}$ and $\mathbf{x}'$ get farther apart. When $\mathbf{x}$ is extremely far from $\mathbf{x}'$, they have a very large Euclidian distance, and thus, $k(\mathbf{x}, \mathbf{x}') \approx 0$, i.e., the covariance between their function values approaches 0. Therefore, when predicting f at a new input point, observations far away in the input space will have a minimum influence. When a new input is OOD, it has a very low covariance with any training point, meaning that the training observations contribute minimally to reducing the prior variance of the function value at the OOD point, leading to high epistemic uncertainty. This kernel-enabled characteristic has important implications for the *distance awareness* property of GPR. On the other extreme, if two input points are extremely close, i.e., $\mathbf{x} \approx \mathbf{x}'$, then $k(\mathbf{x}, \mathbf{x}')$ becomes very close to its maximum, meaning $f(\mathbf{x})$ and $f(\mathbf{x}')$ have an almost perfect correlation. Function values of neighbors being highly correlated ensures smoothness in the GPR model, which is desirable because we often want to fit smooth functions to data.

The squared exponential kernel in Eq. (10) uses the same length scale l across all D dimensions. An alternative approach is to assign a different length scale l_d for each input dimension x_d, known as *automatic relevance determination (ARD)* [108]. The resulting *ARD squared exponential kernel* takes the following form

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - x_d')^2}{l_d^2}\right),$$ (11)

where the $(D+1)$ kernal parameters are the D length scales, $l_1, \ldots, l_D$, and the signal amplitude, σ_f. The ARD squared exponential kernel is also known as the *anisotropic variant* of the (isotropic) squared exponential kernel. Each length scale determines how relevant an input variable is to the GPR model. If l_d is learned to take a very large value, the corresponding input dimension x_d is deemed irrelevant and contributes minimally to the regression. It is worth noting that the squared exponential kernel is a special case of a more general class of kernels called Matérn kernels. See Appendix A.1 for an extended discussion of kernels.

c. Drawing random sample functions.

After defining a mean and a covariance function (kernel), we can draw sample functions from the Gaussian process prior without any observations of the function output. We can also sample function values from the Gaussian process posterior (i.e., the conditional Gaussian process conditioned on observed data), an essential task in GPR. Let us look at sampling functions from a Gaussian prior; a similar process can be followed to draw samples from a Gaussian process posterior. It is practically impossible to generate a perfectly continuous function from the prior, simply because this continuous function theoretically consists of an infinitely sized vector, which is not possible to sample. Alternatively, we can sample function values at a finite, densely populated set of input points and use these function values to reasonably approximate the continuous function. This approximation is acceptable in practice, given that we only need to predict f at a finite set of input points. Since a Gaussian process entails this finite collection of random variables (i.e., the f values at the finite set of input points) follow a multivariate Gaussian distribution, we can conveniently sample the function values from multivariate Gaussian.

Suppose we wish to sample function values at N_* input points, $\mathbf{x}_1^*, \ldots, \mathbf{x}_{N_*}^*$, from the prior. These input points could become new, unseen test points in a regression setting, and we use a subscript/superscript asterisk to distinguish them from training points. We start by defining an N_*-by-D matrix $\mathbf{X}_*$ where each row contains an input point, i.e., $\mathbf{X}_* = [\mathbf{x}_1^*, \ldots, \mathbf{x}_{N_*}^*]^{\mathrm{T}}$. For simplicity, we assume the multivariate Gaussian prior has zero means ($m(\mathbf{x}) = 0$), so we only need to obtain the covariances between the function values at these N_* input points. Using the squared exponential kernel, we can derive the following covariance matrix

$$\mathbf{K}_{\mathbf{X}_*, \mathbf{X}_*} = \begin{bmatrix} k(\mathbf{x}_1^*, \mathbf{x}_1^*) & \cdots & k(\mathbf{x}_1^*, \mathbf{x}_{N_*}^*) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_{N_*}^*, \mathbf{x}_1^*) & \cdots & k(\mathbf{x}_{N_*}^*, \mathbf{x}_{N_*}^*) \end{bmatrix}.$$ (12)

Now we can draw random samples of the function values at the N_* input points $\mathbf{X}$ from
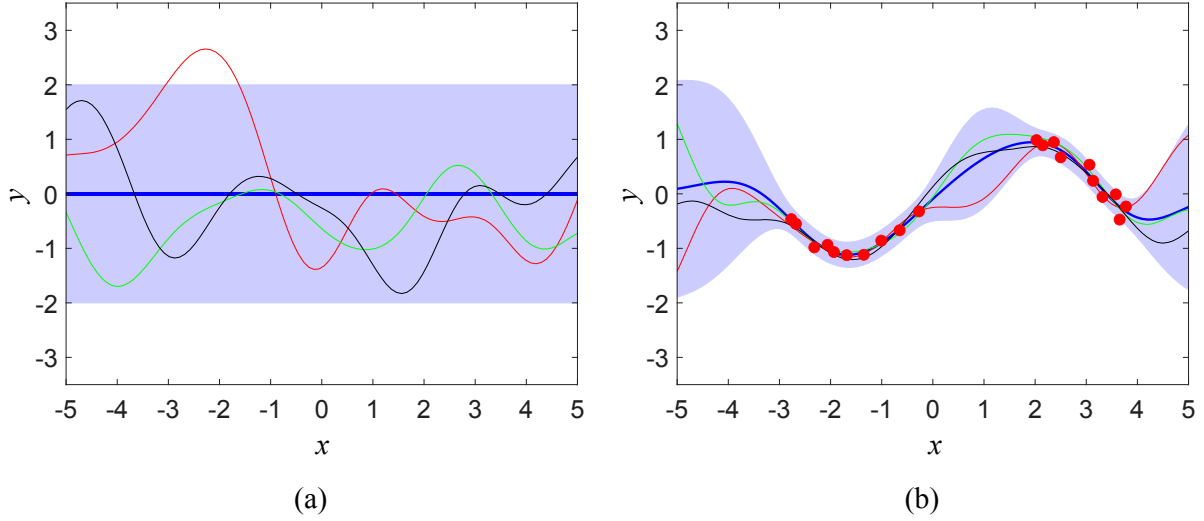
Figure 5: Sample functions drawn a Gaussian process prior (a) and posterior (b). The GPR model uses the squared exponential kernel with a length scale (l) of 1 and a signal amplitude (σ_f) of 1, and a Gaussian observation model with a noise standard deviation (σ_ε) of 0.1. The means are shown collectively as a solid blue line/curve, and ~95% confidence intervals (means plus and minus two standard deviations) are shown collectively as a light blue shaded area. 20 training observations are generated by corrupting a sine function with a white Gaussian noise term, $y = \sin(0.9x) + \varepsilon$ with $\varepsilon \sim \mathcal{N}\left(0, 0.1^2\right)$; these observations are shown as red dots.

$\mathcal{GP}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}'))$ by sampling from the following multivariate Gaussian distribution: $\mathbf{f}_* \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_{\mathbf{X}_*, \mathbf{X}_*})$. Each sample ($\mathbf{f}_*$) consists of N_* function values, i.e., $\mathbf{f}_* = f(\mathbf{X}_*) = [f(\mathbf{x}_1^*), \dots, f(\mathbf{x}_{N_*}^*)]^{\mathrm{T}}$. The most common numerical procedure to sample from a multivariate Gaussian distribution consists of generate random samples (vectors) from the multivariate (D-dimensional) standard normal distribution, $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and (2) transform these random samples linearly based on the mean vector of the target multivariate Gaussian and the Cholesky decomposition of its covariance matrix (see further details in Sec. A.2 (Gaussian Identities) of Ref. [106]). Figure 5(a) shows three sample functions randomly drawn from a Gaussian process prior.

d. Making predictions at new points.

In practice, often, we only have noisy observations of $f(\mathbf{x})$, for example, through the following Gaussian observation model:

$$y = f(\mathbf{x}) + \varepsilon, \tag{13}$$

where ε is a zero-mean Gaussian noise, i.e., $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$. The above additive Gaussian form will also be commonly used for other UQ methods in the upcoming sections. The N noisy observations can be conveniently written in a vector form: $\mathbf{y}_t = [y_1, \dots, y_N]^{\mathrm{T}} \in \mathbb{R}^N$. Note that these observations are sometimes called *targets* in a regression setting. In GPR, we want to infer the input ($\mathbf{x}$) - target (y) relationship from the noisy observations; we may also be interested in learning the input ($\mathbf{x}$) - output (f) relationship in some cases.

The Gaussian observation model in Eq. (13) portrays an observation as two components: a *signal term* and a *noise term*. The signal term $f(\mathbf{x})$ carries the *epistemic uncertainty* (see Sec.

2.1) about $f(\mathbf{x})$, which can be reduced with additional observations of f at a finite set of training points (e.g., $\mathbf{x}_1, \ldots, \mathbf{x}_N$). The noise term ε represents the inherent mismatch between signal and observation (e.g., due to measurement noise; see Table 1), which is a type of *aleatory uncertainty* (see Sec. 2.1) and cannot be reduced from additional observations. In some cases, observations may be noise-free, corresponding to a special case where $\sigma_\varepsilon = 0$. In other words, we have access to the true function (f) output in these cases.

Now it is time to look at how to make predictions of function values $\mathbf{f}_*$ for N_* new, unseen input points $\mathbf{X}_*$, given a collection of training observations, $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$, equivalently expressed as $\mathcal{D} = \{\mathbf{X}_t, \mathbf{y}_t\}$. These predictions can be made by drawing samples from the Gaussian process posterior, $p(f|\mathcal{D})$. We denote the function values at the training inputs as $\mathbf{f}_t = f(\mathbf{X}_t) = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]^T$. Again, according to the definition of a Gaussian process, the function values at the training inputs and those at the new inputs are jointly Gaussian (prior without using observations), written as

$$\begin{bmatrix} \mathbf{f}_t \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t} & \mathbf{K}_{\mathbf{X}_t,\mathbf{X}_*} \\ \mathbf{K}_{\mathbf{X}_*,\mathbf{X}_t} & \mathbf{K}_{\mathbf{X}_*,\mathbf{X}_*} \end{bmatrix}\right), \tag{14}$$

where $\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t}$ is the covariance matrix between the f values at the training points, expressed by simply replacing $\mathbf{X}$ in Eq. (12) with $\mathbf{X}_t$, $\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_*}$ is the covariance matrix between the training points and new points (also called the cross-covariance matrix), $\mathbf{K}_{\mathbf{X}_*,\mathbf{X}_t} = \mathbf{K}_{\mathbf{X}_t,\mathbf{X}_*}^T$, and $\mathbf{K}_{\mathbf{X}_*,\mathbf{X}_*}$ is the covariance matrix between the new points.

As shown in the Gaussian observation model in Eq. (13), we assume all observations contain an additive independent and identically distributed (i.i.d.) Gaussian noise with zero mean and variance σ_ε^2. Under this assumption, the covariance matrix for the training *observations* needs the addition of the noise variance to each diagonal element, i.e., $\mathbf{y}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t} + \sigma_\varepsilon^2\mathbf{I})$, where $\mathbf{I}$ denotes the identity matrix of size N whose diagonal elements are ones and off-diagonal elements are zeros. It then follows that the training observations (known) and the function values at the new input points (unknown) follow a slightly revised version of the multivariate Gaussian prior shown in Eq. (14), expressed as

$$\begin{bmatrix} \mathbf{y}_t \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t} + \sigma_\varepsilon^2\mathbf{I} & \mathbf{K}_{\mathbf{X}_t,\mathbf{X}_*} \\ \mathbf{K}_{\mathbf{X}_*,\mathbf{X}_t} & \mathbf{K}_{\mathbf{X}_*,\mathbf{X}_*} \end{bmatrix}\right). \tag{15}$$

Now we want to ask the following question: "given the training dataset $\mathcal{D}$ and new test points $\mathbf{X}_*$, what is the posterior distribution of the new, unobserved function values $\mathbf{f}_*$?". It has been shown that conditionals of a multivariate Gaussian are also multivariate Gaussian (see, for example, Sec. 3.2.3 of the probabilistic ML book [74]). Therefore, the posterior distribution $p(\mathbf{f}_*|\mathcal{D}, \mathbf{X}_*)$ is multivariate Gaussian. The posterior mean $\bar{\mathbf{f}}_*$ and covariance $cov(\mathbf{f}_*)$ can be derived based on the well-known formulae for conditional distributions of multivariate Gaussian, leading to the following:

$$\bar{\mathbf{f}}_* = \mathbf{K}_{\mathbf{X}_t,\mathbf{X}_*}^T(\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{y}_t, \tag{16}$$

and

$$cov(\mathbf{f}_*) = \mathbf{K}_{\mathbf{X}_*,\mathbf{X}_*} - \mathbf{K}_{\mathbf{X}_t,\mathbf{X}_*}^{\mathrm{T}}(\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_*}. \tag{17}$$

It is worth noting that this posterior distribution is also a Gaussian process, called a Gaussian process posterior. So we have $f(\mathbf{x})|\mathcal{D} \sim \mathcal{GP}(m_{\mathrm{post}}(\mathbf{x}), k_{\mathrm{post}}(\mathbf{x}, \mathbf{x}'))$, where the mean and kernel functions of this Gaussian process posterior take the following forms:

$$m_{\mathrm{post}}(\mathbf{x}) = \mathbf{K}_{\mathbf{X}_t,\mathbf{x}}^{\mathrm{T}}(\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{y}_t, \tag{18}$$

and

$$k_{\mathrm{post}}(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - \mathbf{K}_{\mathbf{X}_t,\mathbf{x}}^{\mathrm{T}}(\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{K}_{\mathbf{X}_t,\mathbf{x}'}. \tag{19}$$

It can be observed from Eqs. (16) and (17) that the key to making predictions with a Gaussian process posterior is calculating the three covariance matrices, $\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t}$, $\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_*}$, and $\mathbf{K}_{\mathbf{X}_*,\mathbf{X}_*}$. Difficulties in computation usually arise when performing a matrix inversion on a large covariance matrix $\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t}$ with many training observations. Much effort has been devoted to solving this matrix inversion problem, resulting in many approximation methods, such as covariance tapering [109, 110] and low-rank approximations [111, 112], mostly applied to handle large spatial datasets. Another important issue associated with the matrix inversion is that the covariance matrix could become ill-conditioned, most likely due to some training points being too close and providing redundant information. Two common strategies to invert an ill-conditioned covariance matrix are (1) performing the Moore–Penrose inverse or pseudoinverse using the singular value decomposition [30] and (2) applying "nugget" regularization, i.e., adding a small positive constant (e.g., 10^{-6}) to each diagonal element of the covariance matrix to make it better conditioned while having a negligible effect on the calculation [113, 114]. Oftentimes, adding the variance of the Gaussian noise σ_ε^2, as shown in Eqs. (16) and (17), serves the purpose of "nugget" regularization.

Following the numerical procedure described in Sec. 3.1.1.c, we can generate random samples of $\mathbf{f}$ from the Gaussian process posterior. For example, we can sample function values at the N_* input points, $\mathbf{x}_1^*, \dots, \mathbf{x}_{N_*}^*$, by sampling from a multivariate Gaussian with mean $\bar{\mathbf{f}}_*$ and covariance $cov(\mathbf{f}_*)$. It is possible that the Cholesky decomposition needs to be performed on an ill-conditioned posterior covariance matrix $cov(\mathbf{f}_*)$. This issue can be tackled by applying "nugget" regularization or adopting an alternative sampling procedure that centers around defining and sampling from a zero-mean, unconditional Gaussian process, as described in Refs. [115–117]. Figure 5(b) shows three sample functions drawn from a Gaussian process posterior after collecting 20 noisy observations of a 1D sine function.

We have been looking at the posterior of noise-free function values. To derive the posterior over the noisy observations, $p(\mathbf{y}_*|\mathcal{D}, \mathbf{X}_*)$, we add a vector of i.i.d. zero-mean Gaussian noise variables to the $\mathbf{f}_*$ posterior, producing a multivariate Gaussian with the same means (Eq. (16)) and a different covariance matrix whose diagonal elements increase by σ_ε^2 compared to the covariance matrix in Eq. (17). It is also straightforward to make predictions on a noise-free Gaussian process using Eqs. (16) and (17). We can simply take out the noise variance term $\sigma_\varepsilon^2\mathbf{I}$ and use $\mathbf{y}_* = \mathbf{f}_*$. As is discussed in

[Appendix B](#), GPR with noise-free observations is widely used to build cheap-to-evaluate surrogates of computationally expensive computer simulation models in engineering design applications such as model calibration, reliability analysis, sensitivity analysis, and optimization. The observations in these applications are free of noise because we have direct access to the true underlying function (i.e., the computer simulation model) that we want to approximate. In contrast, as will be discussed in Sec. [5](#), many applications of GPR in health prognostics require the consideration of noisy observations, as we often do not have access to the true targets (e.g., health indicator) but can only obtain noisy measurements or estimates of these targets.

Now let us look back at the distance awareness property of GPR. Suppose a new input point $\mathbf{x}_*$ keeps moving away from the training distribution $\mathcal{D}$. In that case, the Euclidean distance between $\mathbf{x}_*$ and any input point $\mathbf{x}_i$ in $\mathcal{D}$, i.e., $dist(\mathbf{x}_*, \mathbf{x}_i), \forall i = 1, \ldots, N$, constantly increases. All elements in the cross-covariance matrix and, more strictly, the cross-covariance vector $\mathbf{k}_{\mathbf{X}_t, \mathbf{x}_*} = [k(\mathbf{x}_1, \mathbf{x}_*), \ldots, k(\mathbf{x}_N, \mathbf{x}_*)]^T$ quickly approach zero. Given that neither the training-data covariance matrix $\mathbf{K}_{\mathbf{X}_t, \mathbf{X}_t}$ nor the new-data covariance (variance in this case) $k(\mathbf{x}_*, \mathbf{x}_*)$ experiences any changes, the posterior mean $\overline{f}_*$ will approach zero (i.e., the prior mean), and more importantly, the posterior variance $var(f_*)$ will approach its maximum allowed value σ_f^2. This observation of the GPR model behavior is significant for UQ because it means that a GPR model naturally yields high-uncertainty predictions for OOD samples falling outside of the training distribution.

e. Optimizing hyperparameters.

Suppose we choose the squared exponential kernel as the covariance function. In that case, we will have three unknown hyperparameters that need to be estimated based on training data. These parameters are the characteristic length scale (l), signal amplitude (σ_f), and noise standard deviation (σ_ε), i.e., $\boldsymbol{\theta} = [l, \sigma_f, \sigma_\varepsilon]^T$. Estimating these hyperparameters can be regarded as *training* a GPR model. As it is often difficult yet not much value-added to obtain the full Bayesian posterior of $\boldsymbol{\theta}$, we typically choose to obtain a *maximum a posteriori probability (MAP)* estimate of $\boldsymbol{\theta}$, a point estimate at which the log marginal likelihood $\log p(\mathbf{y}_t | \mathbf{X}_t, \boldsymbol{\theta})$ reaches the largest value. Assuming the prior is uniform, the log marginal likelihood function of the posterior takes the following form [106]:

$$\log p(\mathbf{y}_t | \mathbf{X}_t, \boldsymbol{\theta}) = \underbrace{-\frac{1}{2}(\mathbf{y}_t^T(\mathbf{K}_{\mathbf{X}_t, \mathbf{X}_t} + \sigma_\varepsilon^2 \mathbf{I})^{-1}\mathbf{y}_t}_{\text{Model-data fit}} \underbrace{-\frac{1}{2}\log|\mathbf{K}_{\mathbf{X}_t, \mathbf{X}_t} + \sigma_\varepsilon^2 \mathbf{I}|}_{\text{Complexity penalty}} \underbrace{-\frac{N}{2}\log(2\pi)}_{\text{Constant}}, \qquad (20)$$

The first term on the right-hand side, the so-called "model-data fit" term, quantifies how well the model fits the training observations. The second term, called the "complexity penalty" term, quantifies the model complexity where a smoother covariance matrix with a smaller determinant is preferred [106]. The third and last term is a normalization constant and indicates that the likelihood of data tends to decrease as the training data size increases [31]. It should be noted that the cost complexity of Eq. [(20)](#) is $\mathcal{O}(N^3)$ to compute the inverse of the covariance matrix $\mathbf{K}_{\mathbf{X}_t, \mathbf{X}_t}$ and the space complexity is $\mathcal{O}(N^2)$ to store this matrix. Hyperparameter optimization significantly influences the accuracy of GPR. See [Appendix A.2](#) for an illustrated example on the

effect of hyperparameter optimization.

3.1.2. UQ capability and some limitations of Gaussian process regression

GPR is capable of capturing both aleatory and epistemic uncertainty. For regression problems, the posterior variance for a query or test point, shown in Eq. (17), is an elegant expression of the total predictive uncertainty. The variance of the additive white noise, σ_ε^2, is a measure of the aleatory uncertainty. If this noise variance is assumed to be a constant (learned from the observations), the GPR model is called a "homoscedastic" model. In contrast, a heteroscedastic GPR model represents the noise variance as a function of the input variables $\mathbf{x}$ [118]. Assuming a squared exponential kernel is used, the epistemic uncertainty is determined mainly by $\mathbf{k}_{\mathbf{X}_t,\mathbf{x}_*}$, the covariance vector between the training points and query point, as discussed at the end of Sec. 3.1.1.d. The farther away the query point is from the training points, the smaller the elements of $\mathbf{k}_{\mathbf{X}_t,\mathbf{x}_*}$ and the larger the epistemic uncertainty. Therefore, using a distance-based covariance (or kernel) function and according to conditionals of a multivariate Gaussian, a GPR model produces low epistemic uncertainty at query or test points close to observations used for training and high epistemic uncertainty at query points far away from any training observation. This distance awareness property makes GPR an ideal choice for highly reliable OOD detection for problems of low dimensions and small training sizes. The aleatory and epistemic uncertainty components of the posterior variance determine how wide the confidence interval of a model prediction at the query point should be, reflecting the total predictive uncertainty.

Despite the highly desirable distance awareness property and OOD detection capability, GPR does not always produce posterior variances that reliably measure the predictive uncertainty. The reliability of UQ by GPR depends on many factors, such as the test point where a prediction is made, the behavior of the underlying function to be fitted, and the choices of the kernel and hyperparameters. For example, a necessary condition for reliable UQ by a GPR model is properly choosing its kernel and optimizing the resulting hyperparameters (e.g., the variance of the additive white noise, σ_ε^2, measures the aleatory uncertainty and should be optimized for accurate UQ). As discussed earlier, GPR can detect OOD test points, especially those far from the training distribution. However, the high posterior variances at these "extreme" test points may still not accurately measure the prediction accuracy. Specifically, as a test point moves away from the training distribution, the posterior variance will start to "saturate" at its peak value, as discussed in detail in Sec. 3.1.1.d; in contrast, the prediction error at this test point may continue to rise due to an increasing degree of extrapolation, and so should an "ideal" estimate of the predictive uncertainty. Although GPR may not produce reliable UQ in such an extreme extrapolation scenario, it is important to take a step back and keep in mind that extrapolating to an extensive degree goes against the purpose for which GPR was originally introduced, i.e., interpolation [106, 119].

Standard GPR generally does not scale well to large training datasets (large N) because its training complexity is $\mathcal{O}(N^3)$. This scalability issue originates from the computation of the inverse and determinant of the $N \times N$ covariance matrix $\mathbf{K}_{\mathbf{X}_t,\mathbf{X}_t}$ during model training (i.e., hyperparameter optimization), as shown in Eq. (20). This scalability issue motivated considerable effort in

examining local and global approximation methods to scale GPR to large training datasets while maintaining prediction accuracy and UQ quality. Interested readers may refer to a recent review on scalable GPR in [120]. Another limitation of GPR is its lack of scalability to high input dimensions (high D). This limitation stems from two issues. First, training a GPR model in a high-dimensional input space typically requires optimizing a large number of hyperparameters. This is because an ARD kernel form often needs to be chosen to deal with high-dimensional problems. As a result, the number of hyperparameters increases linearly with the number of input variables (e.g., a GPR model with the ARD squared exponential kernel shown in Eq. (11) has $(D + 2)$ hyperparameters). A direct consequence is that a large quantity of training samples (high N) is needed to optimize the many hyperparameters, leading to a large covariance matrix. As discussed earlier, inverting this large covariance matrix and calculating its determinant have high computational complexity. Second, maximizing the log marginal likelihood (see Eq. (20)) with a large number of hyperparameters becomes a high-dimensional optimization problem. Solving this high-dimensional problem requires many function evaluations, each involving one-time covariance matrix inversion and determinant calculation. Attempts to improve GPR's scalability to high-dimensional problems include (1) projecting the original, high-dimensional input onto a much lower-dimensional subspace and building a GPR model in the subspace [121, 122], (2) defining a new kernel with a substantially smaller number of parameters identified with partial least squares [123], and (3) adopting an additive kernel in place of a tensor product kernel in Eq. (11) [124]. More detailed discussions on scaling GPR to high-dimensional problems can be found in a recent review [125].

As a final note, since this tutorial focuses on UQ of neural networks, it is relevant and interesting to discuss connections between GPR and neural networks. Considerable efforts have been made to establish such connections. Some of these efforts are briefly discussed in Appendix A.3.

3.2. Bayesian neural network

We will first introduce the non-Bayesian (frequentist) training of a DNN, and contrast it against the Bayesian training used to form the BNNs. Consider a DNN $f : \mathbb{R}^D \mapsto \mathbb{R}$ with tunable parameters $\boldsymbol{\theta}$, and its prediction at an $\mathbf{x}$ is written as $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$. In non-Bayesian (frequentist) training, $\boldsymbol{\theta}$ are treated as deterministic, but unknown, parameters (i.e. *not* random variables). An estimator for $\boldsymbol{\theta}$ can then be created from a training dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$ by minimizing a loss function shown below:

$$\boldsymbol{\theta}^\star = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}). \tag{21}$$

For example, a commonly used loss function for regression problems is the mean squared error (MSE) defined below:

$$\boldsymbol{\theta}^\star_{\mathrm{MSE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \frac{1}{N} \sum_{i=1}^{N} ||y_i - f(\mathbf{x}_i; \boldsymbol{\theta})||_2^2. \tag{22}$$

With the gradient of f accessible through back-propagation [126], the loss minimization is typi-

cally solved numerically using stochastic gradient descent [127, 128]. Once $\boldsymbol{\theta}^\star$ is found, prediction at a new point $\mathbf{x}_*$ can be made via $\hat{y}_* = f(\mathbf{x}_*; \boldsymbol{\theta}^\star)$. These predictions, however, are single-valued and do not have quantified uncertainty.

A Bayesian training [129–131] of DNNs, also known as Bayesian deep learning [70, 108, 132–134], produces a *Bayesian neural network* or *BNN*. The Bayesian approach views $\boldsymbol{\theta}$ as a random variable with the goal to find the entire distribution of plausible $\boldsymbol{\theta}$ values that could have generated the observed data $\mathcal{D}$. Following Bayes' rule, the prior probability density function (PDF) $p(\boldsymbol{\theta})$ ("before"-uncertainty in $\boldsymbol{\theta}$) is updated to the posterior PDF $p(\boldsymbol{\theta}|\mathcal{D})$ ("after"-uncertainty in $\boldsymbol{\theta}$) conditioned on the training data $\mathcal{D}$. Mathematically, we have:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} = \frac{p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X})p(\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X})}, \tag{23}$$

where we separate the training dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ into their inputs $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ and corresponding outputs $\mathbf{y} = \{y_1, y_2, \cdots, y_N\}$. Note that in the GPR section (Sec. 3.1), $\mathbf{X}_\mathrm{t}$ and $\mathbf{X}_*$ denote matrices comprising input points and $\mathbf{y}_\mathrm{t}$ and $\mathbf{y}_*$ denote *vectors* consisting of observations. In this BNN section, $\mathbf{X}$ and $\mathbf{y}$ denote *sets* of input points and observations, respectively, to be consistent with the literature on Bayesian inference and BNN. In the above, $p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X})$ is the likelihood and $p(\mathbf{y}|\mathbf{X})$ is the marginal likelihood (model evidence). The Bayesian problem and the BNN entail solving for the posterior $p(\boldsymbol{\theta}|\mathcal{D})$. We further discuss each term in the Bayes' rule in Eq. (23) below.

The prior $p(\boldsymbol{\theta})$ can be formed in an informative or non-informative manner. The former allows one to inject domain knowledge and expert opinions on the probable values of $\boldsymbol{\theta}$, formally through the methods of prior elicitation [135]. However, these methods are difficult to use on DNN parameters $\boldsymbol{\theta}$ due to their abstract and high-dimensional nature. The latter generates a prior following guiding principles for desirable properties (e.g., Jeffreys' prior [136], maximum entropy prior [137]). In practice, isotropic Gaussian is often adopted for their convenience, but caution must be taken to consider their pitfalls and appropriateness as BNN priors [138].

The likelihood $p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X})$ commonly follows a data (observation) model with an additive independent Gaussian noise (similar to Eq. (13) in the GPR case): $y_i = f(\mathbf{x}_i; \boldsymbol{\theta}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$. In the implementation, we often work with the log-likelihood, which is computed as:

$$\log p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X}) = \sum_{i=1}^{N} \log p(y_i|\boldsymbol{\theta}, \mathbf{x}_i) = -N \log(\sqrt{2\pi}\sigma_\varepsilon) - \frac{1}{2\sigma_\varepsilon^2} \sum_{i=1}^{N} ||y_i - f(\mathbf{x}_i; \boldsymbol{\theta})||_2^2. \tag{24}$$

We can see that finding the mode of the Gaussian (log)-likelihood above (i.e. the $\boldsymbol{\theta}$ that maximizes Eq. (24)) is equivalent to the MSE minimization in Eq. (22); hence, $\boldsymbol{\theta}_{\mathrm{MSE}}^\star$ is also known as the maximum likelihood estimator. Furthermore, adding a regularization term to Eq. (22) serves the role of a prior, and in a similar fashion, a regularized loss minimization is also known as a maximum a-posteriori (MAP) estimator (e.g., L2-regularization is the MAP with a Gaussian prior, L1-regularization is the MAP with a Laplace prior).

The marginal likelihood $p(\mathbf{y}|\mathbf{X})$ in the denominator of Eq. (23) is a (normalization) constant

for the posterior that integrates the numerator: $p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X})p(\boldsymbol{\theta}) \, d\boldsymbol{\theta}$. As it requires a non-trivial integration, this term is highly difficult to estimate. Fortunately, Bayesian computation algorithms are often designed to avoid the marginal likelihood altogether; we will describe examples of these algorithms in the upcoming sections.

Lastly, once the Bayesian posterior $p(\boldsymbol{\theta}|\mathcal{D})$ is obtained, the posterior uncertainty can be propagated through the BNN at a new point $\mathbf{x}_*$ via, for example, MC sampling. Importantly, we draw the distinction between the *posterior-pushforward* and *posterior-predictive* distributions. The posterior-pushforward is $p(\hat{y}_*|\mathbf{x}_*, \mathcal{D}) = p(f(\mathbf{x}_*; \boldsymbol{\theta})|\mathbf{x}_*, \mathcal{D})$. It describes the uncertainty on $\hat{y}_*$ (i.e. the "clean" prediction from the DNN) as a result of the uncertainty in $\boldsymbol{\theta}$. In contrast, the posterior-predictive is $p(y_*|\mathbf{x}_*, \mathcal{D}) = p([f(\mathbf{x}_*; \boldsymbol{\theta}) + \varepsilon]|\mathbf{x}_*, \mathcal{D})$, it describes the uncertainty on y_* (i.e. the noisy observed quantity). Hence, the former incorporates epistemic parametric uncertainty, while the latter further augments aleatory data uncertainty to the new prediction. The two distributions can be easily confused with each other, with the danger of improper UQ assessments where one might incorrectly expect the posterior-pushforward uncertainty to "capture" the noisy observation data.

In the following sections, we introduce several major types of Bayesian computational methods for solving the Bayesian posterior: Markov chain Monte Carlo or MCMC (posterior sampling), variational inference (posterior approximating), and MC dropout.

3.2.1. Markov chain Monte Carlo

The classical method for solving the Bayesian problem is to sample the posterior distribution using Markov chain Monte Carlo (MCMC) [139, 140]. MCMC establishes a Markov chain $\{\boldsymbol{\theta}^{(n)}\}, n \in \mathbb{N}$ from a transition kernel (i.e. proposal distribution) such that the chain converges to the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ regardless of its initial position $\boldsymbol{\theta}^{(0)}$. Most importantly, ergodicity theorems ensure that the empirical average of MCMC samples, $\frac{1}{N_s}\sum_{n=1}^{N_s} h(\boldsymbol{\theta}^{(n)})$, converges to the posterior expectation $\mathbb{E}_{\boldsymbol{\theta}|\mathcal{D}}[h(\boldsymbol{\theta})]$ almost surely. The most fundamental MCMC is the Metropolis-Hastings (MH) algorithm [141, 142], which forms the basis of many advanced MCMC variants. Hamiltonian Monte Carlo (HMC) [143, 144], an advanced type of MCMC with improved mixing properties, is more commonly used for BNNs. Drawing intuition from physics, HMC introduces an auxiliary momentum variable to form a system of Hamiltonian dynamics that can generate trajectories following the high-probability regions of the posterior (the so-called *typical set*). However, the effect of concentration of measure brings the typical set to become more singular with increasing dimension, and even HMC has only been exercised for $\boldsymbol{\theta}$ that is hundreds-dimensional [108, 145, 146]. This is still orders of magnitude shorter than modern DNNs that can easily have millions, even billions, of tunable parameters. While MCMC methods are theoretically appealing due to their asymptotic convergence to the true posterior, the Markov chains can be very difficult to mix for high dimensions in practice. As a result, they see limited usage in BNNs.

3.2.2. Variational Inference

A more scalable approach to the Bayesian inference problem can be found through variational inference (VI) [147, 148]. In contrast to MCMC sampling, the idea of VI is to approximate the posterior within a parametric family of distributions (e.g., a family of Gaussian distributions). In

this section, we will start by defining the optimization problem that describes the best posterior approximation, then introduce some examples of numerical algorithms to solve the VI problem.

Denoting a variational distribution (for approximating the posterior) using $q(\boldsymbol{\theta}; \lambda)$ parameterized by λ, VI seeks the best posterior-approximation $q(\boldsymbol{\theta}; \lambda^{\star})$ that minimizes the Kullback-Leibler (KL) divergence between $q(\boldsymbol{\theta}; \lambda)$ and $p(\boldsymbol{\theta}|\mathcal{D})$, that is:

$$\lambda^{\star} = \underset{\lambda}{\operatorname{argmin}} \ D_{\mathrm{KL}}\left[\, q(\boldsymbol{\theta}; \lambda) \,||\, p(\boldsymbol{\theta}|\mathcal{D}) \,\right]. \tag{25}$$

A popular choice for the variational distribution is the independent (mean-field) Gaussian: $q(\boldsymbol{\theta}; \lambda) = \prod_{k=1}^{K} q(\boldsymbol{\theta}_k; \lambda_k) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\theta}_k; \mu_k, \sigma_k^2)$, where K is the total number of parameters in the DNN. The independence structure allows the joint PDF to be factored into a product of univariate Gaussian marginals, and so the variational parameters are $\lambda = \{\mu_k, \sigma_k\}, k = 1, \ldots, K$ that encompasses the mean and standard deviation of each component of $\boldsymbol{\theta}$, for a total of $2K$ variational parameters. As a result, mean-field simplifies to a diagonal global covariance matrix (instead of dense covariance) in the approximate posterior, and it is unable to capture any correlation among the $\boldsymbol{\theta}_k$'s. More expressive representations of $q(\boldsymbol{\theta}; \lambda)$ are also possible, for example via normalizing flows [149] and transport maps [150] that parameterize the mapping from the posterior random variable $\boldsymbol{\theta}$ to a standard normal reference random variable.

Given the variational distribution $q(\boldsymbol{\theta}; \lambda)$, Eq. (25) can be further simplified as follows:

$$\begin{aligned}
\lambda^{*} &= \underset{\lambda}{\operatorname{argmin}} \ D_{\mathrm{KL}}\left[\, q(\boldsymbol{\theta}; \lambda) \,||\, p(\boldsymbol{\theta}|\mathcal{D}) \,\right] \\
&= \underset{\lambda}{\operatorname{argmin}} \ \mathbb{E}_{q(\boldsymbol{\theta}; \lambda)}\left[\ln q(\boldsymbol{\theta}; \lambda) - \ln\left(\frac{p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X})p(\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X})}\right)\right] \\
&= \underset{\lambda}{\operatorname{argmin}} \ \underbrace{D_{\mathrm{KL}}\left[\, q(\boldsymbol{\theta}; \lambda) \,||\, p(\boldsymbol{\theta}) \,\right] - \mathbb{E}_{q(\boldsymbol{\theta}; \lambda)}\left[\ln p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X})\right]}_{- \text{ Evidence Lower Bound (ELBO)}},
\end{aligned} \tag{26}$$

where going from the second to the third equation, the log-denominator's contribution $\mathbb{E}_{q(\boldsymbol{\theta}; \lambda)}\left[\ln p(\mathbf{y}|\mathbf{X})\right] = \ln p(\mathbf{y}|\mathbf{X})$ is omitted since it is constant with respect to both λ and $\boldsymbol{\theta}$ and its exclusion does not change the minimizer. The resulting expression in Eq. (26) is the negative of the well-known *Evidence Lower Bound (ELBO)*. The first term of ELBO acts as a regularization to keep $q(\boldsymbol{\theta}; \lambda)$ close to the prior. The second term of ELBO involves the log-likelihood of generating the observed data under DNN parameters $\boldsymbol{\theta} \sim q(\boldsymbol{\theta}; \lambda)$; hence it measures the expected model-data fit.

In general, it is impossible to evaluate the ELBO analytically, and Eq. (26) must be solved numerically. The simplest approach is to use MC sampling to estimate the ELBO, which only entails sampling $\boldsymbol{\theta} \sim q(\boldsymbol{\theta}; \lambda)$. Often, further simplifications can be made by analyically computing the first term, which involves only the prior and variational distribution. Furthermore, the gradient of ELBO with respect to λ may be derived (e.g., see [134] for Gaussian q) or obtained through automatic differentiation, allowing one to take advantage of gradient-based optimization algorithms (e.g., stochastic gradient descent) to solve Eq. (26).

The Stein variational gradient descent (SVGD) [151] is another VI variant offering a flexible

particle approximation to the posterior distribution. SVGD leverages the relationship between the gradient of the KL divergence in Eq. (25) to the Stein discrepancy, the latter which can be approximated using a set of particles. An update procedure can then formed to iteratively ascent along a perturbation direction $\boldsymbol{\theta}_i^{\ell+1} \leftarrow \boldsymbol{\theta}_i^\ell + \epsilon_\ell \hat{\varphi}^*(\boldsymbol{\theta}_i^\ell)$, where $\boldsymbol{\theta}_i^\ell$, $i = 1, \ldots, N_{\mathrm{p}}$, denotes the i-th particle at the ℓ-th iteration, ϵ_ℓ is the learning rate, and the perturbation direction is defined as:

$$\hat{\varphi}^*(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{p}}} \sum_{j=1}^{N_{\mathrm{p}}} \left[k(\boldsymbol{\theta}_j^\ell, \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}_j^\ell} \ln p(\boldsymbol{\theta}_j^\ell \mid \mathcal{D}) + \nabla_{\boldsymbol{\theta}_j^\ell} k(\boldsymbol{\theta}_j^\ell, \boldsymbol{\theta}) \right], \tag{27}$$

with $k(\cdot, \cdot)$ being a positive definite kernel (e.g., radial basis function kernel in Eq. (10)) Notably, the gradient of the log-posterior in the above equation can be evaluated via the sum of gradients of log-likelihood and log-prior, since the gradient of the log-marginal-likelihood with respect to $\boldsymbol{\theta}$ is zero. The overall effect is an iterative transport of a set of particles to best match the target posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$. Building upon the SVGD, advanced methods of Stein variational Newton [152, 153] that makes use of second-order (Hessian) information, and projected SVGD [154] that finds low dimensional data-informed subspaces, have also been proposed.
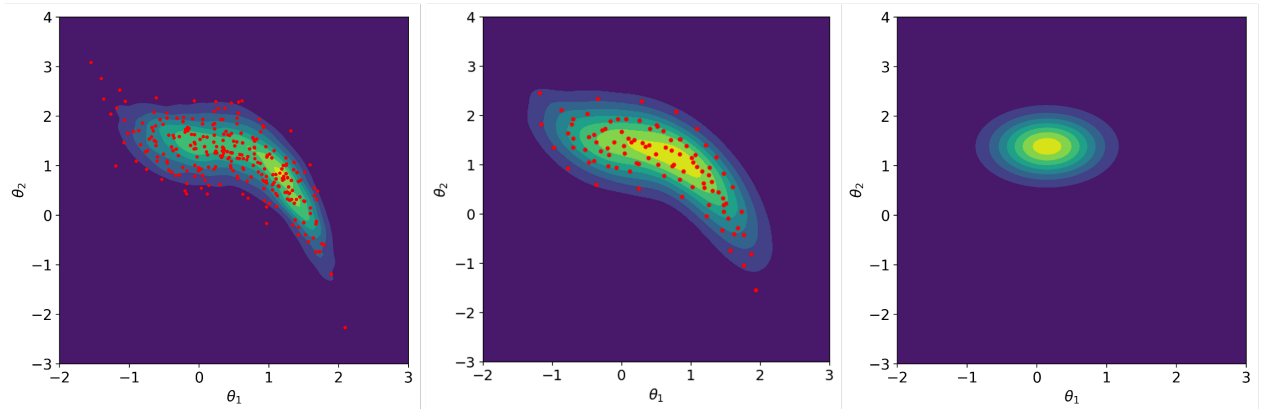

Figure 6: Illustration of Bayesian posterior obtained from (left) MCMC, (middle) SVGD, and (right) mean-field Gaussian VI for a simple low-dimensional Bayesian inference test problem.

Figure 6 compares the different Bayesian posteriors obtained from a simple low-dimensional Bayesian inference test problem using MCMC, SVGD, and mean-field Gaussian VI. MCMC and SVGD provide sample/particle representations of the posterior distribution, while VI produces an analytical Gaussian approximation of the PDF. Both MCMC and SVGD are able to capture non-Gaussian and correlated structure, although SVGD is more restrictive in the number of particles it can use due to higher memory requirement. However, SVGD and VI are more scalable to higher $\boldsymbol{\theta}$ dimensions than MCMC.

We note that another variant of VI can arise from the reverse KL divergence $D_{\mathrm{KL}}\left[\, p(\boldsymbol{\theta}|\mathcal{D}) \,\|\, q(\boldsymbol{\theta}; \lambda) \,\right]$ (in contrast to the $D_{\mathrm{KL}}\left[\, q(\boldsymbol{\theta}; \lambda) \,\|\, p(\boldsymbol{\theta}|\mathcal{D}) \,\right]$ from Eq. (25)). Notable algorithms from this formulation include expectation propagation [155], assumed density filtering [156], and moment matching [157]; in particular, expectation propagation has been shown to be quite effective in logistic-type models in general.

3.2.3. MC dropout

Although the Bayesian approach offers an elegant and principled way to model and quantify the uncertainty in neural networks, it typically comes with a prohibitive computational cost. As introduced earlier, MCMC and VI are two commonly used methods to perform Bayesian inference over the parameters of neural network. However, Bayesian inference with MCMC and variational inference in DNNs suffers from extremely time-consuming computational burden and poor scalability. Specifically, in the case of MCMC, estimating the uncertainty of neural network prediction with respect to a given input requires to draw a large number of samples from the posterior distributions of thousands or even millions of neural network parameters and propagate these samples through the neural network [158]. Compared with MCMC, VI is much faster and has better scalability as it recasts the inference of posterior distributions of neural network parameters as an optimization problem. However, VI unfortunately doubles the parameters to be estimated for the same neural network. In addition, it is intricate to derive and formulate the optimization problem, much less optimization regarding the high-dimensional problem consumes a large amount of time before convergence [21].

Beyond MCMC and VI, further scalability can be achieved through the MC dropout method. Initially proposed as a regularization technique to prevent the overfitting of DNNs [159], MC dropout has been shown to approximate the posterior predictive distribution under a particular Bayesian setup [21]. Procedurally, MC dropout follows the same deterministic DNN training in Eq. (21), except that it forms new sparsely connected DNNs from the original DNN (see method 3 in Fig. 4) by multiplying every weight with an independent Bernoulli random variable. Hence, each weight has some probability of becoming zero (i.e., the weight being dropped). These Bernoulli random variables are re-sampled (i.e., a new, randomized sparse DNN is formed) for every training sample and for every forward pass of the model. At test time, the prediction at a new point $\mathbf{x}_*$ can also be repeated with multiple forward passes each with a new, randomized sparse DNN resulting from the dropout operation. An ensemble of predictions can thus be obtained to estimate the uncertainty. Practical implementation of MC dropout in probabilistic programming languages is often realized by adding a dropout layer after each fully-connected layer.

The connection from MC dropout to a Bayesian setup is detailed in [21, 160]. Those works show that the loss function following the dropout procedure corresponds to a single-sample MC approximation to the VI objective (i.e., the ELBO in Eq. (26)), where the variational posterior of the DNN weights is a Bernoulli mixture of two independent Gaussians of fixed covariance. Furthermore, the prior of each DNN weight is assumed to follow a standard normal distribution, and the likelihood is based on the additive Gaussian noise model in Eq. (24). Established upon such a setup, in MC dropout, the variational distribution $q(\boldsymbol{\theta}; \lambda)$ for approximating the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ becomes a factorization over the weight matrices $\mathbf{W}_i$ of the layers 1 to L. Mathematically, the variational distribution $q(\boldsymbol{\theta}; \lambda)$ takes the following multiplicative form:

$$q(\boldsymbol{\theta}; \lambda) = \prod_{i=1}^{L} q_{\mathbf{M}_i}(\mathbf{W}_i), \tag{28}$$

where $q_{\mathbf{M}_i}(\mathbf{W}_i)$ denotes the density associated with the weight matrices $\mathbf{W}_i$ of layer i, and under MC dropout, it emerges as a Gaussian mixture model consisting of two independent Gaussian components with a fixed and identical variance, as shown below [21, 160]:

$$q_{\mathbf{M}_i}(\mathbf{W}_i) = p_i \underbrace{\mathcal{N}\left(\mathbf{M}_i, \sigma^2 \mathbf{I}_i\right)}_{\text{First Gaussian}} + (1 - p_i) \underbrace{\mathcal{N}\left(\mathbf{0}, \sigma^2 \mathbf{I}_i\right)}_{\text{Second Gaussian}} . \tag{29}$$

In the above, $\mathbf{M}_i$ is the mean of the first Gaussian, which is a vectorization of $n_{i-1} \times n_i$ values pertaining to the weight matrix $\mathbf{W}_i$ of size $n_{i-1} \times n_i$ (n_i denotes the number of units in the i-th layer; when $i = 0$, it denotes the number of inputs), σ is the standard deviation parameter specified by the end user, $\mathbf{I}_i$ is the identity matrix, $\mathcal{N}$ denotes the normal distribution, and p_i ($p_i \in [0, 1]$) is the dropout rate associated with the set of links connecting two consecutive layers of the neural network. Under this VI perspective, MC dropout corresponds to optimizing $\lambda = \mathbf{M}_i$, while both σ and p_i have fixed user-chosen values and are not part of the variational parameter set.

In the MC dropout implementation, for each element of $\mathbf{W}_i$, we sample a υ according to a Bernoulli distribution with a prescribed dropout rate p_i, that is $\upsilon \sim \text{Bernoulli}(p_i)$. If the binary variable $\upsilon = 0$, it indicates that link connecting the i-th and $(i+1)$-th layers is dropped out. This operation corresponds to choosing one of the two Gaussians from the mixture model in Eq. (29), and hence MC dropout can serve as an approximation to the Bayesian posterior in BNNs.

A major advantage of MC dropout is that it is very straightforward to implement, requiring only a few lines of modification to insert the $\boldsymbol{z}$'s to an existing DNN setup and often conveniently available as a dropout layer in many programming environments. Furthermore, its ease of implementation is agnostic of the neural network architecture, and can be readily adopted for many polular types of neural networks such as convolutional neural network (CNN) and recurrent neural network (RNN) [160, 161]. Another major advantage of MC dropout is its low computational cost and high scalability since its training procedure is effectively identical to an ordinary, non-Bayesian training of DNNs but with randomized sparse networks. These appealing properties collectively contribute to the growing popularity of MC dropout in practice.

MC dropout also has some limitations. One disadvantage is that the quality of the uncertainty generated by MC dropout is highly dependent on the choice of several hyperparameters [162–164], such as the dropout rate and number of dropout layers. Thus, these hyperparameters need to be fine tuned. Along this front, we also have similar findings in Section 3.5 that MC dropout exhibits poor stability to the dropout rate, training epochs, and the number of trainable network parameters (see Appendix D for more details). Regardless of the instability, the uncertainty produced by MC dropout exhibits a consistent difficulty in detecting OOD instances. Note that other approximation inference methods, such as MFVI, have a pathology that is slightly different from MC dropout with respect to the soundness of the quantified uncertainty, see Section 3.5 for more details. As highlighted by Foong et al. [165], the pathology of UQ in approximation methods is solely attributed to the restrictiveness of approximating family, while exact inference methods, such as MCMC, do not have such a problem. Another disadvantage of MC dropout is that users do not have the option to inject their prior knowledge by specifying the prior or likelihood function because there

is no mechanism for MC dropout to integrate such information—as a result, MC dropout can only represent a narrow spectrum of Bayesian problems. A further side effect of this limitation is that users may be hindered from critically thinking about the prior and likelihood altogether, which may lead to claims of a Bayesian solution without actually having a Bayesian problem setup. Finally, some researchers [166] have argued that MC dropout is not Bayesian because the variational distribution fails to converge to ground-truth posterior distribution on closed-form benchmarks.

3.3. Neural network ensemble

Ensemble learning is a well-established technique to prevent overfitting and mitigate the poor generalizability of ML models [167]. An essential step in constructing ensemble models is to train multiple individual models independently and aggregate predictions from these individual models to derive the final prediction. When building an ML model ensemble, it is of paramount importance to retain a high degree of diversity among the individual models to achieve desirable performance improvement [168]. Such diversity can be achieved through a broad spectrum of means that can be grouped into two principal categories: (1) randomization approaches, such as bagging (a.k.a. bootstrapping), where ensemble members are trained on different bootstrap samples of the original training set or a random subset of original features [169]; and (2) boosting approaches: boosting learns from the errors of previous iterations by increasing the importance of those wrongly predicted training instances, thus sequentially and incrementally constructing an ensemble [170].

In the context of deep learning, building an ensemble of neural networks entails independently training multiple neural networks with an identical architecture. Due to the easiness of implementation, neural network ensemble has been pervasively used to characterize the uncertainty of neural network predictions [78, 171]. In particular, well-calibrated uncertainty estimates tend to yield higher uncertainty on OOD data than on samples sufficiently similar to the distribution of training data. On this front, the uncertainty of a neural network ensemble is principled to some extent in the sense that this ensemble is inclined to produce higher uncertainty estimates (e.g., entropy in the case of classification problems) for OOD instances [22]. The appealing feature of neural network ensembles in producing higher uncertainty for OOD instances has been actively exploited s a prevailing means to detect dataset shifts in the ML community because the data collected under a shifted environment typically displays salient patterns that are substantially different from the data that the ensemble neural networks are trained with [22, 101, 172].

3.3.1. Aleatory uncertainty: training each network individually

We consider a popular configuration of neural network ensemble where each individual neural network in an ensemble outputs two quantities denoting the predicted mean $\widehat{\mu}(\mathbf{x}_i)$ and variance $\widehat{\sigma}^2(\mathbf{x}_i)$ with respect to an input $\mathbf{x}_i$ in its final output layer (see Fig. 4 for an overview on the architecture of the individual neural network). In this configuration, the predictive distribution of each network is often assumed to be Gaussian; therefore, the final output layer is sometimes called *Gaussian layer*. Such a configuration enables characterizing observational noise of aleatory nature associated with target values.

Let us take a closer look at the aleatory uncertainty, more specifically, the observational noise pertaining to each target observation. The simplest case is that we assume the same amount of noise or aleatory uncertainty for every input $\mathbf{x}_i$, also known as *homoscedasticity* or *homogeneity of variance* in statistics (similar to the homoscedastic case for GPR discussed in Sec. 3.1). To represent the relationship between input $\mathbf{x}_i$ and observation y_i, we can use the Gaussian observation model given in Eq. (13), substituting $\mathbf{x}$ with $\mathbf{x}_i$ and y with y_i. In this model, a random noise term ε, often modeled as a zero-mean Gaussian noise, shifts the target away from the true value $f(\mathbf{x}_i)$ to the observed value y_i. In this simplest case, the variance of random noise ε takes the same value σ_ε^2 for every input and is thus a constant. Although we could learn σ_ε together with the neural network parameters $\boldsymbol{\theta}$, this simplest case may not be realistic as some regions of the input space may have larger measurement noise than other regions.

A more realistic case is one where the noise variance depends on $\mathbf{x}_i$. The basic idea is to tailor aleatory uncertainty to each input, making the uncertainty input-dependent. This heteroscedastic case is also briefly discussed in Sec. 3.1 where heteroscedastic GPR is the focus of the discussion. The observation model now becomes the following:

$$y_i = f(\mathbf{x}_i) + \varepsilon(\mathbf{x}_i), \tag{30}$$

where the variance of the noise term $\varepsilon(\mathbf{x}_i)$, $\sigma_\varepsilon^2(\mathbf{x}_i)$, is now a function of $\mathbf{x}_i$. It turns out that a neural network can be trained to learn the mapping from $\mathbf{x}$ to σ_ε^2 [22, 23]. It then follows that we can train a neural network with parameters $\boldsymbol{\theta}$ that learns to predict both the mean $\mu(\mathbf{x}_i)$ and variance $\sigma^2(\mathbf{x}_i)$ of the target for each input $\mathbf{x}_i$. This neural network has two outputs, predicted mean $\widehat{\mu}(\mathbf{x}_i; \boldsymbol{\theta})$ and variance $\widehat{\sigma}^2(\mathbf{x}_i; \boldsymbol{\theta})$, which fully characterise a Gaussian predictive distribution, i.e., $\widehat{y}_i \sim \mathcal{N}\left(\widehat{\mu}(\mathbf{x}_i; \boldsymbol{\theta}), \widehat{\sigma}^2(\mathbf{x}_i; \boldsymbol{\theta})\right)$

Before optimizing the network parameters $\boldsymbol{\theta}$, we need to define a proper scoring rule that measures the quality of predictive (aleatory) uncertainty. For regression problems, a typical choice of a proper scoring rule is the likelihood function $p(y_i | \mathbf{x}_i; \boldsymbol{\theta})$ whose logarithmic transformation takes the following form [22, 173]:

$$\log p(y_i | \mathbf{x}_i; \boldsymbol{\theta}) = -\frac{\log \widehat{\sigma}^2(\mathbf{x}_i; \boldsymbol{\theta})}{2} - \frac{(y_i - \widehat{\mu}(\mathbf{x}_i; \boldsymbol{\theta}))^2}{2\widehat{\sigma}^2(\mathbf{x}_i; \boldsymbol{\theta})} - \text{constant}. \tag{31}$$

Given a training dataset consisting of N input-output pairs, $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$, $\boldsymbol{\theta}$ can be optimized by minimizing the following negative log-likelihood (NLL) loss on the entire training data, which is equivalent to maximizing the negative counterpart of the likelihood function in Eq. (31), after being summed up over all N training samples.

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left[\frac{\log \widehat{\sigma}^2(\mathbf{x}_i; \boldsymbol{\theta})}{2} + \frac{(y_i - \widehat{\mu}(\mathbf{x}_i; \boldsymbol{\theta}))^2}{2\widehat{\sigma}^2(\mathbf{x}_i; \boldsymbol{\theta})} \right], \tag{32}$$

where the constant term in Eq. (31) is omitted for brevity because it has nothing to do with the optimization of $\boldsymbol{\theta}$.

3.3.2. Epistemic uncertainty: using an ensemble of independently trained networks

As discussed in Sec. 3.3.1, the neural network ensemble approach captures aleatory uncertainty by training a neural network that produces a Gaussian output (or another type of probability distribution) for each input. This modeling process improves over traditional deterministic approaches that only produce a point estimate. Plus, the network-predicted variance varies according to the input, making it possible to capture input-dependent observational noise. One limitation is that minimizing the loss function in Eq. (32) yields a single vector of network parameters. Therefore, the resulting neural network cannot capture the uncertainty related to the network parameters because all parameters are deterministic. This treatment becomes an issue when only limited training data are available. These cases are more realistic than having abundant training data, and when training data are of limited quantities, epistemic uncertainty is high and cannot be ignored. One widely used way to capture epistemic uncertainty is to assume and estimate uncertainty in the parameters of a neural network model, also known as model parameter uncertainty or network parameter uncertainty.

After tuning the neural network parameters $\boldsymbol{\theta}$, at the time of prediction, each individual neural network generates a pair of outputs $(\widehat{\mu}(\mathbf{x}_*), \widehat{\sigma}(\mathbf{x}_*))$ with respect to an unseen instance $\mathbf{x}_*$, where $\widehat{\sigma}(\mathbf{x}_*)$ explicitly quantifies the aleatory uncertainty in model prediction arising from the random noise $\varepsilon(\cdot)$ associated with the target value. Next, to quantify the epistemic uncertainty associated with the neural network parameters $\boldsymbol{\theta}$, we can build an ensemble of neural networks, for example, by adopting the randomization strategy (random parameter initialization and mini-batch sampling) that attains a diverse set of neural networks. Suppose the neural network ensemble is composed of M individual neural networks, then the ensemble model produces M pairs of $(\widehat{\mu}_m(\mathbf{x}_*), \widehat{\sigma}_m(\mathbf{x}_*))$ $(m = 1, 2, \cdots, M)$ for the given input $\mathbf{x}_*$. The M pairs of predictions $(\widehat{\mu}_m(\mathbf{x}_*), \widehat{\sigma}_m(\mathbf{x}_*))$ can be viewed as a mixture of Gaussian distributions. Thus, we can use a single Gaussian distribution to approximate the mixture of Gaussian distributions as long as the mean and variance of the single Gaussian distribution are the same as the mean and variance of the mixture. Assuming that each individual neural network in the ensemble carries an equal weight, we have the mean and variance of the ensemble-predicted single Gaussian distribution as:

$$
\begin{aligned}
\overline{\mu}(\mathbf{x}_*) &= \frac{1}{M} \sum_{m=1}^{M} \widehat{\mu}_m(\mathbf{x}_*), \\
\overline{\sigma}^2(\mathbf{x}_*) &= \frac{1}{M} \sum_{m=1}^{M} \left(\widehat{\sigma}_m^2(\mathbf{x}_*) + \widehat{\mu}_m^2(\mathbf{x}_*) - \overline{\mu}^2(\mathbf{x}_*) \right).
\end{aligned} \tag{33}
$$

In the ensemble of neural networks, both the aleatory and epistemic uncertainty can be measured in a straightforward way. Specifically, the aleatory uncertainty arising from the noise associated with the observation y is reflected in the variance $\widehat{\sigma}_m(\mathbf{x}_*)$ predicted by each individual neural network. In contrast, the epistemic uncertainty associated with the network structure and parameters is manifested mainly as the difference with respect to $\widehat{\mu}_m(\mathbf{x}_*)$ of the M neural networks because each individual neural network is initialized with a random set of weights and biases and trained with a random mini-batch data for the gradient descent algorithm. Such randomness introduces

a sufficient amount of diversity among the individual models. Thus, the difference between the individual mean predictions $\widehat{\mu}_m(\mathbf{x}_*)$ that dominates the epistemic uncertainty characterizes the structural and parametric uncertainty pertaining to the neural network.

An interesting question about neural network ensembles is why training multiple neural networks of an identical architecture independently with just random initializations can capture epistemic uncertainty. The answer lies in that training a neural network with a large number of parameters (e.g., weights and biases) is an extremely intricate large-scale optimization problem in a high-dimensional space, and stochastic gradient descent-based algorithms oftentimes converge to different sets of parameter values $\boldsymbol{\theta}$ that are locally optimal [174]. As mentioned earlier, network training involves two sources of randomness: (1) random parameters initialization at the beginning of model training and (2) random perturbations of the training data to produce mini-batches of data in stochastic gradient descent As a result, the locally optimal parameters $\boldsymbol{\theta}$ vary from one trained neural network to another. Suppose M independent training runs give rise to M different local minima for the network parameters, which then lead to the creation of M individual members of an ensemble, as shown in Eq. (33). From the optimization perspective, the randomness in the initialization of neural network parameters and the sampling of mini-batch data encourages the optimization algorithm to explore different modes of the function space of a neural network. As a result, the predicted means of these M networks may differ substantially in some regions of the input space, while the predicted variances may still be similar, resulting in high epistemic uncertainty. These regions are typically located outside the training data distribution. Test samples falling into these regions are called OOD samples (as previously defined in Sec. 1), where ensemble predictions must be taken cautiously and are often untrustworthy.

3.4. Deterministic methods

A recent line of effort attempted to estimate the predictive uncertainty of neural networks using deterministic UQ methods. These methods require only a *single forward pass* on a neural network with deterministic parameters (weights and biases) to produce probabilistic outputs (e.g., predicted mean and variance for regression). A resulting benefit that makes these methods uniquely attractive is high computational efficiency (test time), particularly suitable for safety-critical applications with stringent real-time inference requirements (e.g., high-rate structural health monitoring and prognostics [175] and autonomous driving [176]). Examples of these deterministic methods include deterministic uncertainty quantification (DUQ) [177], deep deterministic uncertainty (DDU) [178], deterministic uncertainty estimation (DUE) [179], and spectral-normalized neural Gaussian process (SNGP) [180, 181]. This section first discusses distance awareness in the hidden space, which is a fundamental property of many deterministic methods, then provides a brief overview of how distance-aware feature representation (hidden layers) and uncertainty prediction (output layer) are achieved in SNGP.

3.4.1. Feature collapse and hidden-space regularization

The idea fundamental to many recently developed deterministic methods is (input) distance-aware representations in the latent (or hidden) space, achieved by regularizing the learned latent

representations of a neural network such that distances between points in the input space are preserved in the hidden space. The need for distance-aware latent representations comes from a recently reported phenomenon called *feature collapse* [177], where some OOD points in the input space are mapped through feature extraction to in-distribution points in the hidden space, leading to overconfident predictions at these OOD points. Feature collapse must be combatted for feature representations in the hidden space to be useful for epistemic uncertainty estimation and OOD detection. One option is imposing a bi-Lipschitz constraint on the feature extractor (i.e., a neural network excluding its output layer). The term "bi-Lipschitz" means a two-sided constraint on the Lipschitz constant of a feature extractor that determines how much distances in the input space contract (small Lipschitz, feature collapse) and expand (large Lipschitz, small changes in input resulting in drastic changes in latent features).

We now briefly describe the math pertaining to a bi-Lipschitz constraint. Suppose we take any two input points $\mathbf{x}$ and $\mathbf{x}'$ from a training dataset and let $h_{\mathrm{nn}}(\cdot)$ denote a function mapping an input into latent features (i.e., right after the activation function in the last hidden layer of a neural network). A bi-Lipschitz constraint on the mapping function h for any training input pairs looks like:

$$Lip^{\mathrm{lb}}||\mathbf{x} - \mathbf{x}'||_{\mathrm{input}} \leq ||h_{\mathrm{nn}}(\mathbf{x}) - h_{\mathrm{nn}}(\mathbf{x}')||_{\mathrm{hidden}} \leq Lip^{\mathrm{ub}}||\mathbf{x} - \mathbf{x}'||_{\mathrm{input}}. \tag{34}$$

where Lip^{lb} and Lip^{ub} are, respectively, the lower and upper bounds imposed on the Lipschitz constant of the feature extractor $h_{\mathrm{nn}}(\cdot)$, and $||\cdot||_{\mathrm{input}}$ and $||\cdot||_{\mathrm{hidden}}$ are, respectively, the distance metrics chosen for the input and hidden spaces. Setting the lower bound Lip^{lb} ensures that latent representations are distance sensitive, i.e., if $\mathbf{x}$ and $\mathbf{x}'$ are relatively far apart in the input space, they also have a relatively large distance in the hidden space. This sensitivity regularization allows the feature extractors to preserve input distances and directly counteracts the feature collapse issue by preventing OOD points from overlapping with in-distribution feature representations. Setting the upper bound Lip^{ub} ensures that hidden representations are smooth, i.e., small distance changes in the input space do not result in drastically large distance changes in the hidden space. This smoothness enforcement leads to feature extractors that generalize well and are robust to adversarial attacks. As for the distance metric, the Euclidean distance $dist(\cdot, \cdot)$ is often a good choice for measuring distances between input points and even those between hidden representations, except for image-like data. The Euclidean distance has recently been adopted as the distance metric in several deterministic UQ methods [177, 180, 181].

The feature-space regularization via a bi-Lipschitz constraint shown in Eq. (34) can be implemented during model training by applying either of the following two methods: (1) gradient penalty, originally introduced for training generative adversarial networks (GANs) [182] and then adopted for deterministic uncertainty estimation [177], and (2) spectral normalization, originally proposed again for training GANs [183] and then adopted for deterministic uncertainty estimation [178–181]. In the rest of this subsection, we will briefly go over the application of spectral normalization in SNGP. We will also discuss the use of GPR as the output layer by SNGP to produce an uncertainty estimate based on distances in the "regularized" hidden space.

3.4.2. Spectral normalization for distance preservation in hidden space

The algorithm of SNGP enforces the lower bound of the Lipschitz constant in Eq. (34) simply by using network architectures with residual connections (e.g., residual networks) while imposing the upper bound using spectral normalization. Briefly, for each hidden layer, spectral normalization first calculates the spectral norm of the weight matrix $\mathbf{W}$ (i.e., the largest singular value of $\mathbf{W}$), denoted as $||\mathbf{W}||_2$, and then normalizes $\mathbf{W}$ using its spectral norm as:

$$\widehat{\mathbf{W}}_{\text{sn}} = \frac{\gamma \cdot \mathbf{W}}{||\mathbf{W}||_2}, \tag{35}$$

where γ is the upper bound of the spectral norm (i.e., $||\mathbf{W}||_2 \leq \gamma$), also called the *spectral norm upper bound*, which effectively enforces an upper bound on the Lipschitz constant of the mapping function in the hidden layer. The weight matrix needs to be spectral-normalized only when its spectral norm exceeds the upper bound, i.e., when $||\mathbf{W}||_2 > \gamma$ [180]. Introducing the spectral norm upper bound gives rise to the flexibility to balance the expressiveness and distance awareness of the resulting spectral-normalized feature extractor. Specifically, when γ takes a small value ($\gamma < 1$), the feature extractor tends to contract toward identity mapping, thereby limiting the ability of the feature extractor to learn complex nonlinear mapping, critically important for achieving high prediction accuracy on the training distribution; when γ is large ($\gamma \gg 1$), the feature extractor is allowed to expand and be more expressive but may not preserve input distances. However, in reality, this flexibility may become a limitation against adoption, as γ needs to be carefully tuned to balance accuracy/generalizability and distance awareness.

3.4.3. Gaussian process regression output layer for distance-aware prediction

As discussed in Sec. 3.4.2, the feature extraction layers of a neural network can be encouraged to preserve distances in the input space through a combination of residual connections and spectral normalization. Now we can make the predictive uncertainty of this neural network (input) distance-aware by replacing the last (output) layer with a GPR model that takes the learned hidden features as the input. Let us start by using the squared exponential kernel in Eq. (10) as the base kernel. We replace the input points $\mathbf{x}$ and $\mathbf{x}'$ with their "distance-aware" feature representations in the hidden space, $h_{\text{nn}}(\mathbf{x}; \boldsymbol{\theta})$ and $h_{\text{nn}}(\mathbf{x}'; \boldsymbol{\theta})$, where $h_{\text{nn}}(\,\cdot\,; \boldsymbol{\theta})$ denotes the feature extraction part of a neural network parameterized by $\boldsymbol{\theta}$, i.e., the neural network up to the last hidden layer. The resulting kernel takes the following form:

$$k_{\text{nn}}(\mathbf{x}, \mathbf{x}') = k(h_{\text{nn}}(\mathbf{x}; \boldsymbol{\theta}), h_{\text{nn}}(\mathbf{x}'; \boldsymbol{\theta})) = \sigma_{\text{f}}^2 \exp\left(-\frac{||h_{\text{nn}}(\mathbf{x}; \boldsymbol{\theta}) - h_{\text{nn}}(\mathbf{x}'; \boldsymbol{\theta})||^2}{2l^2}\right). \tag{36}$$

When the neural network is a DNN (e.g., with > 5 hidden layers), the above kernel can sometimes be called a *deep kernel*. The prior and posterior derivations follow the standard procedures described in Secs. 3.1.1.c and 3.1.1.d. Essentially, we perform a GPR in the learned, distance-preserving feature space instead of the input space. The resulting GPR model yields the posterior variance of a test

input $\mathbf{x}_*$ based on its Euclidean distances from all training points in the hidden space, leveraging the distance awareness property of GPR, extensively discussed in Secs. 3.1.1.b and 3.1.1.d, to make the output layer distance aware. Intuitively speaking, let us suppose $\mathbf{x}_*$ keeps moving away from the training distribution. The value of the hidden-space kernel between any training input $\mathbf{x}_i$ and $\mathbf{x}_*$, $k_{\text{nn}}(\mathbf{x}_i, \mathbf{x}_*)$, will become smaller and smaller given the distance preservation property of $h_{\text{nn}}(\cdot)$. At some point, this kernel value will quickly approach zero. As a result, the posterior variance at $\mathbf{x}_*$ will keep increasing and eventually approach its maximum value σ_{f}^2. This scenario suggests the distance awareness property of SNGP makes it an ideal tool for OOD detection.

To make inference computationally tractable, SNGP applies two approximations to the GPR output layer: (1) expanding the GPR model into simpler Bayesian linear models in the space of random Fourier features and (2) approximating the resulting posterior via Laplace approximation [180]. It is noted that another deterministic UQ method named DUE also uses spectral normalization plus residual connections to encourage a bi-Lipschitz mapping to the hidden space and GPR in the output layer. The only major difference is that DUE uses variational inducing point approximation for GPR in place of the random Fourier feature expansion [179].

3.4.4. Discussion on deterministic UQ methods

Deterministic methods run only a single forward pass for UQ and are computationally more attractive than BNN and neural network ensemble. These deterministic approaches also thrive at OOD detection thanks to their distance awareness property. However, they typically cannot separate aleatory and epistemic uncertainty. Additionally, they may require modifications to the network architecture (e.g., adding residual connections to enforce the Lipschitz lower bound in SNGP [178, 180]) and training procedure (e.g., to accommodate spectral normalization) with additional hyperparameters (e.g., the spectral norm upper bound γ, length scale l, and signal amplitude σ_{f}). Finally, it was reported that deterministic methods such as SNGP may produce substantially lower-accuracy UQ (e.g., higher values of the ECE defined in Sec. 4.1.3) than more mature methods such as MC dropout and neural network ensemble [184, 185]. Findings from these recent benchmarking studies call for more effort to investigate the calibration performance of deterministic approaches and, in particular, to evaluate how accurately the predictive uncertainty can be used as a proxy for model accuracy for in-distribution, around-distribution, and OOD data.

3.5. Toy example

Following the above discussions on several popular methods for UQ of ML models, we now consider a toy 2D regression problem to compare the performance of these UQ methods quantitatively. The functional relationship between y and $\mathbf{x}$ underlying this toy example takes the following form: $y(\mathbf{x}) = \frac{1}{20}((1.5 + x_1)^2 + 4) \times (1.5 + x_2) - \sin\frac{5 \times (1.5 + x_1)}{2}$. To train an ML model, we randomly generate 800 samples from the following two bivariate Gaussian distributions, with 400 samples randomly drawn from either distribution, and use these 800 samples as the training data.

$$\mathcal{N}\left(\begin{bmatrix} 8 \\ 3.5 \end{bmatrix}, \begin{bmatrix} 0.4 & -0.32 \\ -0.32 & 0.4 \end{bmatrix}\right), \mathcal{N}\left(\begin{bmatrix} -2.5 \\ -2.5 \end{bmatrix}, \begin{bmatrix} 0.4 & -0.32 \\ -0.32 & 0.4 \end{bmatrix}\right). \tag{37}$$
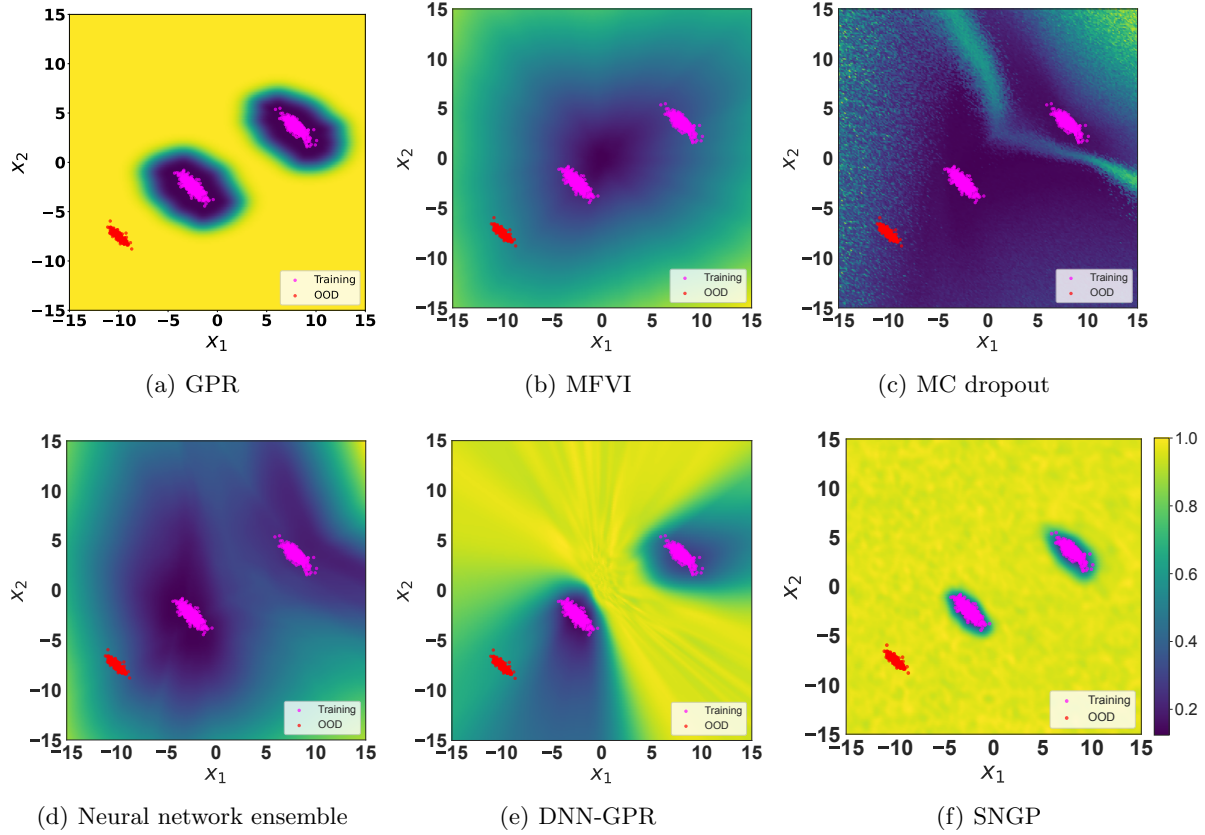
Figure 7: The uncertainty maps by five different methods for UQ of ML models on the toy 2D regression problem. These methods are Gaussian process regression – GPR (a), MFVI – mean-field variational inference (b), Monte Carlo dropout – MC dropout (c), neural network ensemble (d), deep neural network with Gaussian process regression – DNN-GPR (e), Spectral-normalized Neural Gaussian Process – SNGP (f). The two clusters colored in purple represent the training data, while the cluster colored in red indicates a cluster of OOD instances. The background in each 2D plot is color-coded according to the predictive uncertainty by the corresponding UQ method, with yellow (blue) indicating high (low) uncertainty.

These training samples form two separate clusters with no overlap in between, as shown in Fig. 7. As can be observed in both Eq. (37) and Fig. 7, the two clusters have an identical variance-covariance matrix and differ significantly only in the mean vector. We now apply the previously introduced UQ methods on the 800 training samples. For those methods requiring neural networks, the UQ methods are built on a backbone of similar residual neural network architectures with four 64-neuron residual layers. For example, in the case of neural network ensemble, a Gaussian layer is inserted at the end of a residual neural network; while in the case of MC dropout, dropout with a rate of 0.2 is applied at the end of each residual layer.

To test the UQ performance of different ML models, we generate a uniform meshgrid consisting of 40,000 (= 200×200) samples with x_1 and x_2 spanning in the range $[-15, 15]$. Next, an uncertainty heap map is constructed to visualize the predictive uncertainty of each trained ML model within the domain. Figure 7 shows the uncertainty heat maps obtained by the five different UQ methods on this toy problem. At a quick glance, both GPR and SNGP exhibit a desirable behavior in producing high quality predictive uncertainty: the predictive uncertainty is quite low for samples

in the proximity of the in-distribution/training data (dots in pink color). At the same time, both GPR and SNGP generate high predictive uncertainty when test sample point $[x_1, x_2]^\mathrm{T}$ moves far away from the training data clusters. As a result, both GPR and SNGP successfully assigned high uncertainty to the 200 OOD samples (dots in red color at the bottom left of Fig. 7) - which are randomly generated to test the OOD detection capability of different UQ techniques.

Unlike GPR and SNGP, the other four UQ methods have a relatively poor performance in quantifying predictive uncertainty. As can be observed in Fig. 7 (c-e), MC dropout, deep ensemble, and DNN-GPR assign low uncertainty for samples that are quite far away from the training data. As a consequence, these three UQ techniques are likely to fail to detect the 200 OOD samples whose predictions are associated with relatively low uncertainty, as shown in the bottom-left corners of Fig. 7 (c), (d), and (e). Besides the lack of ability in OOD detection, these three UQ techniques share another feature in common: their uncertainty output is more sensitive to the (hypothetical) boundary that separates the two clusters of training data, while they exhibit a substantially faulty behavior when establishing the decision boundary (trustworthy vs. untrustworthy region) around each cluster of training data itself. More specifically, for a given test sample, the predictive uncertainty generated by these three UQ techniques has a low sensitivity to how distant is a test sample's distribution with respect to the training data. Regarding the mean-field variational inference (MFVI), its predictive uncertainty gets increased in accordance with the distance away from the two training clusters, however, MFVI assigns nearly an identical uncertainty for the data between the two training clusters as they are near the data, which contradicts with our anticipation. This suggests that MFVI suffers from the lack of in-between uncertainty due to the approximation to Bayesian inference, and such finding is also confirmed by Foong et al. [165]. Consequently, the predictive uncertainty by these UQ techniques is unprincipled because their quantified uncertainty does not match our expectation that uncertainty should clearly distinguish in-domain and out-domain data.

The significant difference in the uncertainty heat map across different UQ methods is primarily attributed to their distance awareness capability. MC dropout, deep ensemble, and DNN-GPR do not have the ability to properly quantify the distance of an input sample away from the training data manifold. Instead, the predictive uncertainty at an input sample quantified by MC dropout, deep ensemble, and DNN-GPR seems to be established upon the distance of the input sample from a decision boundary separating the two clusters of training data. Therefore, it is not surprising to see all these three UQ methods assign low uncertainty to the 200 OOD samples even though they are quite far from the training data. Distinct from MC dropout and deep ensemble, GPR, DNN-GPR, and SNGP are equipped with a good sense of awareness with respect to the distance between an input sample and the training data manifold. As a result, they are comparatively more principled in the sense that the uncertainty is much higher for the input sample that lies far from the training data. Finally, even though both DNN-GPR and SNGP have GPR as the output layer, DNN-GPR is free from determining what information to discard in the hidden space, while SNGP imposes a spectral normalization on the latent representation of the input sample, thus making the output layer distance sensitive in the hidden space. In a broad context, the sound UQ by GPR and SNGP substantially facilitates the identification of OOD samples, establishing a trustworthy region

in the input space where ML predictions are reliable.

3.6. Summary

Table 2: A qualitative comparison of state-of-the-art UQ approaches covered in this tutorial

Quantity of interest	Gaussian process regression	Bayesian neural network			Neural network ensemble	Deterministic method	
		MCMC	Variational inference	MC dropout		DNN-GPR	SNGP
Quality of UQ (e.g., measured by calibration curve)	High	High-medium[a]	Medium	Medium-low	High	Medium	High
Computational cost (training)	High[b]	High	High-medium	Low	Low	High	High
Computational efficiency (test)	High[b]	Low	High-medium	Medium	Medium-low	Low	Low
Ability to detect OOD samples	Strong	Weak	Weak	Weak	Moderate	Strong-moderate	Strong
Scalability to high dimensions	Low	Low	Medium	High	High	High	High
Effort to convert a deterministic to a probabilistic model	Not applicable	High	High-medium	Low	Medium	High-medium	High-medium
Ability to distinguish aleatory and epistemic uncertainty	Yes	Yes	Yes	No	Yes	No	No
Basis of UQ	Analytical	Sampling	Sampling	Sampling	Hybrid	Analytical	Analytical
Stability of quantified uncertainty to parameter initialization	High	High	High	Low	Medium	High	High

[a] Accuracy is largely affected by the quality of the assumed prior.
[b] Efficient only for problems of low dimensions (typically < 10) and small training data (typically < 5000 points).

The numerical example in Sec. 3.5 demonstrates the performance difference among different UQ methods with an emphasis on OOD detection. Comprehensive comparison of these UQ methods may help better guide users to select appropriate UQ methods for specific ML applications. To this end, we construct a table (Table 2) to qualitatively compare these methods along multiple dimensions, such as the quality of UQ, computational costs in training and test, etc. In the first place, regarding the calibration accuracy of these UQ methods, GPR and SNGP generally outperform other alternate UQ methods, which is also confirmed in the previous numerical example. For the computational cost associated with training an ML model, implementing a Bayesian neural network via MCMC or variational inference incurs a relatively higher computational cost than MC dropout, as MC dropout consumes nearly the same amount of computational time as training a regular neural network. In

terms of scalability, it is well-known that GPR suffers from the curse of high dimensionality, so training and testing GPR models may be computationally very expensive for high-dimensional problems. The other three UQ methods (neural network ensemble, DNN-GPR, and SNGP) are computationally cheaper than GPR, MCMC, and variational inference. We have similar findings regarding the computational burden of these UQ methods at test time.

An important function of UQ built atop the original deterministic ML model is to serve as a safeguard to detect OOD samples for the purpose of increasing the reliability of ML models. In this regard, SNGP achieves similar performance as the gold standard GPR, while the remaining UQ methods may perform poorly in detecting OOD samples. Besides strong OOD detection capability, SNGP also exhibits a desirable feature in scalability, while such a feature is missing in GP. However, compared to GP, SNGP requires an additional effort to turn a deterministic ML model into a probabilistic counterpart for UQ, while GPR is born with the capability of UQ. As for the uncertainty decomposition, GPR, Bayesian neural network, and neural network ensemble all have some capability to quantify aleatory and epistemic uncertainty separately, while such a capability may be lacking in the MC dropout version of Bayesian neural network as well as in DNN-GPR and SNGP. Next, both GPR and SNGP estimate the predictive uncertainty of ML models in an analytical form. In contrast, the other UQ methods draw Monte Carlo samples to approximate the uncertainty, which is a major performance barrier if critical applications require real-time inferences.

4. Evaluation of predictive uncertainty

Let us now shift our focus to the performance evaluation of probabilistic ML models. A unique property of these models is that they do not simply produce a point estimate of y and instead output a probability distribution of y, $p(y)$, that fully characterizes the predictive uncertainty. This unique property requires that the performance evaluation examines both the prediction accuracy, e.g., the RMSE or mean absolute error calculated based on the mean predictions for regression, and the quality of predictive uncertainty, e.g., how accurately the predictive uncertainty reflects the deviation of a model prediction from the actual observation. In what follows, we will discuss ways to assess the quality of predictive uncertainty.

4.1. Calibration curves and metrics

A standard approach to assessing the quality of predictive uncertainty is creating a calibration curve, also called a reliability diagram [186–188]. We will first give a detailed walkthrough of creating calibration curves for regression and classification and then present UQ performance metrics that can be derived from a calibration curve.

4.1.1. Calibration curves for regression

Let us assume, in a regression setting, that we have a validation/test set of N input-output pairs, $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$. Given a trained probabilistic ML model (e.g., an ensemble of probabilistic neural networks or simply called a neural network ensemble as discussed in Sec. 3.3) parameterized by $\boldsymbol{\theta}$, let $\widehat{y}_i = f(\mathbf{x}_i; \boldsymbol{\theta})$ denote the predicted outcome for the i-th

validation/test sample $\mathbf{x}_i$, $i = 1, \cdots, N$. Without loss of generality, let us further assume that the probabilistic output $\widehat{y}_i$ follows a Gaussian distribution, characterized by a Gaussian probability density function, $p\left(\widehat{y}_i; \mu_{\boldsymbol{\theta}}\left(\mathbf{x}_i\right), \sigma_{\boldsymbol{\theta}}\left(\mathbf{x}_i\right)\right) = \frac{1}{\sigma_{\boldsymbol{\theta}}(\mathbf{x}_i)} \phi\left(\frac{\widehat{y}_i - \mu_{\boldsymbol{\theta}}(\mathbf{x}_i)}{\sigma_{\boldsymbol{\theta}}(\mathbf{x}_i)}\right)$, with the predicted mean $\mu_{\boldsymbol{\theta}}\left(\mathbf{x}_i\right)$ and standard deviation $\sigma_{\boldsymbol{\theta}}\left(\mathbf{x}_i\right)$. For a given confidence level $c \in [0, 1]$, we can easily derive a two-sided $100c\%$ confidence interval for the Gaussian random variable $\widehat{y}_i$ as:

$$CI_i^c = \left[\mu_{\boldsymbol{\theta}}\left(x_i\right) - z_{\frac{1+c}{2}}\sigma_{\boldsymbol{\theta}}\left(\mathbf{x}_i\right), \mu_{\boldsymbol{\theta}}\left(x_i\right) + z_{\frac{1+c}{2}}\sigma_{\boldsymbol{\theta}}\left(\mathbf{x}_i\right)\right], \tag{38}$$

where $z_{\frac{1+c}{2}}$ denotes the $\left(\frac{1+c}{2}\right)^{\text{th}}$ quantile of the standard normal distribution, i.e., $z_{\frac{1+c}{2}} = \Phi^{-1}\left(\frac{1+c}{2}\right)$, with $\Phi(\cdot)$ denoting the cumulative distribution function (CDF) of the standard normal distribution. The probability of a random realization of $\widehat{y}_i$ falling into CI_i^c equals c, expressed as

$$\int_{\mu_{\boldsymbol{\theta}}(x_i) - z_{\frac{1+c}{2}}\sigma_{\boldsymbol{\theta}}(\mathbf{x}_i)}^{\mu_{\boldsymbol{\theta}}(x_i) + z_{\frac{1+c}{2}}\sigma_{\boldsymbol{\theta}}(\mathbf{x}_i)} p\left(\widehat{y}_i; \mu_{\boldsymbol{\theta}}\left(\mathbf{x}_i\right), \sigma_{\boldsymbol{\theta}}\left(\mathbf{x}_i\right)\right) d\widehat{y}_i \underset{\tau \equiv \left(\frac{\widehat{y}_i - \mu_{\boldsymbol{\theta}}(\mathbf{x}_i)}{\sigma_{\boldsymbol{\theta}}(\mathbf{x}_i)}\right)}{=} \int_{-z_{\frac{1+c}{2}}}^{z_{\frac{1+c}{2}}} \phi\left(\tau\right) d\tau = c. \tag{39}$$

If we choose to use a CDF P_i to characterize the probability distribution of $\widehat{y}_i$ that may not follow a Gaussian distribution, we can write out the $100c\%$ confidence interval for any arbitrary distribution type,

$$CI_i^c = \left[P_i^{-1}\left(\frac{1 - c}{2}\right), P_i^{-1}\left(\frac{1 + c}{2}\right)\right], \tag{40}$$

where $P_i^{-1}\left(c\right) = \inf\left(\widehat{y}_i : P_i^{-1}\left(\widehat{y}_i\right) \geq c\right)$. Here, P_i^{-1} is an inverse of the CDF P_i, also called a quantile function, and becomes Φ^{-1} for the standard normal distribution. Alternatively, we can derive a one-sided confidence interval $CI_i^c = \left[-\infty, P_i^{-1}\left(c\right)\right]$.

Ideally, the UQ of this ML model should yield a $100c\%$ confidence interval that contains the observed y for approximately $100c\%$ of the time. For example, if $c = 0.95$, then y_i should fall into a 95% confidence interval $CI_i^{0.95}$, one- or two-sided, for nearly 95% of the time. In other words, we expect that approximately 95% of the N validation/test samples have their observed y values fall into the respective 95% confidence intervals. The fraction of validation/test samples for which the confidence intervals contain the observations can be called *observed confidence* ($\widehat{c}$) or sometimes *accuracy*, expressed as $\widehat{c} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(y_i \in CI_i^c\right)$, where $\mathbb{I}\left(prop\right)$ is an indicator function that takes the value of 1 if the proposition *prop* is true and 0 otherwise. If we plot observed confidence against *expected confidence* (c) over $[0, 1]$, we will create a calibration curve, sometimes called a reliability diagram (see an example in the right-most plot of Fig. 9). This calibration curve shows how well predictive uncertainty is quantified, and a perfect UQ should yield a calibration curve that overlaps with the diagonal line ($y = x$). If the observed confidence is higher than expected at some c values, the model is said to be underconfident at these confidence levels; otherwise, the model is deemed overconfident. In predictive maintenance practices, reliability/maintenance engineers often prefer underconfident predictions over overconfident predictions, as overconfident predictions are more likely to trigger maintenance actions that are either unnecessarily early or too late. If 90% or 95% is chosen as the confidence level, it is preferred that the observed confidence (or accuracy) is very

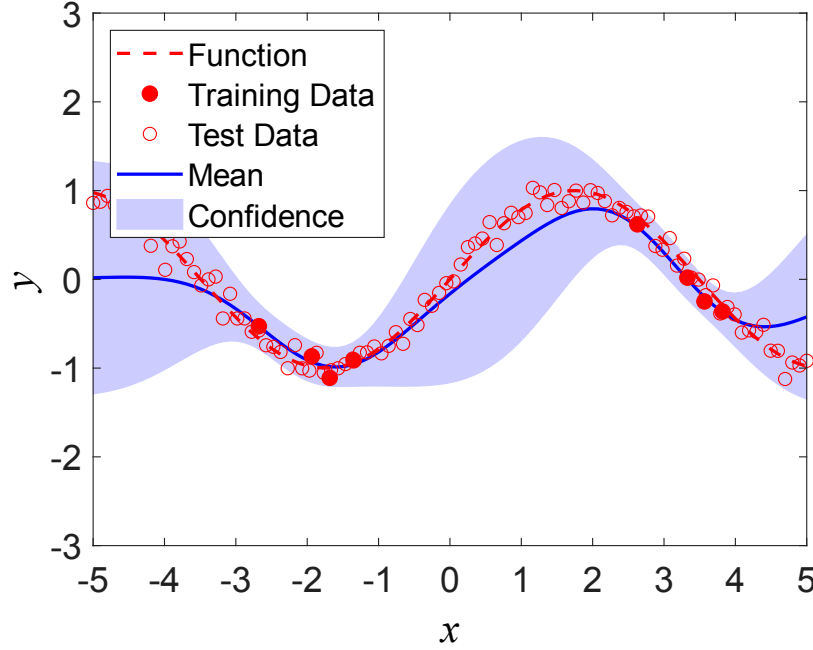close to or slightly higher than 90% or 95%.

Figure 8: An example dataset with eight training samples (solid red circles) and 100 test samples (hollow red circles), plotted with the underlying one-dimensional function and fitted GPR model. Shown for the fitted GPR model is the poterior mean function (solid blue curve) and a collection of 95% confidence intervals (light blue shade) for the noisy observations ($\mathbf{y}_*$) at new/test points. These test points are equally spaced between -5 and 5 along the x-axis.

Let us now do a step-by-step walkthrough of how a calibration curve is created using a toy example. This example uses training and test data generated from the same 1D function and Gaussian observation model used to generate Figs. 5 and A.25 in Sec. 3.1.1. The observation model consists of a sine function corrupted with a white Gaussian noise term, $y = \sin(0.9x) + \varepsilon$ with $\varepsilon \sim \mathcal{N}\left(0, 0.1^2\right)$. As shown in Fig. 8, we fit a GPR model to the eight training data points and test this model on 100 test points. It can be seen from the figure that the regressor reports high uncertainty at test points that fall outside of the x ranges where training samples exist. If we compare the in-distribution test samples (i.e., whose x values fall into $[-3, -1)$ or $[2, 4)$) with the OOD samples (whose x values lie within $[-5, -3)$, $[-1, 2)$, or $[4, 5)$), we observe higher predictive uncertainty on the OOD samples, where the model's predictions are more likely to be incorrect. Creating a calibration curve for this toy example consists of three steps.

Step 1: We start by choosing K confidence levels between 0 and 1, $0 \le c_1 < c_2 < \cdots < c_K \le 1$. In this example, we choose 11 ($K = 11$) confidence levels equally spaced between 0 and 1, i.e., 0, 0.1, $\cdots$, 0.9, 1 (see Step 1 in Fig. 9).

Step 2: We then compute for each expected confidence level c_j the observed confidence as:

$$\widehat{c}_j = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(y_i \in CI_i^c\right). \tag{41}$$

As mentioned above, $CI_i^c = \left[P_i^{-1}\left(\frac{1-c}{2}\right), P_i^{-1}\left(\frac{1+c}{2}\right)\right]$ for a two-sided confidence interval and $CI_i^c = \left[-\infty, P_i^{-1}\left(c\right)\right]$ for a one-sided confidence interval. Step 2 in Fig. 9 shows an example of how to implement Eq. (40) for $c_6 = 0.5$.

Step 3: We finally plot the K pairs of expected vs. observed confidence, $\{(c_1, \widehat{c}_1), \cdots, (c_K, \widehat{c}_K)\}$, which gives rise to a calibration curve. In the toy example, we have 11 pairs of $(c_j, \widehat{c}_j)$ plotted to form a discrete calibration curve in Step 3 in Fig. 9.
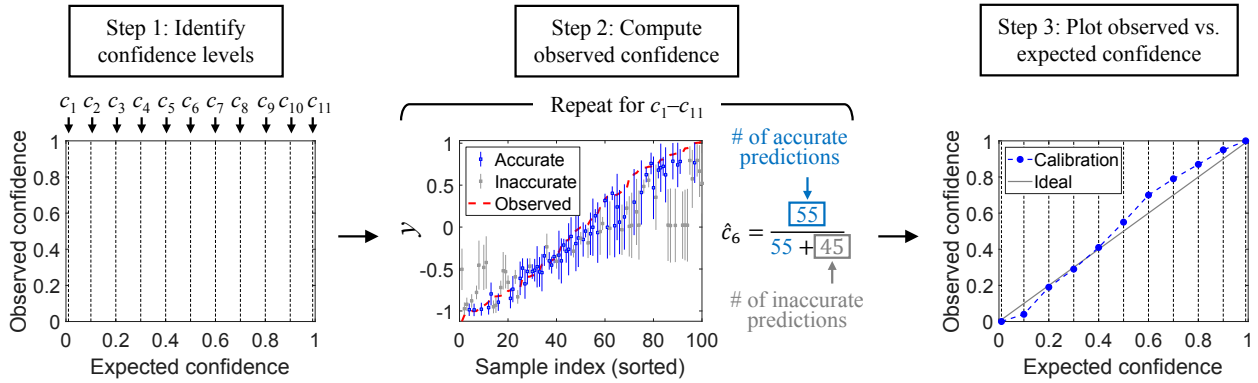
Figure 9: Illustration of three-step procedure to create a calibration curve for toy regression problem shown in Fig. 8.

Suppose we are interested in assessing the regression model's UQ quality at the confidence level of 90%. In that case, we can observe from the calibration curve drawn in Step 3 that the Gaussian process is underconfident because the confidence we expect the regressor to have ($c_{10} = 90\%$) is lower than the observed (empirically estimated) confidence ($\widehat{c}_{10} = 95\%$) or simply $c_{10} < \widehat{c}_{10}$. More specifically, the actual proportion of times that the model's 90% confidence interval contains the ground truth (i.e., the model is correct) is higher than the expected value (i.e., 90%). Being underconfident also means that the model tends to produce higher-than-true uncertainty in its predictions, which is often more desirable in safety-critical applications than having an overconfident model.

To understand how a calibration curve behaves as a test window varies, we expand the data from $[-5, 5]$, as shown in Fig. 8, to $[-15, 15]$, as shown in Fig. 10, while keeping the same number of test samples (i.e., 100). As shown in Fig. 10, the new test dataset includes much more OOD samples that fall outside the range of $[-5, 5]$. The calibration curve on this new dataset is plotted alongside the one on the original dataset in Fig. 11. Let us compare the new (red) and original (blue) calibration curves. We can observe that having more OOD test samples degrades the quality of UQ by moving the calibration curve further away from the ideal line. This observation is not surprising because high quality UQ (i.e., producing predictive uncertainty that accurately
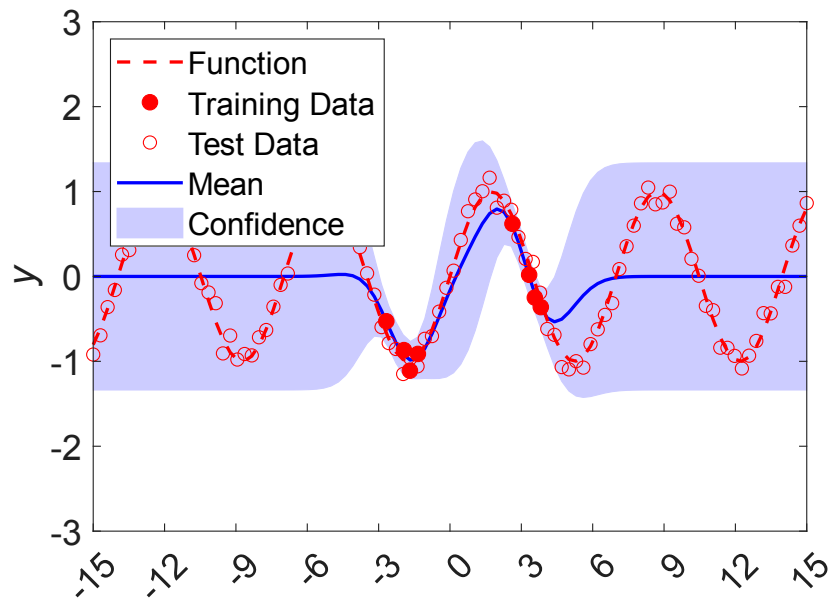
Figure 10:
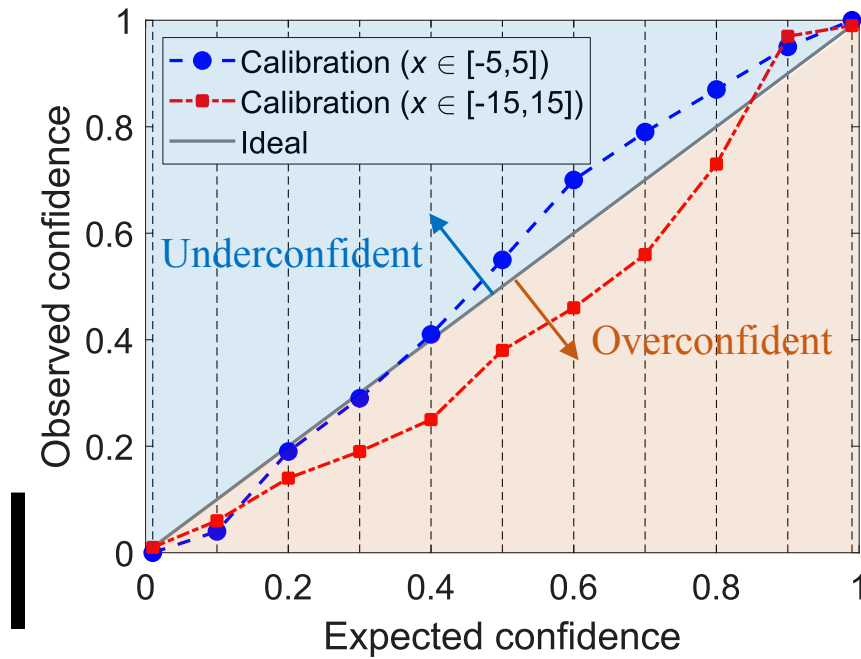
Figure 11: Comparison of calibration curves for two different ranges of test data for the toy 1D mathematical problem. Test samples are equally spaced between -5 and 5 (the same as Figs. 8 and 9) and between -15 and 15, respectively, for the two test ranges.

reflects prediction errors) is expected to be more challenging on OOD samples than in-distribution samples. Another interesting observation is that the GPR model appears more overconfident in

figures/comparison_calibration_curves.pdf

making predictions on the new test dataset with more OOD samples. Our explanation for this observation is that as a test sample x_i moves farther away from the training data, the prediction error may increase drastically (i.e., the model-predicted mean may deviate substantially more from the true observation), but the predictive uncertainty by a UQ method may start to saturate at a certain distance away from the training distribution (see, for example, the flat confidence bounds in Fig. 10 when $x_i \in [-15, -6] \cup [7, 15]$), making it more difficult for a probabilistic prediction to be accurate (i.e., the predictive confidence interval at x_i contains the ground truth y_i). Essentially, in some cases, the predictive uncertainty cannot catch up with the prediction error as a test sample moves further away from a training distribution. In that case, it is critically important to establish boundaries in the input space within which predictive uncertainty cannot be trusted. Very little effort has been devoted to trustworthy UQ, and more effort is urgently needed on this front.

4.1.2. Calibration curves for classification

Creating calibration curves for classification models involves a multi-step procedure that differs from that for regression models. Let us use a binary classifier as an example. Similar to the regression setting, we also have access to a validation/test set of N input-output pairs, $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$. In a binary classification setting, the output takes the value of either 0 or 1, i.e., $y \in \{0, 1\}$. Creating a calibration curve for this classification setting involves three steps.

Step 1: The first step is to discretize the observed confidence c into some number (K) of bins of width $1/K$. For example, if $K = 10$, we then have ten intervals of observed confidence, $[0, 0.1], (0.1, 0.2], \cdots, (0.9, 1.0]$.

Step 2: We then compute for each bin $B_j = \left(c_j - \frac{1}{2K}, c_j + \frac{1}{2K}\right]$ the observed confidence as

$$\widehat{c}_j = \frac{\sum\limits_{i=1}^{N} y_i \mathbb{I}\left(f_{\boldsymbol{\theta}}(\mathbf{x}_i) \in B_j\right)}{\sum\limits_{i=1}^{N} \mathbb{I}\left(f_{\boldsymbol{\theta}}(\mathbf{x}_i) \in B_j\right)}, \tag{42}$$

where $f_{\boldsymbol{\theta}}(\mathbf{x}_i)$ outputs the probability of $y_i = 1$.

Step 3: The final step is to plot the predicted vs. the observed confidence for class 1 for each bin B_j.

4.1.3. Calibration metrics

Several calibration metrics can be defined based on a calibration curve (see an example in Fig. 9). For example, a simple metric can be the area between the calibration curve and the identity line, sometimes called the miscalibration area, which interestingly shares a similar concept with the area metric or u-pooling metric commonly used in the validation of computer simulation models [189]. Another calibration metric that is more widely used is the so-called expected calibration error (ECE), originally proposed for classification [190] and later extended for regression [191].

Note though that the extension in [191] focused on deriving calibration curves and did not propose an ECE definition under regression settings. The ECE can be defined as the weighted average difference between a calibration curve and the ideal linear line, $ECE = \sum_{j=1}^{K} w_j |\widehat{c}_j - c_j|$, where the weight w_j can be set as either a constant (i.e., $1/K$) or proportional to the number of samples falling into each bin, i.e., $w_j \propto \sum_{i=1}^{N} \mathbb{I}\left(y_i \in CI_i^C\right)$ for regression and $w_j \propto \sum_{i=1}^{N} \mathbb{I}\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}_i\right) \in B_j\right)$ for binary classification [191]. Figure 12 illustrates the calibration-ideal differences as error bars on the calibration curve obtained for the toy 1D mathematical problem shown in Fig. 8. Assuming equal weights $(w_1 = w_2 =, \cdots,$

which means the observe
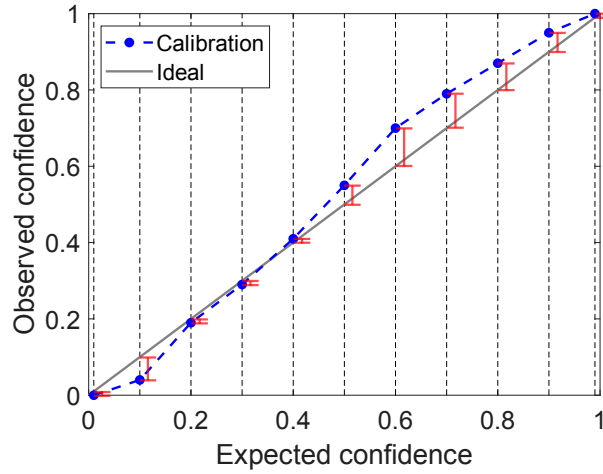
Figure 12: Calibration curve for the toy 1D mathematical problem shown in Fig. 8. This figure builds on the calibration curve shown in Step 3 of Fig. 9 and also includes the differences between calibrated and ideal (red error bars) used to calculate the ECE for this example.

figures/calibration_with_error_bars.pdf

4.1.4. Recalibration

If the calibration curve deviates significantly from the identity function (perfect calibration), a recalibration may be needed to bring the calibration curve closer to the linear line. For example, this recalibration can be [Final] ametric approach called Platt scaling, which modifies the non-probabilistic predict [ ] binary classifier (e.g., a neural network or support vector classifier) using a two-parameter, simple linear regression model and optimizes the two model parameters by minimizing the NLL on a validation dataset [188, 192]. It is straightforward to extend Platt scaling to multi-class settings, for example, by expanding the simple linear regression model to a multivariate linear regression model [193]. Another simple extension is temperature scaling, a single-parameter version of Platt scaling [193], which was shown to be effective in re-calibrating deterministic neural networks capable of UQ [178]. Another approach to recalibrating classification models is training an auxiliary regression model on top of the trained machine learning predictor, again using a validation dataset [191]. A popular choice of the auxiliary regression model is an iso-

tonic regression model, where a non-parametric isotonic (monotonically increasing) function maps probabilistic predictions to empirically observed values on a validation set. Recalibration using isotonic regression was originally proposed for classification [187, 188] and then extended to regression [191]. It found recent applications in the PHM field, such as battery state-of-health estimation [194].

Both Platt scaling and isotonic regression require a separate validation dataset of a decent size (typically 20-50% of the training dataset) to either optimize scaling parameters (Platt scaling) or build a non-parametric regression model (isotonic regression), while in reality, such a decent sized validation dataset may not be available. A comparative study of re-calibration approaches was performed in [193], where temperature scaling was found to be the most simple and effective.

4.1.5. Connecting UQ calibration with model validation

It is worth noting the connection between UQ calibration and the u-pooling method. U-pooling is a method for validating computer simulation models and has been well-established in the model validation community [189, 195]. The u-pooling method aims to test whether all experimental observations, often made under multiple experimental conditions and sparse under each condition, come from the probability distributions predicted by a computer simulation model for the respective experimental conditions. If each observation comes from the corresponding predictive distribution, the CDF values of the experimental observations, "pooled" together from all physical experiments, should follow a standard uniform distribution. Briefly, the u-pooling method first calculates the CDF value or *u value* of each observation, u_i, based on the predictive CDF by a computer simulation model, then plots the empirical CDF of u, where u is along the x-axis and CDF is along the y-axis, and finally computes the area difference between the empirical CDF of u and the CDF of the standard uniform distribution (diagonal line). The smaller the area difference, the more accurate (in a probabilistic sense) the computer simulation model.

Plotting a UQ calibration curve like the ones in Fig. 11 but for one-sided confidence intervals could also start by calculating the predictive CDF values (u values in the u-pooling method) of all test samples, $u_1, \cdots, u_N$. Then, the observed confidence $\widehat{c}$ (y-axis) for any expected confidence c (x-axis) can be calculated as the fraction of the CDF values that are smaller or equal to c, i.e., $\widehat{c} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(u_i \leq c)$. The differences are that (1) the empirical CDF plot in the u-pooling method shows N eventually spaced empirical CDF values on the y-axis, while the number of expected confidence levels on the x-axis of a UQ calibration plot is manually selected; and (2) for each empirical CDF (y-axis for u-pooling) or expected confidence (x-axis for UQ calibration) value c, the u-pooling method plots the corresponding percentile of u, i.e., the $100c^{\text{th}}$ percentile of u based on the dataset of N u values, while UQ calibration plots the corresponding fraction of the u values that are no greater than c. Additionally, the u-pooling method strictly starts by looking at u values. It then derives their empirical CDF values. In contrast, UQ calibration, to some degree, has a reverse process where it begins with manually choosing expected confidence levels and then calculates fractions of probabilistically accurate predictions (observed confidence values). However, the fraction calculation can use the u values, as mentioned earlier.

49

Before concluding on the connection between UQ calibration (ML community) and the u-pooling method (model validation community), we want to note that the u-pooling method could also be applied to assess the quality of the UQ of an ML model, with a different objective of measuring the degree to which each observation comes from the probability distribution predicted by the ML model, which differs from the objective of UQ calibration to test how underconfident or overconfident the ML model is. Similarly, the area metric or "u-pooling" metric can be used to measure the mismatch between predictive distributions and observations in a global sense [189].

4.2. Sparsification plots and metrics

Another method to assess the quality of predictive uncertainty is by creating the so-called *sparsification plot* [196]. A sparsification plot can be used to examine how well the predictive uncertainty of an ML model can serve as a proxy of the actual model prediction error, which is unknown without access to the ground truth. Creating a sparsification plot on a validation/test dataset consists of three steps. These three steps will be explained using the toy 1D regression problem from Sec. 4.1.1 (see Fig. 8).

Step 1: Given an uncertainty metric (e.g., variance for regression, entropy for classification), all samples in the validation/test dataset are sorted in descending order, starting with those with the highest predictive uncertainty. In the toy example, the 100 test samples are ranked according to the GPR model-predicted variance, with the first few samples having the largest predicted variances.

Step 2: A subset of samples (e.g., 2% of the validation/test dataset) with the highest uncertainty is gradually removed, leaving an increasingly smaller dataset whose samples have lower predictive uncertainty than those removed. In the toy example, the sample removal process involves 50 iterations, each of which takes out 2% of the remaining test samples with the highest predictive uncertainty.

Step 3: Given an error metric (e.g., RMSE, mean absolute error), the prediction error is computed on the remaining samples each time a subset of high uncertainty samples is removed in Step 2. The toy example uses the RMSE as the error metric, computed by comparing the GPR model-predicted means with the actual (noisy) observations.

Step 4: The final step is to plot the error metric vs. fraction of removed samples for the combinations obtained in Steps 2 and 3. Figure 13 shows the sparsification plot (dashed blue curve) for the toy example.

The resulting sparsification plot (see, for example, Fig. 13) visualizes how the prediction error changes as a function of the fraction of removed samples. If predictive uncertainty is a good proxy for prediction error, the error metric on a sparsification plot should decrease monotonically with the fraction of removed high-uncertainty samples, as is the case in Fig. 13. If ground truth is available, an ideal error curve (oracle) can be derived by ranking all samples in the validation/test dataset in descending order according to the actual prediction error. The oracle for the 1D toy regression
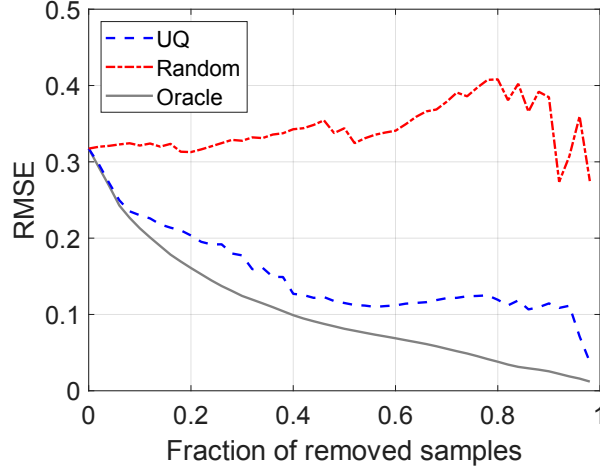
Figure 13: Sparsification curve and oracles for the toy 1D mathematical problem shown in Fig. 8.

sparsification_plot.pdf

problem is shown as a solid gray curve in Fig. 13, where we can observe a small difference between the calculated and ideal error curves. If predictive uncertainty is a perfect representation of model prediction error, the calculated error curve and oracle will overlap on the sparsification plot. On the other extreme, random uncertainty estimates that do not reflect prediction error meaningfully would result in an almost constant error on the remaining samples, i.e., a (close to) flat error curve. An example of the sparsification under random uncertainty estimates is shown in Fig. 13 for the 1D toy regression (the dash-dotted red curve). In this extreme case, a flat curve suggests that UQ provides little information about identifying problematic samples (e.g., OOD samples and those in regions of the input space with high measurement noise) whose model predictions may contain large errors.

Prior UQ studies in the ML community used plots similar to the sparsification plot to examine model accuracy as a function of model confidence [22, 197]. The only difference may be the label used for the x-axis, sometimes explicitly called *confidence threshold* for classification [22] and regression [197], instead of *fraction of removed samples*. Per-sample model confidence was derived as the probability of the predicted label for classification [22] and the percentage of validation/test samples whose variances are higher than the validation/test sample of interest [197]. However, estimating the per-sample model confidence from the per-sample predictive uncertainty without access to the ground truth is difficult and remains an open research question.

Since the model prediction error of one UQ approach on a validation/test sample most likely differs from that of a different approach, the ideal error curve (oracle) is likely to differ among UQ approaches. To compare these approaches, we can first calculate the difference between the sparsification and oracle for each fraction of removed samples, named *sparcification error*. Then, we can compute two sparsification metrics: (1) the Area Under the Sparsification Error curve (AUSE), i.e., the area between the actual error curve and its oracle [198], and (2) the Area Under the Random Gain curve (AURG), i.e., the area between the (close-to) flat random curve and the actual error curve. The lower the AUSE, the better the predictive uncertainty (derived from UQ) represents

the actual prediction error (unknown). The higher the AURG (assuming the error curve shows a monotonically decreasing trend), the better UQ is compared to no UQ.

4.3. Negative log-likelihood

Given a training dataset $\mathcal{D}$ and a validation/test data point $\mathbf{x}$, we can look at calculating the probability of observing its target value y using the predictive probability density function of the target, expressed as $\hat{p}(y|\mathbf{x}, \mathcal{D})$. We can repeat this process to get the probability of observing the target value for each sample in the validation/test dataset. Multiplying these predictive probabilities gives rise to a predictive likelihood. Taking a logarithmic transformation yields a predictive log-likelihood, which is a good measure of the goodness of fit of the probabilistic ML model to the validation/test data. The larger the log-likelihood, the better the model-data fit. Often, the negative counterpart of a log-likelihood, named NLL, is used in place of log-likelihood as the loss function or part of the loss function when training a probabilistic ML model. An example of the NLL has been given in Eq. (32) as the loss function for training a prababilisitic neural network in a neural network ensemble, as discussed in Sec. 3.3.1. It has been widely accepted that log-likelihood, or equivalently NLL, is a good measure of a probabilistic model's quality of fit [199]. NLL can be viewed as an indirect measure of model calibration [193] and is often used alongside calibration metrics to assess the quality of predictive uncertainty (see, for example, three recent methodological studies on UQ of ML models in [180, 181, 200]).

4.4. Accuracy vs. UQ quality

An interesting finding about UQ of ML models was reported in [193], where NLL was found to behave inconsistently with traditional accuracy measures, such as the RMSE or mean absolute error for regression, during model training. It appeared that NLL and accuracy could become conflicting at some point during the training process when neural networks could learn to be more accurate at the cost of lower quality in UQ, as reported for classification problems in [193]. This finding may help explain the observation in [201] that wide and deep neural networks trained with very limited regularization sometimes generalize surprisingly well [193]. Specifically, the inconsistency between NLL and accuracy provides evidence that (1) these large-scale models exhibiting good generalization performance may still suffer from the common overfitting issue, and (2) overfitting occurs only for a probabilistic error metric (e.g., NLL), not a classification error metric (e.g., classification accuracy) for classification or an error metric calculated based on mean predictions (e.g., RMSE or mean absolute error) for regression. Nonetheless, it is still important to understand how well a model does probabilistically by looking at UQ quality metrics, such as calibration metrics (Sec. 4.1), sparsification metrics (Sec. 4.2), and NLL (Sec. 4.3). Therefore, we strongly recommend academic researchers and industrial practitioners examine their ML models' performance in terms of both accuracy and UQ quality rather than focusing solely on accuracy metrics such as classification accuracy or RMSE. A seemingly highly accurate ML model may still have difficulties extrapolating to OOD samples, and it is crucial to estimate model confidence accurately through high quality UQ. We can now connect this discussion to an important statement in Sec. 2.3, i.e., all models are wrong, but some are useful [91].

5. UQ of ML models in prognostics

As stated in Sec. 1, our tutorial has an additional, secondary role, i.e., reviewing recent studies on engineering design and health prognostics applications of emerging UQ approaches. To make this tutorial focused, we place our review of engineering design applications in Appendix B and only present the review of health prognostics applications in the main text of this tutorial (i.e., the present section). We believe such an arrangement will provide the additional benefit of creating a methodological transition into the two case studies in Sec. 6 that are both related to health prognostics.

5.1. Uncertainty-aware ML for prognostics and health management

5.1.1. Prognostics and the role of UQ

PHM is an engineering field that focuses on developing techniques and tools to establish effective maintenance strategies that balance system availability and performance with operational requirements and maintenance costs [202, 203]. PHM comprises the main tasks of detecting the initiation of a fault (*fault detection*), distinguishing between different types of fault and isolating the root cause (*fault diagnostics*), and predicting the RUL (referred to as *prognostics* [202, 204]). Notoriously, prognostics represents the most challenging task among the three main tasks of PHM [14]. Effective prognostics enables *just-in-time maintenance* [205, 206], which holds the promise of significantly reducing maintenance costs and system downtime while prolonging the lifetime of industrial and infrastructure assets, thereby increasing system availability. Besides its potential in terms of cost savings, effective prognostics also enables more environmentally sustainable operations of industrial and infrastructure assets by lowering the frequency of replacement and reducing the consumption of spare parts and resources [203]. To be useful in mission- and safety-critical applications, successful prognostics approaches should be capable of not only predicting the RUL but also quantifying the associated uncertainty [207]. Knowledge of the associated uncertainty quantified in a principled manner allows users to conscientiously optimize the schedule of interventions and machine downtime with confidence rather than blindly relying upon the deterministic predictions of broadly applied black-box ML algorithms. In reality, inaccurate predictions of the end of life or RUL due to low quality UQ can have catastrophic consequences in safety-critical applications. For example, when an ML model makes overconfident predictions, it could either over- or under-predict the end of life and RUL. Significant overpredictions can lead to unexpected safety failures, while substantial underpredictions can lead to a shortened useful lifespan of components. Ensuring reliable uncertainty estimates from data-driven algorithms is essential to mitigate these problems and optimize safety and cost-effectiveness in maintenance operations. This involves preventing disruptive events by avoiding delayed replacements and minimizing costs by preventing premature maintenance actions, such as replacements or repairs.

5.1.2. The potential of DL for PHM

Recently, deep learning (DL) methods have become more prevalent in PHM applications. One of the major advantages offered by DL techniques in PHM is the ability to automatically analyze sensor

data, learn important features that characterize the system's health status, and track its changes over time until reaching the end of life. Industrial asset prognostics using DL can be implemented in two ways: directly predicting the RUL from sensor data or forecasting the future evolution of the system's health status until a pre-defined threshold is reached. The first approach, referred to as *direct mapping* [14], requires a dataset that links sensor readings to corresponding RUL target labels and is treated as a regression task. The second approach, called *time series forecasting* [14], involves identifying condition indicators that change in a predictable manner as the system deteriorates under different operational modes. These indicators may either be predetermined as strongly correlated with the machine's health and hence, interpretable, such as the internal resistance and capacity of a lithium-ion battery [208] or may be derived implicitly. A health indicator integrates several condition indicators into a single value, providing the user with information about the component's health status. The threshold for the health indicator, which may be subject to noise, also needs to be derived or learned. The importance of UQ in both approaches lies in the need to avoid unexpected safety-critical failures due to too-late replacements and to minimize costs by avoiding too-early replacements. UQ is, therefore, crucial to provide meaningful estimations and ensure accurate predictions in DL-based industrial asset prognostics. While quantifying the total predictive uncertainty (e.g., as a single variance value) already provides essential information for decision making, distinguishing between aleatory and epistemic uncertainty is equally important for prognostic applications. Particularly, considering that faults/failures are rare in safety-critical applications, epistemic uncertainty substantially impacts model performance due to the challenges in collecting representative run-to-failure datasets for training.

5.1.3. Uncertainty-aware DL in prognostics

Modern DL techniques can often not be directly interpreted by humans. The black-box nature of DL models is clearly at odds with the need for trustworthy prognostic algorithms. UQ can remediate this drawback, and its integration in DNNs is the subject of an exciting - yet constantly evolving - research field in the DL community [48, 134, 209–213], as discussed in Secs. 3.2-3.4. While a large number of research studies have focused on developing ML and DL approaches for providing point estimates of the RUL ([202, 203, 214] and the references therein), uncertainty-aware models, despite their great relevance, have not yet significantly impacted the research in this field.

In data-driven prognostics, the models' predictions are inevitably affected by various sources of uncertainty. These sources of uncertainty include model-form uncertainty, insufficient representative historical data for model training, as well as errors in measurement and communication transmission, among others (refer to Table 1). While ML and DL approaches have been increasingly applied for prognostics, most developed algorithms did not quantify the associated uncertainty. This limitation, among other factors, has prevented such approaches from being practically deployable in real mission- and safety-critical applications. UQ plays a vital role in enabling ML and DL to deliver high value in practical health prognostics applications. By instilling greater confidence in the predictions and streamlining the integration of the results into maintenance planning and scheduling, UQ reinforces user trust and enhances the effectiveness and safety of these applications

[46, 202, 207, 215–217].

5.2. Uncertainty evaluation metrics for prognostics

While UQ for prognostics already significantly benefits from the standard UQ performance evaluation metrics commonly applied in other disciplines as well, such as NLL, the MSE, the RMSE, or the mean absolute percentage error (MAPE), the specificity of the prognostics problem often requires a set of customized metrics. One of the particularities of RUL predictions is for example that the closer the predictions progress to the end of life, the more certain the models should behave when making estimations about the predicted end of life. Therefore, the performance evaluation metrics should take such behavior into consideration and provide quantitative evaluation for it.

Metrics, such as MSE or MAPE, do not take into account the statistical distribution of the RUL predictions around the ground-truth values. To account for such statistical deviations, a number of more informative probabilistic metrics have been introduced for applications in prognostics. Most of these metrics are built under the assumption that predicting the RUL at the initial time steps of the machine operation, is much harder and, as progressively additional information is acquired, the prediction task also gets simplified thanks to the fact that the severity of the fault increases and the corresponding symptoms tend to become more pronounced as the system approaches the end of life.

In the seminal work of Saxena et al. [218], the authors introduce four performance evaluation metrics for prognostics - meant to be measured sequentially - assessing different aspects of the RUL prediction problems, namely: the Prognostic Horizon, the α-λ performance, the Relative Accuracy, and the Convergence (Fig. 14). While these performance evaluation metrics have mainly targeted physics-based prognostic methods, they are also applicable to DL-based UQ approaches. In the following, we briefly review their definitions and rationales. We refer interested readers to the original paper for more details. In essence, Prognostic Horizon is defined as the difference between the time step when the predicted RUL first meets the specified performance criteria and the time index for the end of life. The performance criteria are met if the predicted RUL value falls within an area determined by the ground-truth RUL value plus/minus a certain pre-selected confidence interval (called α). The metric can be easily adapted to cases where the output of the model is probabilistic. In that case, the criterion is met if the probability of the predicted RUL falling within the previously defined area is larger than β, an additional parameter to be chosen a priori (Fig. 14).
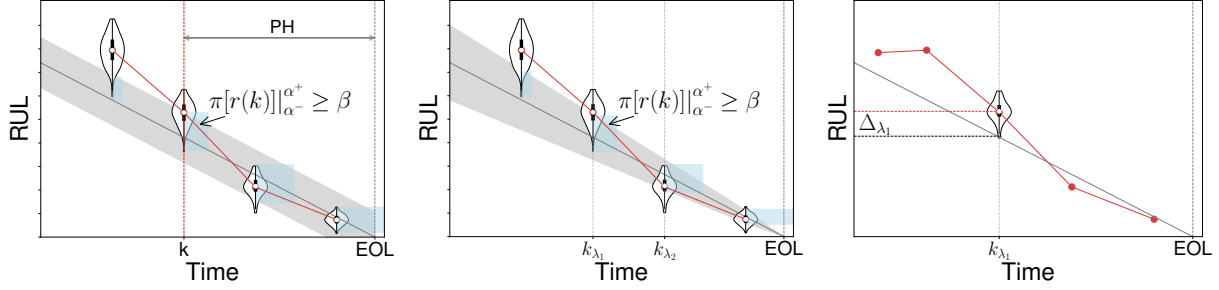
Figure 14: (Left) Prognostic Horizon (PH): here $[\pi(r(k))]_{\alpha^-}^{\alpha^+}$ indicates the probability that the distribution of the prediction r at time k falls within the confidence region $[r^*(k) - \alpha^-, r^*(k) + \alpha^+]$ (grey area), and β is a pre-determined threshold; (Middle) α-λ metric calculated at k_{λ_1} and k_{λ_2}: same notation as before, note that the confidence bounds around the ground-truth shrink as the end of life is approached; (Right) Relative Accuracy calculated at k_{λ_1}: Δ_{λ_1} indicates the difference between the median of the predictive distribution and the ground-truth value.

The α-λ metric is very similar to the Prediction Horizon but it differs in two aspects: first, it is binary, if the criterion is met at a certain time step, its value will be one, otherwise 0. Second, the confidence bounds around the ground-truth RUL are now a function of the predicted RUL and, as a result, will tend to shrink as the machine approaches the end of life.

The relative accuracy is simply calculated as one minus the relative error of the model with respect to the ground truth at a certain time step. In particular, the relative error is computed by taking the ratio between the absolute difference between ground truth and a properly-chosen central tendency point estimate of the predicted RUL distribution, and the ground truth RUL value. The central tendency point estimate of the prediction distribution is arbitrary and depends on the statistical properties of the predictive distribution (Gaussian, mixture-of-Gaussians, multi-modal, etc.). Finally, the Convergence acts as a meta-metric to measure how quickly each of the above metrics improves over time.

5.3. Discussion

Meaningful uncertainty estimates are crucial for ensuring the safe and reliable deployment of DL models in real-world applications, especially for safety-critical assets. This is essential to build trust in the models and ensure their effectiveness. This is because, in practice, decision making in the context of industrial applications involves a complicated trade-off between risky decisions and large potential economic benefits. DL has undoubtedly advanced the field by offering a valuable set of tools to efficiently learn from data and automate the entire prognostics process. Nevertheless, this is only one part - yet very significant - of the challenges arising in prognostics. ML and DL techniques need to be as trustworthy and reliable as possible, and for this reason, effective UQ and its integration into existing techniques remain an essential desideratum.

In previous research studies, MC dropout has been by far the most widely employed strategy for tackling UQ of neural networks, especially DNNs. There are likely two reasons for this: first, the interpretation of MC dropout is very intuitive; and second, it requires only a minimal modification to existing architectures, namely activating dropout layers at training time. Nevertheless, as shown in multiple studies [22, 219, 220], the UQ performance of MC dropout is not always satisfactory,

and more advanced solutions should be explored. Fortunately, the fields of UQ and Bayesian DL are constantly progressing, and applications of the resulting techniques to prognostics are an important research area to be further explored [48, 134, 209–213].

In addition, uncertainty-aware ML methods have been mainly used in the context of prognostics for RUL prediction. While this is arguably the most important end goal in this field, several other avenues could be investigated in the future. An example is, for instance, anomaly detection. In this setting, uncertainty can be used to detect abnormal health states in the machine operation by evaluating the level of confidence of the model corresponding to that time step. The assumption is that a high level of epistemic uncertainty associated with a certain input will be indicative of test data points that are less representative of the training data distribution. Hence, such data will probably correspond to unusual health states, assuming the training data are collected from a machine operating in a nominal regime.

To conclude, a crucial criterion for any UQ technique used in prognostics is the ability to accurately disentangle aleatory and epistemic uncertainty. These two measures contain distinct types of information and, therefore, must be interpreted separately to ensure appropriate analysis.

6. Case studies for benchmarking – Code Sharing on GitHub

In this section, we benchmark the performance of several UQ methods in two engineering applications: (1) early life prediction of lithium-ion batteries and (2) RUL prediction of turbofan engines. In both case studies, we built UQ models with publicly available datasets and compared the models' performance. To ensure a fair comparison, these UQ models are built with nearly identical backbone architectures wherever applicable. These two case studies are widely used in the literature due to their broad significance in safety-critical applications and, therefore, a comprehensive understanding of the performance of different UQ methods helps to identify the right model to deploy in a particular application. A code walk-through is provided for the first case study to demonstrate the practical implementation of UQ methods. We acknowledge that there could be several other ways of implementing the same UQ models using different sets of libraries. In this discussion, we try to limit ourselves to using only TensorFlow and Keras libraries for building the neural network models.

6.1. Case study 1: Battery early life prediction

In this section, we explore the utility of various UQ for ML model methods to tackle the early life prediction of lithium-ion batteries. The dataset used in this case study consists of run-to-failure data from 169 LFP/graphite APR18650M1A cells with a nominal capacity of 1.1Ah [221, 222]. The goal of this case study is to predict, with confidence, the remaining cycle life of lithium-ion cells based on data collected only in the first 100 cycles. This early life prediction is a challenging problem as most cells do not exhibit significant levels of degradation during the first 100 cycles. Therefore, it is important for researchers to associate each prediction with an uncertainty estimate.

The code for this case study can be found at our Github page. In this section, we take the opportunity to provide a brief walk-through of the code while discussing the following UQ methods:

(1) neural network ensemble, (2) MC dropout, (3) GPR, and (4) SNGP. The goal of this study is to compare several UQ methods with comparable prediction accuracy based on the current literature. The neural network-based models, namely neural network ensemble, MC dropout, and SNGP, are built on a ResNet with a similar backbone architecture as shown in Fig. 15.
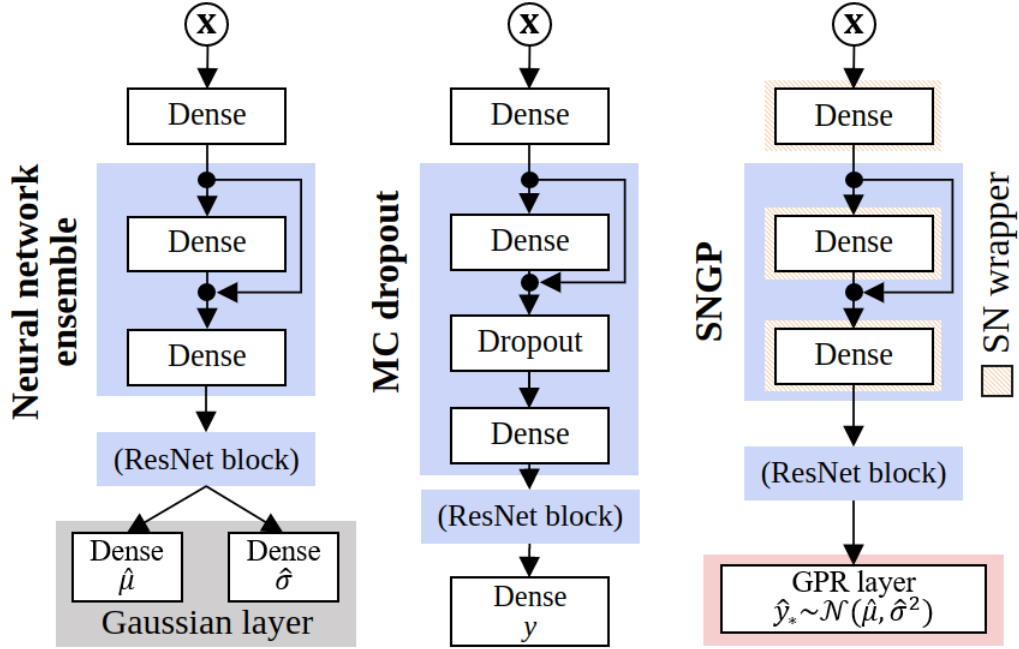

Figure 15: UQ model architectures with ResNet backbone used in case study 1. The ResNet block for each model is defined by the blue box.

6.1.1. Dataset overview

The 169 LFP cell dataset is a combination of the 124-cell dataset provided by Severson et al. [221] and the 45-cell dataset provided by Attia et al. [222]. These 169 cells are divided into three subsets as described in Table 3, where the partition for training, primary test, and secondary test datasets is consistent with that of Severson et al. [221], and the dataset from Attia et al. [222] is used as the tertiary test dataset. The 169 LFP cells underwent different fast-charge protocols and storage time, but they had identical discharging conditions, which in turn led to a diverse set of capacity trajectories as illustrated in Fig. 16. Similar to the existing literature, we assume a cell to have reached the end of life when its capacity reaches 80% of the nominal value (cutoff of 0.88Ah). A more detailed description of the battery cycling tests and raw data can be found at https://data.matr.io/1/.

The cycle-to-cycle evolution of voltage as a function of discharge capacity $V(Q)$ is often captured when conducting the experiments. However, the authors of the original dataset Severson et al. [221] hypothesize and prove that the inverse relationship, where the discharge capacity as a function of voltage $Q(V)$ during the early cycles carries sufficient information to accurately predict the cycle life. We adopt a similar strategy of using $\Delta Q_{100-10}(V) = Q_{100}(V) - Q_{10}(V)$ as the input to our

Table 3: Summary of LFP battery dataset

Type	No. of cells
Training	41
Primary test	43
Secondary test	40
Tertiary test	45
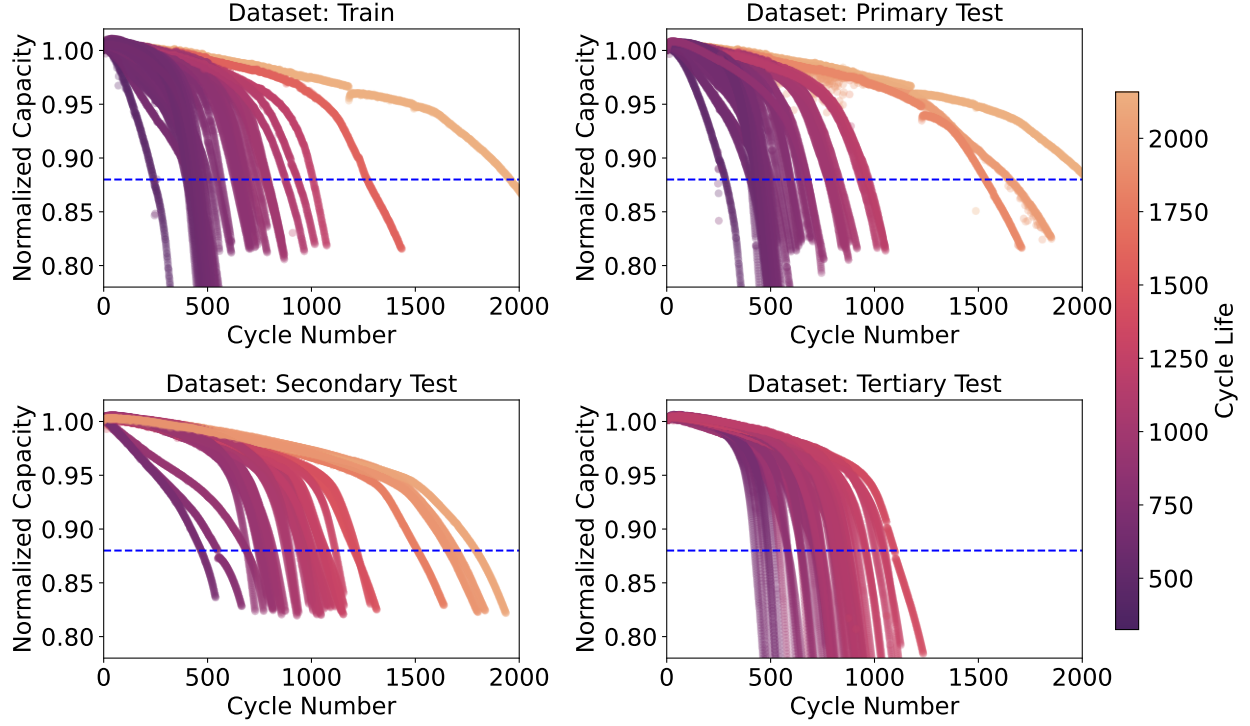
Figure 16: Normalized capacity curves for the four datasets mentioned in Table 3.

UQ models. Similar to Severson et al. [221], we find that the cycle life is significantly correlated with $Var(\Delta Q_{100-10}(V))$ as shown in Fig. 17.

6.1.2. Neural Network Ensemble

We first develop a neural network ensemble model (NNE) following the discussion from Sec. 3.3. Particularly, we develop a neural network learning framework following the work by Lakshminarayanan et al. [22]. Each individual model of the ensemble consists of a Gaussian layer as the final layer, and the Gaussian layer outputs a predicted mean μ and variance σ^2 for a given input $\mathbf{x}$. Parameters $\boldsymbol{\theta}$ of the neural network are trained to minimize the NLL loss function defined in Eq. (31) earlier, which corresponds to the implementation below:

```python
def custom_loss(variance):
    def nll_loss(y_true, y_pred):
        return tf.reduce_mean(0.5*tf.math.log((variance)) +
                              0.5*tf.math.divide(tf.math.square(y_true - y_pred),
                                                 variance)) + 1e-6
```
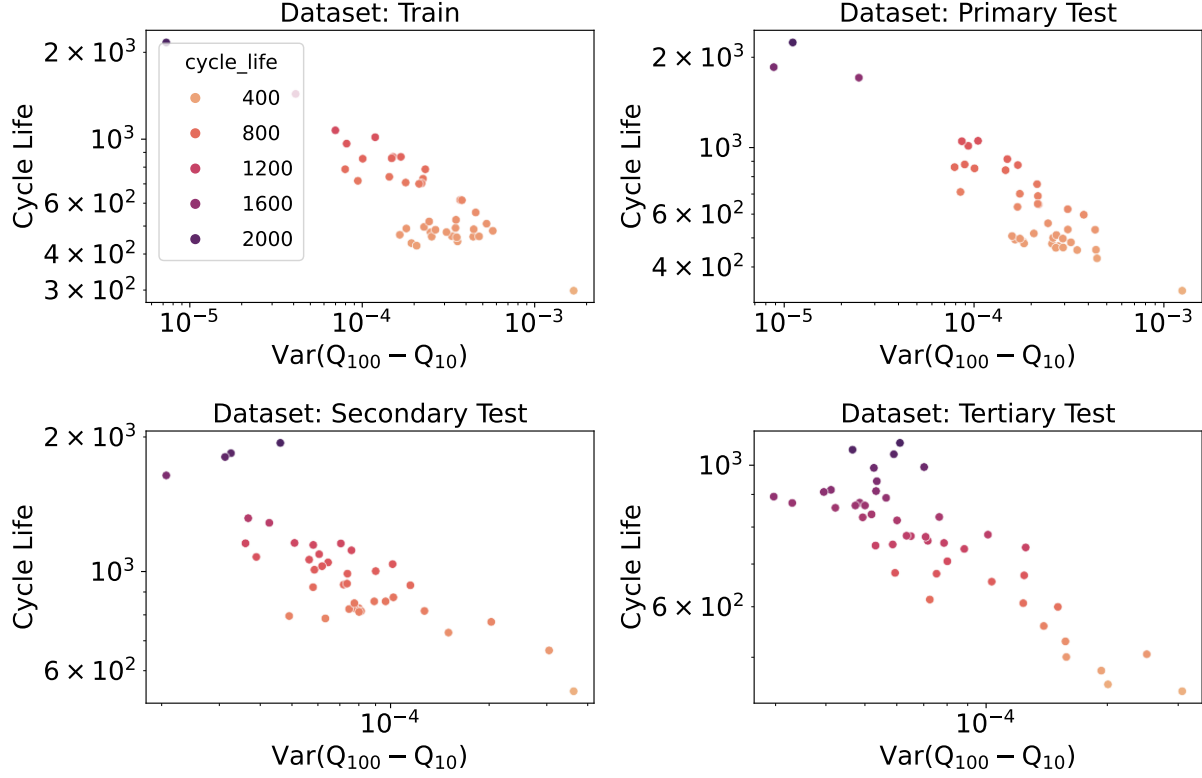
Figure 17: Correlating cycle life with $Var(\Delta Q_{100-10}(V))$.

```
                                                        ┌─────────────────────┐
    return nll_loss          ◄────────────────────────  │ NLL loss in Eq. (31)│
                                                        └─────────────────────┘
```

In the code below, the Gaussian layer uses two kernels and biases to characterize μ and σ by splitting the output of the previous layer (traditionally a fully connected layer with one dimension). Note that the kernel shape should be compatible with the number of hidden units in the previous dense layer.

```python
class GaussianLayer(Layer):
    def build(self, input_shape):
        self.kernel_1 = self.add_weight(shape=(10, self.output_dim),...)
        self.kernel_2 = self.add_weight(shape=(10, self.output_dim),...)
        ... # (define bias_1 and bias_2)
    def call(self, x):
        output_mu  = K.dot(x, self.kernel_1) + self.bias_1
        output_var = K.dot(x, self.kernel_2) + self.bias_2
        output_var_pos = K.log(1 + K.exp(output_var)) + 1e-06
        return [output_mu, output_var_pos]
```

Two kernels + biases to split the output

Make variance positive

Output mean and variance

Finally, a neural network model is constructed by appending the Gaussian layer to a simple ResNet model. The architecture for each individual model of the neural network ensemble is shown

in Table 4.

Table 4: Individual model of the neural network ensemble
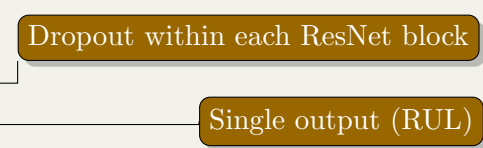
Layer	Output Shape	No. of Parameters
Input	[(None, 1000)]	0
Fully connected	(None, 100)	100100
Fully connected	(None, 50)	5050
Fully connected	(None, 50)	2550
Fully connected	(None, 50)	2550
Fully connected	(None, 50)	2550
Fully connected	(None, 10)	510
Gaussian layer	[(None, 1), (None, 1)]	22
Total trainable parameters		113332

In total, we independently trained 15 models by randomizing the initialization of model weights in addition to shuffling the training samples. The size of the neural network ensemble is determined based on the elbow method - see Fig. 18 for more details. Each individual model is trained for 300 epochs (based on validation split/validation loss to test for overfitting).

6.1.3. MC Dropout

In this section, a simple MC dropout model is developed following the method described in Section 3.2.3. The only differences between the implementation of the MC dropout and the neural network ensemble are (1) the inclusion of dropout layers with dropout being active during the prediction phase and (2) having a single deterministic output as the final output. Note that the dropout layer can also be introduced in other UQ methods, for example, in neural network ensembles, to mitigate overfitting. However, dropout is typically not activated during the prediction phase in such models. In the case of MC dropout, the output varies from one prediction run to another, where a certain percentage of neural network weights from the trained model are randomly dropped out at the prediction phase. The code snippet below showcases our implementation of the dropout layers within the ResNet block as shown in Fig. 15.

```
for _ in range(num_res_layers): # for each residual block
    x  = Dense(50, activation = actfn)(x)
    x1 = Dense(50, activation = actfn)(x)
    x = x1 + x
    x = Dropout(rate = 0.10)(x)          Dropout within each ResNet block
mu = Dense(1, activation = actfn)(x)     Single output (RUL)
model = Model(feature_input, mu)
```
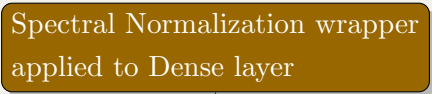
The MC dropout model architecture and trainable parameters are similar to Table 4 except for the presence of dropout layers with a 10% dropout rate. During the prediction phase, the trained MC dropout model is run 15 times with dropout enabled (the ensemble size was determined based on the elbow method - see description for Fig. 18). An ensemble of all the individual deterministic RUL predictions produces the RUL prediction with uncertainty quantified.

6.1.4. Spectral Normalization Gaussian Process (SNGP)

Next, we implement the SNGP model discussed in Section 3.4 with the core idea of preserving distance awareness between training and test/OOD distributions when producing the uncertainty for each prediction. This is achieved by: (1) applying spectral normalization to the hidden layers of the neural network and (2) replacing the final layer with a Gaussian process layer. This is a single-model method with high performance in OOD detection.

Following Liu et al. [180] and a corresponding tutorial of TensorFlow, as shown below, we first define a model class *FC_SNGP* inherited from the class of TensorFlow model. In this model class, we wrap some dense layers with the spectral normalization layer, where the normalization threshold has a constant value of *spec_norm_bound*. The *RandomFeatureGaussianProcess* layer with RBF kernel serves as the Gaussian process layer.

```
import official.nlp.modeling.layers as nlp_layers
class RN_SNGP(tf.keras.Model):
        ...
        self.dense_layers1 = nlp_layers.SpectralNormalization(
                self.make_dense_layer(100),norm_multiplier=self.spec_norm_bound)
        ...
    def make_output_layer(self, no_outputs):
        """Uses Gaussian process as the output layer."""
        return nlp_layers.RandomFeatureGaussianProcess(no_outputs,
                                    gp_cov_momentum=-1,**self.kwargs)
```

Spectral Normalization wrapper applied to Dense layer

The value of *gp_cov_momentum* in the above figure decides if the calculated covariance is exact or approximated. A positive value of *gp_cov_momentum* updates the covariance across the batch using a momentum-based moving average technique, whereas a value of -1 calculates the exact covariance. Since the calculation of covariance could be affected by the batch size, it is recommended that the covariance matrix estimator be reset during each epoch. This can be done using Keras API to define a callback class and then appending it to *FC_SNGP*. Finally, we train an SNGP model with the ReLU activation function and *spec_norm_bound = 0.9*.

```
class ResetCovarianceCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        """Resets covariance matrix at the beginning of the epoch."""
        if epoch > 0:
            self.model.regressor.reset_covariance_matrix()
```

6.1.5. Gaussian Process Regression

At last, a standard GPR model with RBF kernel is trained using the scikit-learn Python package. The hyperparameters of the GPR models, such as length scale, are optimized using grid search during model fitting.

6.1.6. Evaluation/Results

In this section, we exploit the following metrics to quantitatively examine the uncertainty quantification performance of all the models: (1) root mean square error (RMSE), (2) average NLL defined in Eq. (31), (3) expected calibration error (ECE) as defined in Section 4.1.3, and (4) calibration curve introduced in Section 4.1.1. Since both neural network ensemble and MC dropout require an ensemble of individual models, it is essential to determine the ensemble size. Ideally, it is preferred that an ensemble has as many individual models as possible so that all the potential variations get manifested during the prediction stage. In other words, an ensemble benefits from models that undergo diverse learning paths and this would effectively capture the variations in predictions. However, beyond a certain ensemble size, the learning becomes increasingly less diverse and only trivially contributes to the ensemble at the expense of increased computational cost. Therefore, inspired by the elbow method, we systematically vary the ensemble size for constructing the neural network ensemble and MC dropout models while capturing the training RMSE and ECE as shown in Fig. 18. RMSE and ECE are chosen to strike a trade-off between accuracy and uncertainty quantification capabilities. Based on this study, we choose an ensemble size of 15 for both neural network ensemble and MC dropout.
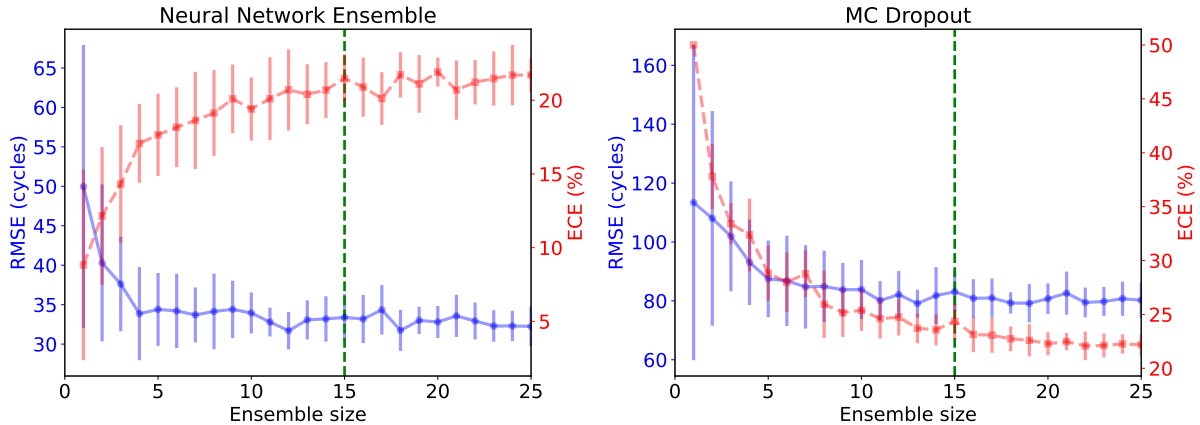
Figure 18: Determining the ensemble size for neural network ensemble and MC dropout. The selected ensemble size for this case study is determined by the green vertical line.

Table 5 reports the RMSE, NLL, and ECE across different UQ methods for the dataset described in Table 3. The variation in Table 5 results from 10 end-to-end independent runs. Note that the results may not be the best that each method could offer as all these methods are built on a backbone of a simple ResNet architecture except for GPR. It is likely that different UQ methods would require different architectures to obtain the best results. From Table 5, we observe that the GPR model perfectly fits the 41 training data points with an RMSE of zero and an extremely low NLL. However, GPR exhibits poor generalization when learning, as can be seen in the large RUL prediction error as well as high uncertainty at testing. In particular, for the secondary and tertiary test datasets that are known to be significantly different from the training dataset, the performance of GPR gets even worse. *Secondly*, the non-ensemble SNGP model performs much better in generalization when

Table 5: Performance comparison across UQ methods for the 169 LFP cell dataset in terms of MSE ± standard deviation

Dataset	NNE	MC	SNGP	GPR
	RMSE (cycles) ↓			
Train	68.1±22.1	69.4±16.8	34.8±14.7	0.0±0.0
Primary test	137.3±20.9	149.9±18.4	148.1±16.2	141.1±0.0
Secondary test	205.1±27.4	194.1±15.1	249.3±33.6	319.0±0.0
Tertiary test	183.9±46.9	195.0±29.1	258.9±60.3	406.5±0.0
	NLL ↓			
Train	4.7±0.3	8.6±2.6	5.6±0.02	-3.8±0.0
Primary test	5.4±0.2	14.3±6.5	5.7±0.03	5.7±0.0
Secondary test	5.7±0.2	6.9±1.3	6.1±0.2	6.0±0.0
Tertiary test	5.7±0.1	9.2±1.7	5.9±0.1	6.4±0.0
	ECE (%) ↓			
Train	29.8±3.7	15.2±6.8	42.5±3.0	49.9±0.0
Primary test	10.5±5.0	24.4±5.3	21.5±2.3	6.9±0.0
Secondary test	13.5±5.7	9.5±4.6	12.7±4.6	10.4±0.0
Tertiary test	9.8±4.5	22.6±3.4	9.3±4.4	8.0±0.0

compared to GPR. The presence of neural network layers helps condense crucial information in the hidden space which is further enhanced by the spectral normalization wrapper. But we generally found in this case study that SNGP tends to generate unnecessarily large uncertainty for each prediction, thus resulting in a large NLL and ECE. *Third*, among the two ensemble-like models, the neural network ensemble performs slightly better than MC dropout in terms of accuracy but exhibits a substantial advantage in UQ over MC dropout. We observe that the MC dropout predictions are generally overconfident with a low uncertainty estimate $\hat{\sigma}_{\text{RUL}}$ for each prediction. This low $\hat{\sigma}_{\text{RUL}}$ leads to large NLLs along with increased run-to-run variation. In the case that there is a larger $\hat{\sigma}_{\text{RUL}}$, small changes in $\hat{\mu}_{\text{RUL}}$ do not significantly affect the run-to-run variation. On the other hand, when $\hat{\sigma}_{\text{RUL}}$ is small, run-to-run variation of NLL becomes more sensitive to the changes in $\hat{\mu}_{\text{RUL}}$ around the true RUL. Note that the dropout rate hyperparameter of the MC dropout model significantly affects the model performance. A low dropout rate would lead to almost identical models within the ensemble, leading to very low predictive uncertainty and, thus, an overconfident model. On the contrary, a larger dropout rate could cause significant differences between different runs, thereby increasing uncertainty while compromising accuracy. *Lastly*, the better UQ ability of the neural network ensemble can be primarily attributed to the ability of each individual model within the ensemble to provide aleatory uncertainty, which during the ensemble process provides a more holistic picture of uncertainty.
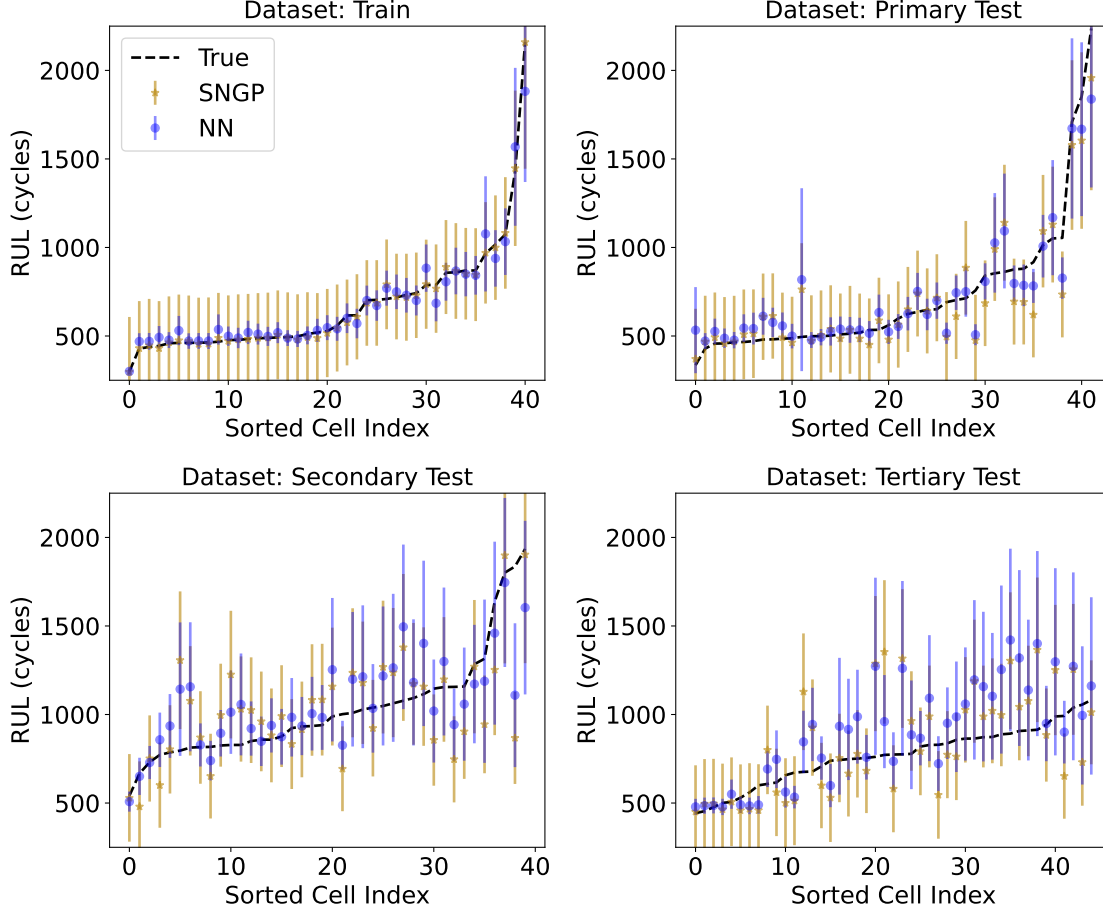
Figure 19: RUL prediction error curves with cells sorted based on true RUL values.

Next, we visualize the prediction error with respect to a single end-to-end run for neural network ensemble and SNGP in Fig. 19. To better depict prediction accuracy and the uncertainty estimate pertaining to each prediction, we plot the error curve associated with each cell in the dataset by their RUL in ascending order. As can be observed, regarding the training data, the mean RUL predictions of both SNGP and neural network ensemble models highly align with the true RUL prediction. In the case of the primary and secondary test datasets, a few instances of discrepancy between the mean RUL prediction and ground truth arise. However, these models fail to capture the true RULs of the tertiary test dataset, which is well known to be significantly different from the other three datasets. Another interesting observation across the first three considered datasets is that SNGP tends to yield a large uncertainty estimate for almost all predictions. As a result, SNGP is underconfident in most cases. In contrast, the neural network ensemble model produces significantly lower prediction uncertainty than SNGP. Only in the case of the tertiary test dataset, both neural network ensemble and SNGP associate large $\hat{\sigma}_{\text{RUL}}$ to most of the batteries.

In what follows, we construct the calibration curve based on each model's performance on the four datasets. As illustrated in Fig. 20, the shaded area of each curve characterizes the run-to-run

65
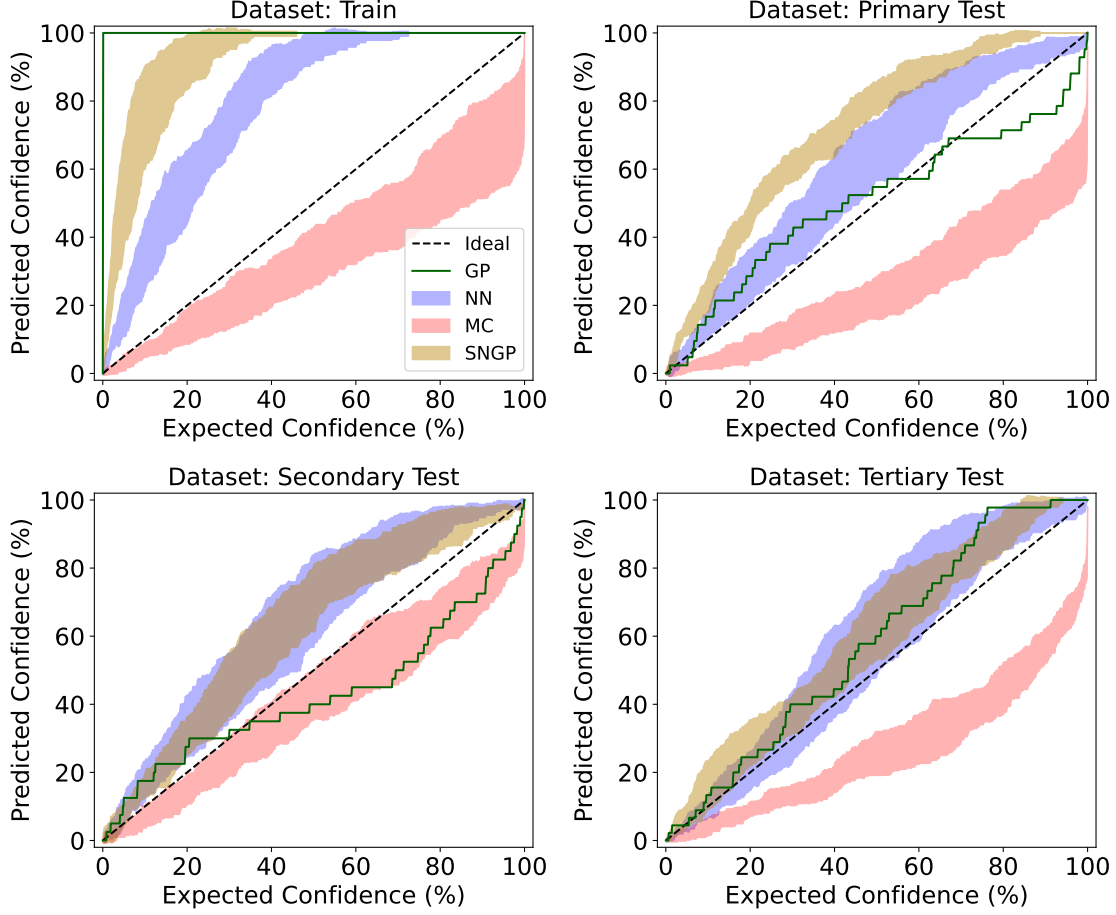
Figure 20: Calibration curves for the four models on all the datasets of the 169 LFP cell dataset. The shaded area captures the run-to-run variation of all the models.

variation over 10 independent trials. *First*, since the GPR model fits the training data perfectly (zero RMSE), the observed confidence is 100% and does not change with the expected confidence level. For the other datasets, GPR seems to be the closest to the expected line leading to the least ECE (see Table 5). *Next*, we observe that both GPR and SNGP are relatively stable irrespective of model initialization leading to low run-to-run variation. On the other hand, models like neural network ensemble and MC dropout exhibit higher run-to-run variation (with MC dropout having the highest run-to-run variation), especially when considering OOD datasets like the tertiary dataset. These observations regarding model stability are in line with our qualitative comparison of UQ models summarized in Table 2. *Lastly*, MC dropout is generally overconfident across all the datasets, as reflected in the relatively low uncertainty associated with each RUL prediction. Different from MC dropout, neural network ensemble, and SNGP are consistently underconfident. Considering the safety-critical nature of early life prediction of batteries, underconfident models are desirable as they allow end users to stay on the safe side.

6.2. Case study 2: Turbofan engine prognostics

In this section, similar to Case Study 1, we evaluate the performance of multiple UQ methods in predicting the RUL of nine turbofan engines that operate under varying conditions. To carry out our analysis, we utilize the New Commercial Modular Aero-Propulsion System Simulation (N-CMAPSS) prognostics dataset [223], which has been recently open-sourced. Specifically, we use the sub-dataset DS02, which has been used in several previous works, see Refs. [224–226]. Our objective is to predict the target RUL by employing a set of multivariate time series as inputs. In addition to providing a point estimate of the RUL, our aim is to quantify the uncertainty associated with the RUL prediction with the UQ methods surveyed in this paper. The code for this case study is available on our Github page. The primary goal of this study is to pedagogically compare various UQ methods that exhibit similar prediction accuracy based on the current literature. We do not make any claims that the discussed methods outperform the existing literature's models.

6.2.1. Dataset overview

Table 6: Overview of the input variables. These condition monitoring signals include both scenario descriptors (first 6 rows) and measured physical properties (last 14 rows). The symbol used for each variable corresponds to its internal name in the CMAPSS dataset.

Variable No	Symbol	Description	Unit
1	alt	Altitude	ft
2	XM	Flight Mach number	-
3	TRA	Throttle-resolver angle	%
4	T2	Total temperature at fan inlet	°R
5	Nf	Physical fan speed	rpm
6	Nc	Physical core speed	rpm
7	Wf	Fuel flow	pps
8	T24	Total temperature at LPC outlet	°R
9	T30	Total temperature at HPC outlet	°R
10	T40	Total temp. at burner outlet	°R
11	T48	Total temperature at HPT outlet	°R
12	T50	Total temperature at LPT outlet	°R
13	P15	Total pressure in bypass-duct	psia
14	P2	Total pressure at fan inlet	psia
15	P21	Total pressure at fan outlet	psia
16	P24	Total pressure at LPC outlet	psia
17	Ps30	Static pressure at HPC outlet	psia
18	P30	Total pressure at HPC outlet	psia
19	P40	Total pressure at burner outlet	psia
20	P50	Total pressure at LPT outlet	psia

This case study comprises a collection of run-to-failure trajectories for a fleet of nine aircraft engines that operate under authentic flight conditions [223]. We use the open-source code presented in Ref. [227] to download and preprocess the data. For every RUL prediction time step, the input

to the UQ model is a 20-dimensional vector that represents the measured physical properties of the engine as well as the scenario descriptors characterizing the engine's operating mode during the flight. At each time step, the UQ model produces RUL and its associated uncertainty as outputs. Table 6 provides an overview of the input variables used in the model. As we adopted a purely data-driven approach, we did not utilize the virtual sensors or the calibration parameters that are available in the N-CMAPSS dataset [223, 228].

Consistent with Ref. [228], we split the entire dataset into a training dataset, which comprises the time-to-failure trajectories of six units (i.e., units 2, 5, 10, 16, 18 and 20), and a testing dataset, which includes the trajectories of three units (i.e., units 11, 14 and 15). Figure 21 illustrates the distributions of the flight conditions across all units and provides an example of a flight cycle obtained by traces of the scenario-descriptor variables for unit 10. Finally, to address the memory consumption concerns associated with the size of the dataset, we downsampled the data by a factor of 500 by using the code from Ref. [227], thus resulting in a sampling frequency of 0.002 Hz.
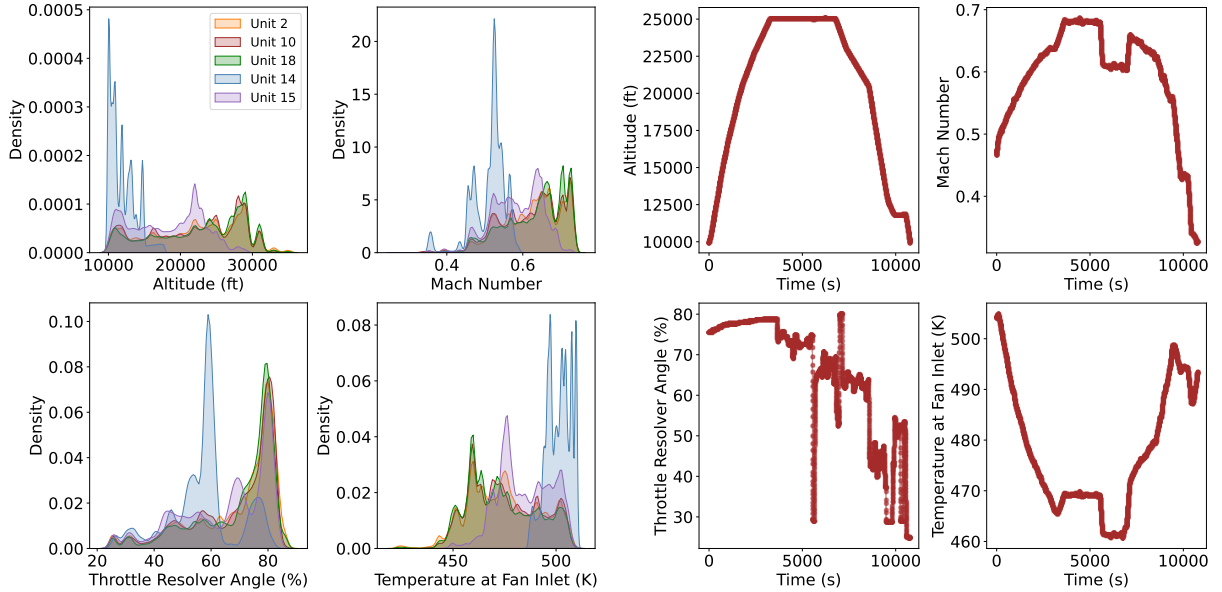
Figure 21: (Left) The flight envelopes simulated for climb, cruise, and descend conditions were estimated using kernel density estimation based on measurements of altitude, flight Mach number, throttle-resolver angle, and total temperature at the fan inlet. The densities of these measurements are shown for three representative training units ($u = 2$, 10, and 18) and two test units ($u = 14$ and 15). (Right) A typical flight cycle for unit 10 with traces of the scenario-descriptor variables depicting the climb, cruise, and descend phases of the flight, covering different flight routes operated by the aircraft, where altitude was above 10,000 ft.

6.2.2. Evaluation/Results

For the sake of clarity and consistency, in this case study, we have used the same code structure/functions from the previous case study. However, we have excluded GPR from our evaluation due to the large size of the dataset and the well-known scaling issues associated with this UQ method. For further implementation details, we refer the reader to the detailed descriptions in the previous case study or to the code implementation on GitHub.

The performance of NNE, MC, and SNGP on the three test units is compared in Table 7 using RMSE, NLL, and ECE metrics. Overall, NNE seems to outperform MC and SNGP in terms of all the metrics considered, with SNGP providing slightly better performance than MC. Figure 22 shows that all the three models are able to capture the decreasing trend of the RUL over time, but they encounter difficulties at the beginning of the trajectory, i.e., at the onset of degradation. Interestingly, NNE appears to address this issue by assigning higher uncertainty corresponding to such points.

Table 7: Comparison of the error metrics across different UQ methods on the N-CMAPSS dataset

Dataset	NNE	MC	SNGP
	RMSE (cycles) ↓		
Train	7.1±0.1	10.2±0.1	8.7±0.7
Unit 11	8.5±0.5	10.0±0.3	8.9±1.8
Unit 14	7.4±0.2	11.5±0.1	9.3±1.4
Unit 15	4.8±0.3	8.2±0.2	6.8±1.2
	NLL ↓		
Train	2.0±0.0	3.7±0.1	4.4±0.7
Unit 11	2.3±0.1	3.0±0.1	4.8±1.8
Unit 14	2.2±0.0	4.2±0.2	4.4±1.3
Unit 15	1.8±0.0	2.8±0.1	3.1±0.6
	ECE (%) ↓		
Train	6.2±0.8	12.8±1.2	9.6±2.7
Unit 11	15.1±2.5	19.6±1.5	15.9±7.3
Unit 14	5.8±1.0	25.1±1.2	13.0±3.5
Unit 15	14.9±2.7	11.5±1.6	8.5±3.0
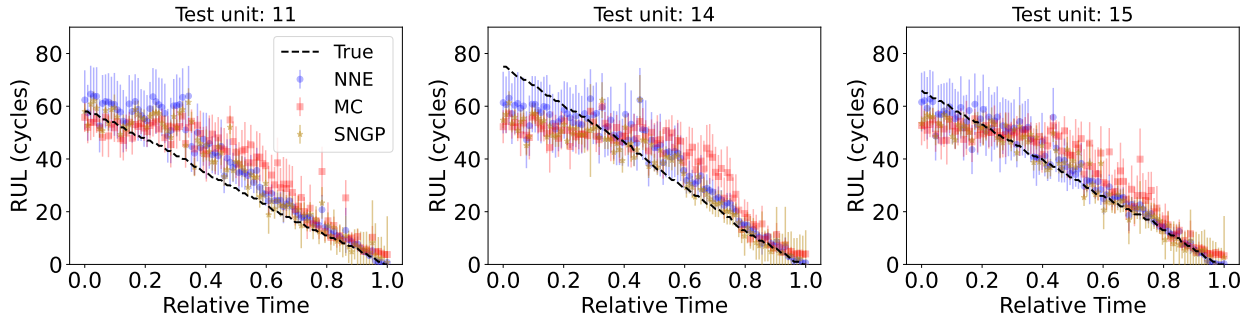
Figure 22: RUL prediction error curves for the N-CMAPSS dataset.

The calibration curves presented in Fig. 23 suggest that the methods used in this study tend to produce over-confident predictions, particularly for unit 11. This overconfidence can have serious implications for safety in prognostics. While MC exhibits overconfidence across all test units, NNE performs best on unit 14 and SNGP on unit 15, displaying a calibration curve that is closer to the ideal. Overall, NNE generally outperforms other UQ models as demonstrated by its accurate predictions (i.e., low RMSE and NLL scores). Furthermore, NNE's calibration curve is more closely

aligned with the ideal leading to low ECE values.
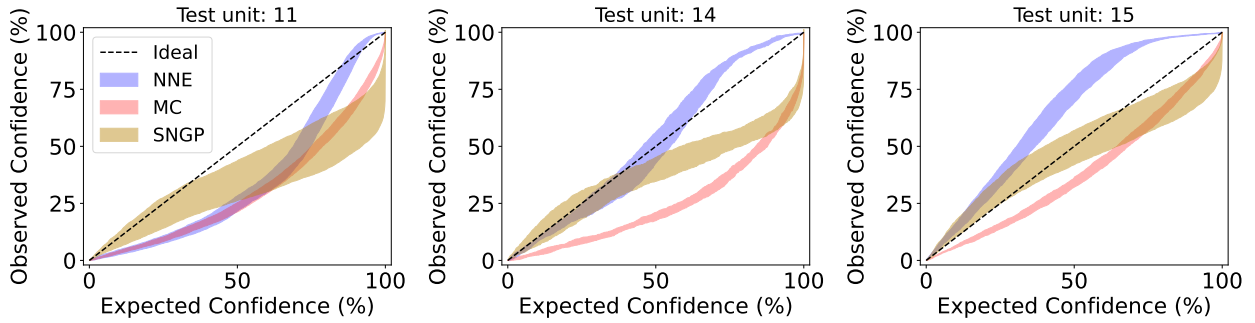
Figure 23: Calibration curves for the three models on all the datasets. The shaded area captures the run-to-run variation of all the models.

As a final remark, we would like to acknowledge that the present results could be improved by optimizing the hyperparameters of each model individually, i.e., the number of layers and nodes, the dropout rate, the number of ensemble components, and the type of activation functions. However, the present study serves as a solid foundation for investigating the UQ capabilities of the analyzed methods in challenging and realistic case studies.

7. Other topics related to UQ of ML models

7.1. Physics-informed ML and its synergy with UQ and probabilistic ML

Physics-informed ML, and more broadly methods of scientific ML, has been developed to alleviate the challenge of training data scarcity and to improve the predictive capability of ML models by combining *physics*-based and *data*-driven modeling. Such a hybrid strategy is especially valuable for domains where training data is difficult or expensive to obtain, and where the modeling and downstream decision-making consequences are high (e.g., pertaining to health, safety, and security). In essence, physics-informed ML develops techniques to enable a seamless combination of physics-based models and observation data, or the embedding of physical and domain knowledge into data-driven ML models. Prior work on physics-informed ML can be broadly grouped into seven classes [14]: (1) impose physical knowledge as soft constraints in the loss functions of an ML model such as neural networks, for example the works of physics-informed neural networks (PINNs) [229–231]; (2) combine first-principle simulation data with experimental data to construct an augmented training dataset [232, 233]; (3) train an ML model with first-principle simulation data, then fine-tune the trained ML model with experimental data [96, 234], which is often referred to as transfer learning; (4) build an ML model in parallel with a physics-based model, and using the ML model to learn missing/unmodeled physics from experimental data [235, 236]; (5) use ML models to enhance physics-based models such as in delta or residual learning [237–239] and reduced-order modeling for building models with lower complexity and degrees of freedom for rapid and reliable model evaluations [240–242]; (6) use neural networks to predict the input or parameters of a physics-based model [243–246]; and (7) enforcing physical models in the architecture design of neural networks,

70

such as architectures dedicated to specific physics and engineering problems [247, 248] and utilizing a large amount of simulation data to emulate the dynamics of physical systems, such as deep operator networks [249] and Fourier neural operator [250]. A more detailed summary of these seven physics-informed ML categories can be found in Part 1 of our recent review on digital twins [14]. As mentioned in this review, the above list of seven categories is not exhaustive by any means, and many other approaches for combining data and physics have been developed over the past decade. Comprehensive reviews dedicated to physics-informed ML are also available in Refs. [77, 251].

Regardless of the specific means of incorporating physical knowledge into ML modeling, parameter and model-form uncertainty inevitably persist due to the imperfect knowledge of physics, and assumptions and approximations made to simplify the problem setup during the modeling process. In the case of uncertainty of physical parameters (e.g., uncertain parameters in a PDE), the corresponding probability distribution of solution variables can be generated with those parameters as inputs to neural network representations of the solution field [252] or utilizing generative adversarial networks [253]. However, these approaches do not consider the uncertainty induced by the use of physics-informed ML model itself (e.g., uncertainty due to the use of a neural network). For neural networks, the commonly used MC dropout helps increase robustness of training associated with randomization of the network architecture, while BNNs more directly seek to quantify the parameter uncertainty of the neural network (e.g., for its weight and bias terms). Moreover, physics-constrained BNNs [82, 254] have been developed to address the uncertainty in PINNs. We direct interested readers to two recent review papers for a more comprehensive, in-depth discussion on UQ for physics-informed ML [51, 255], with emphasis on PINNs [51, 255] and deep operator networks [51].

7.2. Probabilistic Learning on Manifolds (PLoM)

Another ML approach that naturally captures the uncertainty of data while simultaneously performing dimension reduction is the Probabilistic Learning on Manifolds (PLoM) [256]. PLoM builds a generative model from an initial set of data samples by identifying a manifold where the unknown probability measure concentrates. The learning procedure starts by scaling the training data via principal component analysis (PCA) followed by performing a density estimation (e.g., Gaussian kernel density estimation) on the training data in the PCA space. Then, an Itô stochastic differential equation is established as a sample-generating mechanism whose invariant distribution matches the probability density just estimated. In order to ensure the generated samples coalesce around a low-dimensional manifold, additional structure is injected by forming a reduced-order "diffusion-maps basis" induced by an isotropic diffusion kernel to help constrain the sample coordinates. Putting everything together, new samples consistent with the training data distribution can be generated on a low-dimensional manifold by numerically solving the Itô equations through a discretization scheme.

With its ability to find low-dimensional manifolds, PLoM is particularly suitable for dimension reduction of high-dimensional datasets. Its strength and focus thus differs from ML constructs such as GPR and BNN that are more directly concerned with function approximation and regression

tasks. To be effective, PLoM generally requires a sufficiently large quantity of data samples that can reasonably reveal the underlying distribution geometry. This also differs from GPR and BNN that, by design, engender a larger degree of uncertainty in the model (e.g., by falling back towards their prior uncertainty) when less data is available. Nonetheless, PLoM has been demonstrated to work well even in settings with relatively small datasets, especially if additional constraining from relevant governing PDEs is available [257]. Lastly, the generative model resulting from PLoM can be highly versatile and used for a range of applications beyond sample generation and surrogate modeling, such as density estimates of statistics of interest [258], optimization under uncertainty [259], and design using digital twins [260], as some examples.

7.3. Interpretability of ML models for dynamic systems

Data-driven system identification plays a vital role in structural health monitoring, system failure prognostics, design and control as well as risk assessment of dynamic systems. In the past decades, various approaches have been developed to accomplish this task. Some representative examples include autoregressive models, autoregressive–moving-average models, nonlinear autoregressive moving average with exogenous inputs models, the Volterra series gray-box tooling method, and ML-based methods emerging in recent years [78, 261]. While these black-box or grey-box models show promising performance in various applications, they are often criticized for their lack of interpretability.

As introduced earlier, significant efforts have been devoted to addressing the challenge of interpretability in ML models. Among the techniques that stand out are SHAP, Grad-CAM, and other methods. Notably, over the past decade, there has been a remarkable stride in enhancing the interpretability of ML models through the integration of data, genetic programming, and sparsity. This fusion has led to the formulation of evolution equations that are not only simplistic but also parsimonious. Several approaches have been proposed to construct interpretable ML models, particularly symbolic regression, which has been applied with different techniques [262]. A pivotal advancement in this realm is the emergence of the Sparse Identification of Nonlinear Dynamics (SINDy) technique, which has become a cornerstone in addressing this issue. Initially proposed by Brunton et al. [263], SINDy aims to uncover the underlying partial differential equations governing nonlinear dynamic systems. This discovery is accomplished even in the presence of noisy measurement data [264, 265].

What sets SINDy apart is its ability to exploit the dominance of only a handful of terms in shaping the behavior of nonlinear dynamic systems. This is achieved by encouraging sparsity in the data-driven identification of governing equations, leveraging an extensive library of potential function bases. From the model interpretation point of view, the sparsity promoting the discovery of governing equations of dynamic systems results in parsimonious and interpretable models that strike a sound balance between regression accuracy with model complexity. In particular, the parsimonious model is achieved by employing sparsity-promoting regularization techniques [263, 266], such as LASSO regression, also known as L1 regularization, using sparsifying priors, hard thresholding with Pareto analysis. The resulting parsimonious representations through sparsity lead to interpretable

models with good generalization to unseen data. Besides, the sparsity in the resulting function basis offers valuable insights into the management of model selection uncertainty in the context of hybrid dynamical systems [267]. For instance, hybrid SINDy employed the Akaike information criterion score on out-of-sample validation data to match the SINDy model with a specific regime in a hybrid dynamical system, from which the switching point of the hybrid system can be found [267].

The elegance and clarity inherent in the models derived through SINDy are of particular importance when considering ML model interpretability. Building upon the foundational work of Brunton and Kutz, a multitude of SINDy variants have emerged, finding applications even in UQ contexts. A remarkable instance worth highlighting is the approach introduced by Hirsh et al. [266], wherein the SINDy approach is extended into a Bayesian probabilistic framework. This novel approach, termed Uncertainty Quantification SINDy (UQ-SINDy), accounts for uncertainties in SINDy coefficients arising from observation errors and limited data. The central innovation lies in the integration of sparsifying priors, specifically the spike and slab prior and the regularized horseshoe prior, into the Bayesian inference of SINDy coefficients. By unifying UQ with SINDy variants, this approach not only heightens the interpretability of ML models but also facilitates the quantification of the prediction's confidence level.

7.4. PCE and its relationship with GPR and connection with ML

A key role for both GPR (see Sec. 1 and Appendix B) and polynomial chaos expansion (PCE) is building surrogate models for solving engineering design problems. The need for surrogate modeling stems from the multi-query nature of uncertainty propagation and design optimization, which often require many repeated simulation runs (e.g., $10^3 - 10^6$) to assess the behavior of output responses under different realizations of input design variables and simulation model parameters. This process may become prohibitively expensive for high-fidelity models where each simulation may require hours to days. One strategy to accelerate these computations, as explained in Appendix B.1, is to build a cheap-to-evaluate surrogate of the computationally expensive simulation model—i.e. to trade model fidelity for speed. The surrogate model, sometimes called metamodel or response surface, is often an explicit mathematical function (e.g., as in GPR and PCE), allowing for rapid predictions at different input realizations.

Having presented GPR in detail in Sec. 1 and Appendix B, we briefly introduce PCE here. PCE was originally proposed in the 1930s to model stochastic processes using a spectral expansion of multivariate Hermite polynomials of Gaussian random variables [268]. These Hermite polynomial basis functions are orthogonal with respect to the joint probability distribution of the respective Gaussian variables. PCE was later applied to solve physics and engineering problems [269] and extended to non-Gaussian probability distributions, giving rise to the *generalized PCE* [270]. Since the input variables of a PCE are naturally formulated to follow certain probability distributions, PCE has been a convenient and popular tool for conducting UQ. However, PCE has not been employed much for UQ of ML models, since most ML models are already relatively inexpensive to evaluate; rather, PCE brings more value for enabling UQ of expensive computer simulation models. In that case, a PCE surrogate model is built to approximate the original simulation model,

where the PCE's expansion coefficients can be computed, for example, non-intrusively by projection (numerical integration via quadrature or simulation) [271] or regression (least squares minimization of the fitting error) [272].

One major challenge faced by PCE is the curse of dimensionality, where the number of model parameters (and in turn training samples of the simulation model) increases exponentially with the input dimension (i.e., the number of input random variables). Several algorithmic techniques have been developed to alleviate this issue through truncation schemes that can identify a sparse set of important polynomials to be included. Two notable methods for introducing sparsity are the Smolyak sparse constructions (and their adaptive versions) [273–275], and variants of compressive sensing (such as least angle regression and LASSO) [276–278]. Such effort has been made in the context of surrogate modeling [276–279] and reliability analysis [280–283]. A comprehensive review of sparse PCE is provided in Ref. [284].

Historically, PCE and GPR (or kriging) have been studied separately and mostly in isolation, although both methods have produced many success stories in surrogate modeling. Recently, attempts have been made to combine PCE and kriging, resulting in PCE-kriging hybrids [285]. The basic idea is to use PCE to represent the mean function $m(\mathbf{x})$ of the Gaussian process prior (see Eq. (8)) that captures the global trend of the computer simulation model (i.e., $f(\mathbf{x})$). The GPR formulation with a non-zero, non-constant mean function is called universal kriging, which differs from ordinary kriging where the mean function is set as a constant (e.g., zero). When combined with kriging in this manner, PCE serves the purpose of a *deterministic* (non-probabilistic) mean (trend) function. Such PCE-kriging hybrids have found applications to uncertainty propagation in computational dosimetry [286] and damage quantification in structural health monitoring [287]. More broadly, while PCE is typically not used for UQ of ML models, it may be combined with other ML techniques (e.g., kriging [285] and radial basis functions [288]) to produce hybrid PCE-ML models with improved prediction accuracy over standalone PCE surrogates. On a final note, although PCE is typically not categorized as an ML technique, it was reported to offer surrogate modeling accuracy on par with state-of-the-art ML techniques such as regression tree, neural network, and support vector machine [289].

8. Conclusion and outlook

This tutorial aims to cover the fundamental role of UQ in ML, particularly focusing on a detailed introduction of state-of-the-art UQ methods for neural networks and a brief review of applications in engineering design and PHM. It possesses four salient characteristics: (1) classification of uncertainty types (aleatory vs. epistemic), sources, and causes pertaining to ML models; (2) tutorial-style descriptions of emerging UQ techniques; (3) quantitative metrics for evaluation and calibration of predictive uncertainty; and (4) easily accessible source codes for implementing and comparing several state-of-the-art UQ techniques in engineering design and PHM applications. Two case studies are developed to demonstrate the implementation of UQ methods and benchmark their performance in predicting battery life using early-life data (case study 1) and turbofan engine RUL using online-accessible measurements (case study 2). Our rigorous examination of the state-of-the-art techniques

74

for UQ, calibration, and evaluation and the two case studies offers a holistic lens on pressing issues that need to be tackled along the future development of UQ techniques in terms of scalability, principleness, and decomposition given the increasing importance of UQ in safeguarding the usage of ML models in high-stakes applications.

It is important to note that the case studies presented in this paper are not optimized in terms of their hyperparameters, and it is reasonable to expect that optimizing them would yield even better performance results. The primary objective of this paper is to offer a user-friendly platform for individuals seeking to comprehend the analyzed methods and to encourage them to enhance and suggest new ones.

Essentially, UQ acts as a layer of safety assurance on top of ML models, enabling rigorous and quantitative risk assessment and management of ML solutions in high-stakes applications. As UQ methods for ML models continue to mature, they are anticipated to play a crucial role in creating safe, reliable, and trustworthy ML solutions by safeguarding against various risks such as OOD, adversarial attacks, and spurious correlations. From this perspective, the development of UQ methods is of paramount significance in expanding the adoption of ML models in breadth and depth. The accurate, sound, and principled quantification of uncertainty in ML model prediction has great potential to fundamentally tackle the safety assurance problem that haunts ML's development. Towards this end, several long-standing challenges encompassing the UQ development need to be addressed by the research community:

1. The need for a unified and well-acknowledged testbed to comprehensively examine the performance of the diverse and expanding set of UQ methods in uncertainty quantification, calibration (and recalibration), decomposition, attribution, and interpretation. Although some recent efforts were devoted to developing standardized benchmarks for UQ [290], most of these efforts primarily emphasized conventional performance metrics, such as prediction accuracy metrics and UQ calibration errors. However, other key performance aspects (e.g., uncertainty decomposition and uncertainty attribution) essential to ensuring high quality UQ have rarely been investigated. The lack of these key elements emerges as a significant challenge to the sound development of the UQ ecosystem. Hence, there is an imperative demand calling for establishing UQ testbeds with community-acknowledged standards to facilitate comprehensive testing and verification of the behavior of uncertainty generated by different UQ methods, especially on edge cases. Establishing such testbeds with the support of synthetic data generation is expected to tremendously benefit the long-term and sustainable development of UQ methods for ML models.

2. The need for principled, scalable, and computationally efficient UQ methods to enable high quality and large-scale UQ. As summarized in Table 2, each method covered in this tutorial has its own strengths and shortcomings. Although numerous efforts have been made to elevate the soundness and principleness of UQ methods of ML models, the existing methods still suffer from a common but critical deficiency: a lack of (limited) theoretical guarantee in detecting OOD instances. It is thus imperative to investigate further along this direction to fill the

75

loophole. Emerging deterministic methods such as SNGP exhibit a strong OOD detection capability due to distance awareness. In addition, the computational efficiency of UQ methods needs to be further improved to satisfy the need for real-time or near real-time decision making in a broad range of safety-critical applications (e.g., autonomous driving and aviation). Thus, more research efforts need to be invested in enabling three key essential features of high quality UQ: principleness, scalability, and efficiency.

3. ML models have shown promising potential in addressing long-standing engineering design problems in recent years. Especially for GPR, its applications in engineering design have resulted in a family of adaptive surrogate modeling methods for reliability-based design optimization, robust design, and design optimization in general. These ML-based design methods have revolutionized engineering design in various applications, including but not limited to design and discovery of new materials, design for additive manufacturing, and topology optimization. Despite these revolutionary advances, extending these methods to larger-scale and more complicated problems becomes increasingly urgent. To this end, various DNN-based methods have been investigated in engineering design to overcome the limitations of classical ML methods, such as the GPR-based approaches. Even though the emerging DNN-based methods show promise in addressing computational challenges in high-dimensional engineering design problems, their potential as efficient surrogates or accelerated optimizers has not yet been fully realized. The UQ methods for ML models presented in this paper will play a key role in fully releasing the power of DNNs in engineering design by enabling adaptive DNNs in the context of active learning to reduce the required quantity of training data without sacrificing the accuracy in surrogate modeling, reliability analysis, and optimization, (2) accelerated design optimization for large-scale systems, and (3) efficient and accurate UQ in engineering design accounting for various sources of aleatory and epistemic uncertainty (e.g., input-dependent aleatory uncertainty).

4. The PHM community has long recognized the importance of estimating the predictive uncertainty of prognostic models. These prognostic models can be built based on supervised ML or more traditional state-space models (see, for example, the Bayes filter in one of the earliest studies on battery prognostics [38]). As discussed in Sec. 5.3, in the PHM field, UQ of ML models has been predominantly applied to the task of predicting the RUL of a system or component. The focus of UQ in this context is to provide a probability distribution of the RUL rather than a single point estimate. While UQ in the PHM field has primarily been focused on RUL prediction, there is a growing interest in applying UQ to other tasks, such as anomaly detection, fault detection and classification, and health estimation. Many of the UQ methods discussed in detail in Sec. 3 can also be readily applied to these classification and regression tasks in the PHM field. Looking ahead, we identify three research directions along which positive and significant impacts could be made on the PHM field surrounding UQ of ML models. First, decomposing the total predictive uncertainty into its aleatoric and epistemic components is highly desirable and sometimes essential, as noted in Sec. 5.3. Such a decomposition has

several benefits, for example, highlighting the need for improved sensing solutions with lower measurement noise to reduce aleatory uncertainty and identifying areas where further data collection or model refinement efforts may be necessary to reduce epistemic uncertainty. More work is needed to develop UQ methods with built-in uncertainty decomposition capability and create procedures to assess the accuracy of uncertainty decomposition. Second, prognostic studies involving UQ mostly evaluate UQ quality subjectively and qualitatively by looking at whether a two-sided 95% confidence interval of the RUL estimate gets narrows with time and contains the true RUL, especially toward the end of life. As discussed in a general context in Sec. 4.4, we call for consistent effort among PHM researchers and practitioners to quantitatively evaluate their ML models' UQ quality using some of the metrics introduced in Sec. 4, such as calibration metrics (Sec. 4.1), sparsification metrics (Sec. 4.2), and NLL (Sec. 4.3). Ideally, UQ quality assessment should also become standard practice when building and deploying ML models in PHM applications, just as prediction accuracy assessment is currently standard practice. Third, both UQ and interpretation serve the purpose of improving model transparency and trustworthiness, as noted in Sec. 1. An under-explored question is whether UQ capability can help improve interpretability and vice versa. For example, interpretability can provide insights into the most important input features for making predictions. Such an understanding could allow distance-aware UQ models to define their distance measures based only on highly important features, potentially improving the UQ quality.

5. Model uncertainty quantification for label-free learning is another future research direction. Obtaining labels by solving implicit engineering physics models is usually costly. Label-free machine learning embeds physics models in a cost function or as constraints in the model training process without solving them. As a result, labels are not required. Physics-informed neural network (PINN) is one such label-free method [77, 231]. This method has gained much attention because it makes the regression task feasible without solving the true label. In addition, the physical constraints prevent the regression from severe overfitting in conventional neural networks, especially when data are limited. Since labels are not available, the quantification of prediction uncertainty of the machine learning model is extremely difficult. Even the prediction errors at the training points are unknown. Due to this reason, the GPR method has not been used for label-free learning since the prediction of a GPR model requires labels at the training points. A proof-of-concept study has been conducted for quantifying epistemic uncertainty for physics-based label-free regression [291]. This method integrates neural networks and GPR models and can produce both systematic error (represented by a mean) and random error (represented by a standard deviation) for a model prediction. The method, however, has not been extended to time- and space-dependent problems where partially different equations are involved. There is a need to develop generic uncertainty quantification methods for label-free learning.

Authors' contributions

All the authors read and approved the final manuscript. Hu, C. and Zhang, X. devised the original concept of the tutorial paper. Hu, Z., Hu, C., Du, X., Wang, Y., and Huan, X. were responsible for the classification of types and sources of uncertainty pertaining to ML models. Hu, C. and Tran, A. were responsible for GPR. Huan, X. was responsible for implementing BNN via the means of MCMC and variational inference. Zhang, X. and X. Huan were responsible for MC dropout. Zhang, X. and Hu, C. were responsible for neural network ensemble. Hu, C. was responsible for deterministic methods for UQ of neural networks. Zhang, X. and Nemani, V., were responsible for the toy example to compare the predictive uncertainty produced by different UQ methods. Zhang, X. and Hu, C. were responsible for the summary of the qualitative comparison of different UQ methods. Hu, C. and Nemani, V. were responsible for the evaluation of predictive uncertainty. Hu, Z., Zhang, X., Hu., C., and Tran, A. were responsible for the review on UQ of ML models in engineering design. Biggio, L., and Fink, O. were responsible for the review of UQ of ML models in prognostics. Nemani, V. and Hu, C. were responsible for case study 1 – battery early life prediction. Biggio L. and Fink O. were responsible for case study 2 – turbofan engine prognostics. Zhang, X., Hu, C., and Hu, Z. were responsible for the conclusion and outlook. All authors participated in manuscript writing, review, and editing. All correspondence should be addressed to Xiaoge Zhang (e-mail: xiaoge.zhang@polyu.edu.hk) and Chao Hu (e-mails: chao.hu@uconn.edu; huchaostu@gmail.com).

References

[1] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324, doi: http://dx.doi.org/10.1109/5.726791.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 248–255, doi: http://dx.doi.org/10.1109/CVPR.2009.5206848, 2009.

[3] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, A. Oliva, Learning deep features for scene recognition using places database, Advances in Neural Information Processing Systems 27.

[4] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft COCO: Common objects in context, in: European Conference on Computer Vision, Springer, 740–755, 2014.

[5] J. Blitzer, M. Dredze, F. Pereira, Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification, in: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, 440–447, 2007.

[6] X. Glorot, A. Bordes, Y. Bengio, Domain adaptation for large-scale sentiment classification: A deep learning approach, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), 513–520, 2011.

[7] Q. Li, C. Shen, L. Chen, Z. Zhu, Knowledge mapping-based adversarial domain adaptation: A novel fault diagnosis method with high generalizability under variable working conditions, Mechanical Systems and Signal Processing 147 (2021) 107095, doi: https://doi.org/10.1016/j.ymssp.2020.107095.

[8] S. M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, Advances in Neural Information Processing Systems 30.

[9] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, Grad-CAM: Visual explanations from deep networks via gradient-based localization, in: Proceedings of the IEEE International Conference on Computer Vision, 618–626, doi: http://dx.doi.org/10.1109/ICCV.2017.74, 2017.

[10] C. Molnar, Interpretable machine learning, Lulu. com, 2020.

[11] J. Jiménez-Luna, F. Grisoni, G. Schneider, Drug discovery with explainable artificial intelligence, Nature Machine Intelligence 2 (10) (2020) 573–584, doi: https://doi.org/10.1038/s42256-020-00236-4.

[12] S. Guo, H. Ding, Y. Li, H. Feng, X. Xiong, Z. Su, W. Feng, A hierarchical deep convolutional regression framework with sensor network fail-safe adaptation for acoustic-emission-based structural health monitoring, Mechanical Systems and Signal Processing 181 (2022) 109508, doi: https://doi.org/10.1016/j.ymssp.2022.109508.

[13] S. Khan, T. Yairi, A review on the application of deep learning in system health management, Mechanical Systems and Signal Processing 107 (2018) 241–265, doi: https://doi.org/10.1016/j.ymssp.2017.11.024.

[14] A. Thelen, X. Zhang, O. Fink, Y. Lu, S. Ghosh, B. D. Youn, M. D. Todd, S. Mahadevan, C. Hu, Z. Hu, A comprehensive review of digital twin—part 1: modeling and twinning enabling technologies, Structural and Multidisciplinary Optimization 65 (12) (2022) 1–55, doi: https://doi.org/10.1007/s00158-022-03425-4.

[15] E. Begoli, T. Bhattacharya, D. Kusnezov, The need for uncertainty quantification in machine-assisted medical decision making, Nature Machine Intelligence 1 (1) (2019) 20–23, doi: https://doi.org/10.1038/s42256-018-0004-1.

[16] C. Rudin, Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, Nature Machine Intelligence 1 (5) (2019) 206–215, doi: https://doi.org/10.1038/s42256-019-0048-x.

[17] M. Sensoy, L. Kaplan, M. Kandemir, Evidential deep learning to quantify classification uncertainty, Advances in Neural Information Processing Systems 31, doi: https://doi.org/10.48550/arXiv.1806.01768.

[18] X. Zhang, S. Zhong, S. Mahadevan, Airport surface movement prediction and safety assessment with spatial–temporal graph convolutional neural network, Transportation Research Part C: Emerging Technologies 144 (2022) 103873, doi: https://doi.org/10.1038/s42256-019-0048-x.

[19] E. Hüllermeier, W. Waegeman, Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods, Machine Learning 110 (3) (2021) 457–506, doi: https://doi.org/10.1007/s10994-021-05946-3.

[20] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks doi: https://doi.org/10.48550/arXiv.1312.6199.

[21] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: Representing model uncertainty in deep learning, in: International Conference on Machine Learning, PMLR, 1050–1059, doi: https://doi.org/10.48550/arXiv.1506.02142, 2016.

[22] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, Advances in Neural Information Processing Systems 30.

[23] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision?, Advances in Neural Information Processing Systems 30, doi: https://doi.org/10.48550/arXiv.1703.04977.

[24] R. Jin, W. Chen, T. W. Simpson, Comparative studies of metamodelling techniques under multiple modelling criteria, Structural and Multidisciplinary Optimization 23 (1) (2001) 1–13, doi: https://doi.org/10.1007/s00158-001-0160-4.

[25] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, P. K. Tucker, Surrogate-based analysis and optimization, Progress in Aerospace Sciences 41 (1) (2005) 1–28, doi: https://doi.org/10.1016/j.paerosci.2005.02.001.

[26] G. G. Wang, S. Shan, Review of metamodeling techniques in support of engineering design optimization, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 4255, 415–426, doi: https://doi.org/10.1115/1.2429697, 2006.

[27] R. Jin, W. Chen, A. Sudjianto, On sequential sampling for global metamodeling in engineering design, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 36223, 539–548, doi: https://doi.org/10.1115/DETC2002/DAC-34092, 2002.

[28] B. J. Bichon, M. S. Eldred, L. P. Swiler, S. Mahadevan, J. M. McFarland, Efficient global reliability analysis for nonlinear implicit performance functions, AIAA Journal 46 (10) (2008) 2459–2468, doi: http://dx.doi.org/10.2514/1.34321.

[29] B. Echard, N. Gayton, M. Lemaire, AK-MCS: an active learning reliability method combining Kriging and Monte Carlo simulation, Structural Safety 33 (2) (2011) 145–154, doi: https://doi.org/10.1016/j.strusafe.2011.01.002.

[30] D. R. Jones, M. Schonlau, W. J. Welch, Efficient global optimization of expensive black-box functions, Journal of Global Optimization 13 (4) (1998) 455–492, doi: https://doi.org/10.1023/A:1008306431147.

[31] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, N. de Freitas, Taking the human out of the loop: A review of Bayesian optimization, Proceedings of the IEEE 104 (1) (2016) 148–175, doi: https://doi.org/10.1109/JPROC.2015.2494218.

[32] S. H. Lee, W. Chen, A comparative study of uncertainty propagation methods for black-box-type problems, Structural and Multidisciplinary Optimization 37 (3) (2009) 239–253, doi: https://doi.org/10.1007/s00158-008-0234-7.

[33] S. Chakraborty, Simulation free reliability analysis: A physics-informed deep learning based approach doi: https://doi.org/10.48550/arXiv.2005.01302.

[34] M. Li, Z. Wang, Deep learning for high-dimensional reliability analysis, Mechanical Systems and Signal Processing 139 (2020) 106399, doi: https://doi.org/10.1016/j.ymssp.2019.106399.

[35] C. Zhang, A. Shafieezadeh, Simulation-free reliability analysis with active learning and Physics-Informed Neural Network, Reliability Engineering & System Safety 226 (2022) 108716, doi: https://doi.org/10.1016/j.ress.2022.108716.

[36] J. B. Coble, J. W. Hines, Prognostic algorithm categorization with PHM challenge application, in: 2008 International Conference on Prognostics and Health Management, IEEE, 1–11, doi: https://doi.org/10.1109/PHM.2008.4711456, 2008.

[37] M. E. Tipping, Sparse Bayesian learning and the relevance vector machine, Journal of Machine Learning Research 1 (Jun) (2001) 211–244, doi: https://doi.org/10.1162/15324430152748236.

[38] B. Saha, K. Goebel, S. Poll, J. Christophersen, Prognostics methods for battery health monitoring using a Bayesian framework, IEEE Transactions on Instrumentation and Measurement 58 (2) (2008) 291–296, doi: https://doi.org/10.1109/TIM.2008.2005965.

[39] D. Wang, Q. Miao, M. Pecht, Prognostics of lithium-ion batteries based on relevance vectors and a conditional three-parameter capacity degradation model, Journal of Power Sources 239 (2013) 253–264, doi: https://doi.org/10.1016/j.jpowsour.2013.03.129.

[40] Y. Chang, J. Zou, S. Fan, C. Peng, H. Fang, Remaining useful life prediction of degraded system with the capability of uncertainty management, Mechanical Systems and Signal Processing 177 (2022) 109166, doi: https://doi.org/10.1016/j.ymssp.2022.109166.

[41] P. Wang, B. D. Youn, C. Hu, A generic probabilistic framework for structural health prognostics and uncertainty management, Mechanical Systems and Signal Processing 28 (2012) 622–637, doi: https://doi.org/10.1016/j.ymssp.2011.10.019.

[42] C. Hu, B. D. Youn, P. Wang, J. T. Yoon, Ensemble of data-driven prognostic algorithms for robust prediction of remaining useful life, Reliability Engineering & System Safety 103 (2012) 120–135, doi: https://doi.org/10.1016/j.ress.2012.03.008.

[43] D. Liu, J. Pang, J. Zhou, Y. Peng, M. Pecht, Prognostics for state of health estimation of lithium-ion batteries based on combination Gaussian process functional regression, Microelectronics Reliability 53 (6) (2013) 832–839, doi: https://doi.org/10.1016/j.microrel.2013.03.010.

[44] R. R. Richardson, M. A. Osborne, D. A. Howey, Gaussian process regression for forecasting battery state of health, Journal of Power Sources 357 (2017) 209–219, doi: https://doi.org/10.1016/j.jpowsour.2017.05.004.

[45] A. Thelen, M. Li, C. Hu, E. Bekyarova, S. Kalinin, M. Sanghadasa, Augmented model-based framework for battery remaining useful life prediction, Applied Energy 324 (2022) 119624, doi: https://doi.org/10.1016/j.apenergy.2022.119624.

[46] S. Sankararaman, Significance, interpretation, and quantification of uncertainty in prognostics and remaining useful life prediction, Mechanical Systems and Signal Processing 52 (2015) 228–247, doi: https://doi.org/10.1016/j.ymssp.2014.05.029.

[47] S. Sankararaman, K. Goebel, Uncertainty in prognostics and systems health management, International Journal of Prognostics and Health Management 6 (4), doi: https://doi.org/10.36001/ijphm.2015.v6i4.2319.

[48] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, et al., A review of uncertainty quantification in deep learning: Techniques, applications and challenges, Information Fusion 76 (2021) 243–297, doi: https://doi.org/10.1016/j.inffus.2021.05.008.

[49] U. Bhatt, J. Antorán, Y. Zhang, Q. V. Liao, P. Sattigeri, R. Fogliato, G. Melançon, R. Krishnan, J. Stanley, O. Tickoo, et al., Uncertainty as a form of transparency: Measuring, communicating, and using uncertainty, in: Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society, 401–413, doi: https://doi.org/10.48550/arXiv.2011.07586, 2021.

[50] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, et al., A survey of uncertainty in deep neural networks doi: https://doi.org/10.48550/arXiv.2107.03342.

[51] A. F. Psaros, X. Meng, Z. Zou, L. Guo, G. E. Karniadakis, Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons, Journal of Computational Physics 477 (2023) 111902, doi: https://doi.org/10.1016/j.jcp.2022.111902.

[52] L. Basora, A. Viens, M. A. Chao, X. Olive, A Benchmark on Uncertainty Quantification for Deep Learning Prognostics, arXiv preprint arXiv:2302.04730 .

[53] W. Zhang, Z. Yang, H. Jiang, S. Nigam, S. Yamakawa, T. Furuhata, K. Shimada, L. B. Kara, 3D shape synthesis for conceptual design and optimization using variational autoencoders, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 59186, American Society of Mechanical Engineers, V02AT03A017, doi: https://doi.org/10.1115/DETC2019-98525, 2019.

[54] W. Chen, M. Fuge, BézierGAN: Automatic Generation of Smooth Curves from Interpretable Low-Dimensional Parameters doi: https://doi.org/10.48550/arXiv.1808.08871.

[55] W. Chen, M. Fuge, Synthesizing designs with interpart dependencies using hierarchical generative adversarial networks, Journal of Mechanical Design 141 (11) (2019) 111403, doi: https://doi.org/10.1115/1.4044076.

[56] M. He, D. He, Deep learning based approach for bearing fault diagnosis, IEEE Transactions on Industry Applications 53 (3) (2017) 3057–3065, doi: https://doi.org/10.1109/TIA.2017.2661250.

[57] D.-T. Hoang, H.-J. Kang, A survey on deep learning based bearing fault diagnosis, Neurocomputing 335 (2019) 327–335, doi: https://doi.org/10.1016/j.neucom.2018.06.078.

[58] H. Lu, V. P. Nemani, V. Barzegar, C. Allen, C. Hu, S. Laflamme, S. Sarkar, A. T. Zimmerman, A physics-informed feature weighting method for bearing fault diagnostics, Mechanical Systems and Signal Processing 191 (2023) 110171, doi: https://doi.org/10.1016/j.ymssp.2023.110171.

[59] B. Hou, D. Wang, Y. Chen, H. Wang, Z. Peng, K.-L. Tsui, Interpretable online updated weights: Optimized square envelope spectrum for machine condition monitoring and fault diagnosis, Mechanical Systems and Signal Processing 169 (2022) 108779, doi: https://doi.org/10.1016/j.ymssp.2021.108779.

[60] V. Sinitsin, O. Ibryaeva, V. Sakovskaya, V. Eremeeva, Intelligent bearing fault diagnosis method combining mixed input and hybrid CNN-MLP model, Mechanical Systems and Signal Processing 180 (2022) 109454, doi: https://doi.org/10.1016/j.ymssp.2022.109454.

[61] J. Deutsch, D. He, Using deep learning-based approach to predict remaining useful life of rotating components, IEEE Transactions on Systems, Man, and Cybernetics: Systems 48 (1) (2017) 11–20, doi: https://doi.org/10.1109/TSMC.2017.2697842.

[62] W. Yu, I. Y. Kim, C. Mechefske, Remaining useful life estimation using a bidirectional recurrent neural network based autoencoder scheme, Mechanical Systems and Signal Processing 129 (2019) 764–780, doi: https://doi.org/10.1016/j.ymssp.2019.05.005.

[63] X. Li, W. Zhang, Q. Ding, Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction, Reliability Engineering & System Safety 182 (2019) 208–218, doi: https://doi.org/10.1016/j.ress.2018.11.011.

[64] A. Der Kiureghian, O. Ditlevsen, Aleatory or epistemic? Does it matter?, Structural Safety 31 (2) (2009) 105–112, doi: https://doi.org/10.1016/j.strusafe.2008.06.020.

[65] Y. Gal, J. Hron, A. Kendall, Concrete dropout, Advances in Neural Information Processing Systems 30.

[66] R. Sanjay, R. Sriram, Data Fidelity and Latency: All things Clinical, Innovaccer 1 (2022) https://innovaccer.com/resources/blogs/data--fidelity--and--latency--all--things--clinical.

[67] A. Saltelli, S. Tarantola, F. Campolongo, M. Ratto, et al., Sensitivity analysis in practice: a guide to assessing scientific models, Chichester, England Doi: https://doi.org/10.1111/j.1467-985X.2005.358_16.x.

[68] I. M. Sobol', On sensitivity estimation for nonlinear mathematical models, Matematicheskoe Modelirovanie 2 (1) (1990) 112–118.

[69] I. M. Sobol, Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates, Mathematics and Computers in Simulation 55 (1-3) (2001) 271–280, doi: https://doi.org/10.1016/S0378-4754(00)00270-6.

[70] Y. Gal, Uncertainty in deep learning, Ph.D. thesis, PhD thesis, University of Cambridge, 2016.

[71] S. Depeweg, J.-M. Hernandez-Lobato, F. Doshi-Velez, S. Udluft, Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning, in: International Conference on Machine Learning, PMLR, 1184–1193, 2018.

[72] L. Smith, Y. Gal, Understanding measures of uncertainty for adversarial example detection doi: https://doi.org/10.48550/arXiv.1803.08533.

[73] A. Malinin, M. Gales, Predictive uncertainty estimation via prior networks, Advances in Neural Information Processing Systems 31, doi: https://doi.org/10.48550/arXiv.1802.10501.

[74] K. P. Murphy, Probabilistic machine learning: an introduction, MIT press, 2022.

[75] A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, S. Tarantola, Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index, Computer Physics Communications 181 (2) (2010) 259–270, doi: https://doi.org/10.1016/j.cpc.2009.09.018.

[76] C. Shorten, T. M. Khoshgoftaar, A survey on image data augmentation for deep learning, Journal of Big Data 6 (1) (2019) 1–48, doi: https://doi.org/10.1186/s40537-019-0197-0.

[77] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nature Reviews Physics 3 (6) (2021) 422–440.

[78] A. Thelen, X. Zhang, O. Fink, Y. Lu, S. Ghosh, B. D. Youn, M. D. Todd, S. Mahadevan, C. Hu, Z. Hu, A Comprehensive Review of Digital Twin–Part 2: Roles of Uncertainty Quantification and Optimization, a Battery Digital Twin, and Perspectives, Structural and Multidisciplinary Optimization 66 (1) (2023) 1–43, doi: https://doi.org/10.1007/s00158-022-03410-x.

[79] Y. Xu, S. Kohtz, J. Boakye, P. Gardoni, P. Wang, Physics-informed machine learning for reliability and systems safety applications: State of the art and challenges, Reliability Engineering & System Safety 230 (2022) 108900, doi: https://doi.org/10.1016/j.ress.2022.108900.

[80] C. Hu, K. Goebel, D. Howey, Z. Peng, D. Wang, P. Wang, B. D. Youn, Special issue on Physics-informed machine learning enabling fault feature extraction and robust failure prognosis, Mechanical Systems and Signal Processing 192 (2023) 110219, doi: https://doi.org/10.1016/j.ymssp.2023.110219.

[81] P. Wang, D. Coit, Physics-Informed Machine Learning for Reliability and Safety, URL https://www.sciencedirect.com/journal/reliability-engineering-and-system-safety/special-issue/1084PD0CV5B, 2023 (Accessed on 2023-04-18).

[82] L. Malashkhia, D. Liu, Y. Lu, Y. Wang, Physics-Constrained Bayesian Neural Network for Bias and Variance Reduction, Journal of Computing and Information Science in Engineering 23 (1) (2023) 011012, doi: https://doi.org/10.1115/1.4055924.

[83] Y. Deng, Multifidelity Data Fusion via Gradient-Enhanced Gaussian Process Regression, Communications in Computational Physics 28 (5) (2020) 1812–1837, doi: https://doi.org/10.4208/cicp.OA-2020-0151.

[84] M. Plumlee, V. R. Joseph, Orthogonal Gaussian process models, Statistica Sinica (2018) 601–619Doi: https://doi.org/10.5705/ss.202015.0404.

[85] A. Tran, K. Maupin, T. Rodgers, Monotonic Gaussian process for physics-constrained machine learning with materials science applications, Journal of Computing and Information Science in Engineering 23 (1) (2023) 011011, doi: https://doi.org/10.1115/1.4055852.

[86] M. Raghu, C. Zhang, J. Kleinberg, S. Bengio, Transfusion: Understanding transfer learning for medical imaging, Advances in Neural Information Processing Systems 32.

[87] L. Bottou, Stochastic gradient descent tricks, in: Neural networks: Tricks of the trade, Springer, 421–436, doi: https://doi.org/10.1007/978-3-642-35289-8, 2012.

[88] D. Liu, Y. Wang, A Dual-Dimer method for training physics-constrained neural networks with minimax architecture, Neural Networks 136 (2021) 112–125, doi: https://doi.org/10.1016/j.neunet.2020.12.028.

[89] J. Cai, J. Luo, S. Wang, S. Yang, Feature selection in machine learning: A new perspective, Neurocomputing 300 (2018) 70–79, doi: https://doi.org/10.1016/j.neucom.2017.11.077.

[90] G. Chandrashekar, F. Sahin, A survey on feature selection methods, Computers & Electrical Engineering 40 (1) (2014) 16–28, doi: https://doi.org/10.1016/j.compeleceng.2013.11.024.

[91] G. Box, All models are wrong, but some are useful, Robustness in Statistics 202 (1979) (1979) 549, doi: https://doi.org/10.1007/s10815-020-01895-3.

[92] A. Tran, J. Tranchida, T. Wildey, A. P. Thompson, Multi-fidelity machine-learning with uncertainty quantification and Bayesian optimization for materials design: Application to ternary random alloys, The Journal of Chemical Physics 153 (7) (2020) 074705, doi: https://doi.org/10.1063/5.0015672.

[93] G. Pilania, J. E. Gubernatis, T. Lookman, Multi-fidelity machine learning models for accurate bandgap predictions of solids, Computational Materials Science 129 (2017) 156–163, doi: https://doi.org/10.1016/j.commatsci.2016.12.004.

[94] D. Liu, Y. Wang, Multi-fidelity physics-constrained neural network and its application in materials modeling, Journal of Mechanical Design 141 (12) (2019) 121403, doi: https://doi.org/10.1115/1.4044400.

[95] D. Liu, P. Pusarla, Y. Wang, Multi-Fidelity Physics-Constrained Neural Networks with Minimax Architecture, Journal of Computing and Information Science in Engineering 23 (3) (2023) 031008, doi: https://doi.org/10.1115/1.4055316.

[96] X. Huang, T. Xie, Z. Wang, L. Chen, Q. Zhou, Z. Hu, A transfer learning-based multi-fidelity point-cloud neural network approach for melt pool modeling in additive manufacturing, ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part B: Mechanical Engineering 8 (1), doi: https://doi.org/10.1115/1.4051749.

[97] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, doi: https://doi.org/10.1038/nature14539.

[98] X. Zhang, F. T. Chan, C. Yan, I. Bose, Towards risk-aware artificial intelligence and machine learning systems: An overview, Decision Support Systems 159 (2022) 113800, doi: https://doi.org/10.1016/j.dss.2022.113800.

[99] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770–778, doi: https://doi.org/10.1109/CVPR.2016.90, 2016.

[100] X. Zhang, S. Mahadevan, Bayesian neural networks for flight trajectory prediction and safety assessment, Decision Support Systems 131 (2020) 113246, doi: https://doi.org/10.1016/j.dss.2020.113246.

[101] X. Zhang, F. T. Chan, S. Mahadevan, Explainable machine learning in image classification models: An uncertainty quantification perspective, Knowledge-Based Systems 243 (2022) 108418, doi: https://doi.org/10.1016/j.knosys.2022.108418.

[102] S. Cheng, Y. Yang, M. J. Brear, M. Frenklach, Quantifying uncertainty in kinetic simulation of engine autoignition, Combustion and Flame 216 (2020) 174–184, doi: https://doi.org/10.1016/j.combustflame.2020.02.025.

[103] G. Mårtensson, D. Ferreira, T. Granberg, L. Cavallin, K. Oppedal, A. Padovani, I. Rektorova, L. Bonanni, M. Pardini, M. G. Kramberger, et al., The reliability of a deep learning model in clinical out-of-distribution MRI data: a multicohort study, Medical Image Analysis 66 (2020) 101714, doi: https://doi.org/10.1016/j.media.2020.101714.

[104] N. Tagasovska, D. Lopez-Paz, Single-model uncertainties for deep learning, Advances in Neural Information Processing Systems 32.

[105] K. Osawa, S. Swaroop, M. E. E. Khan, A. Jain, R. Eschenhagen, R. E. Turner, R. Yokota, Practical deep learning with Bayesian principles, Advances in Neural Information Processing Systems 32.

[106] C. E. Rasmussen, Gaussian processes in machine learning, MIT Press, doi: https://doi.org/10.1007/978-3-540-28650-9_4, 2006.

[107] E. Brochu, V. M. Cora, N. de Freitas, A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, arXiv preprint arXiv:1012.2599 Doi: https://doi.org/10.48550/arXiv.1012.2599.

[108] R. M. Neal, Bayesian learning for neural networks, vol. 118, Springer Science & Business Media, 2012.

[109] R. Furrer, M. G. Genton, D. Nychka, Covariance tapering for interpolation of large spatial datasets, Journal of Computational and Graphical Statistics 15 (3) (2006) 502–523, doi: https://doi.org/10.1198/106186006X132178.

[110] C. G. Kaufman, M. J. Schervish, D. W. Nychka, Covariance tapering for likelihood-based estimation in large spatial data sets, Journal of the American Statistical Association 103 (484) (2008) 1545–1555, doi: https://doi.org/10.1198/016214508000000959.

[111] N. Cressie, G. Johannesson, Fixed rank kriging for very large spatial data sets, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 70 (1) (2008) 209–226, doi: https://doi.org/10.1111/j.1467-9868.2007.00633.x.

[112] S. Banerjee, A. E. Gelfand, A. O. Finley, H. Sang, Gaussian predictive process models for large spatial data sets, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 70 (4) (2008) 825–848, doi: https://doi.org/10.1111/j.1467-9868.2008.00663.x.

[113] R. M. Neal, Monte Carlo implementation of Gaussian process models for Bayesian regression and classification, arXiv preprint physics/9701026 .

[114] I. Andrianakis, P. G. Challenor, The effect of the nugget on Gaussian process emulators of computer models, Computational Statistics & Data Analysis 56 (12) (2012) 4215–4228, doi: https://doi.org/10.1016/j.csda.2012.04.020.

[115] L. Le Gratiet, C. Cannamela, B. Iooss, A Bayesian approach for global sensitivity analysis of (multifidelity) computer codes, SIAM/ASA Journal on Uncertainty Quantification 2 (1) (2014) 336–363, doi: https://doi.org/10.1137/130926869.

[116] M. Menz, S. Dubreuil, J. Morio, C. Gogu, N. Bartoli, M. Chiron, Variance based sensitivity analysis for Monte Carlo and importance sampling reliability assessment with Gaussian

processes, Structural Safety 93 (2021) 102116, doi: https://doi.org/10.1016/j.strusafe.2021.102116.

[117] P. Wei, Y. Zheng, J. Fu, Y. Xu, W. Gao, An expected integrated error reduction function for accelerating Bayesian active learning of failure probability, Reliability Engineering & System Safety 231 (2023) 108971, doi: https://doi.org/10.1016/j.ress.2022.108971.

[118] Q. V. Le, A. J. Smola, S. Canu, Heteroscedastic Gaussian process regression, in: Proceedings of the 22nd International Conference on Machine learning, 489–496, 2005.

[119] M. L. Stein, Interpolation of spatial data: some theory for kriging, Springer Science & Business Media, 1999.

[120] H. Liu, Y.-S. Ong, X. Shen, J. Cai, When Gaussian process meets big data: A review of scalable GPs, IEEE Transactions on Neural Networks and Learning Systems 31 (11) (2020) 4405–4423, doi: https://doi.org/10.1109/TNNLS.2019.2957109.

[121] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, N. De Feitas, Bayesian optimization in a billion dimensions via random embeddings, Journal of Artificial Intelligence Research 55 (2016) 361–387, doi: https://doi.org/10.1613/jair.4806.

[122] R. Tripathy, I. Bilionis, M. Gonzalez, Gaussian processes with built-in dimensionality reduction: Applications to high-dimensional uncertainty propagation, Journal of Computational Physics 321 (2016) 191–223, doi: https://doi.org/10.1016/j.jcp.2016.05.039.

[123] M. A. Bouhlel, N. Bartoli, A. Otsmane, J. Morlier, Improving kriging surrogates of high-dimensional design models by Partial Least Squares dimension reduction, Structural and Multidisciplinary Optimization 53 (2016) 935–952, doi: https://doi.org/10.1007/s00158-015-1395-9.

[124] N. Durrande, D. Ginsbourger, O. Roustant, Additive covariance kernels for high-dimensional Gaussian process modeling, in: Annales de la Faculté des sciences de Toulouse: Mathématiques, vol. 21, 481–499, 2012.

[125] M. Binois, N. Wycoff, A survey on high-dimensional Gaussian process modeling with application to Bayesian optimization, ACM Transactions on Evolutionary Learning and Optimization 2 (2) (2022) 1–26, doi: https://doi.org/10.1145/3545611.

[126] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536, doi: https://doi.org/10.1038/323533a0.

[127] H. Robbins, S. Monro, A Stochastic Approximation Method, The Annals of Mathematical Statistics 22 (3) (1951) 400–407, doi: https://doi.org/10.1214/aoms/1177729586.

[128] Y. A. LeCun, L. Bottou, G. B. Orr, K.-R. Müller, Efficient BackProp, in: G. Montavon, G. B. Orr, K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade, Springer-Verlag Berlin Heidelberg, 9–48, doi:https://doi.org/10.1007/978-3-642-35289-8_3, 2012.

[129] J. O. Berger, Statistical Decision Theory and Bayesian Analysis, Springer Series in Statistics, Springer New York, New York, NY, ISBN 978-1-4419-3074-3, doi: https://doi.org/10.1007/978-1-4757-4286-2, 1985.

[130] J. M. Bernardo, A. F. M. Smith, Bayesian Theory, John Wiley & Sons, New York, NY, 2000.

[131] D. S. Sivia, J. Skilling, Data Analysis: A Bayesian Tutorial, Oxford University Press, New York, NY, 2nd edn., 2006.

[132] D. J. C. MacKay, A Practical Bayesian Framework for Backpropagation Networks, Neural Computation 4 (3) (1992) 448–472, doi: https://doi.org/10.1162/neco.1992.4.3.448.

[133] A. Graves, Practical Variational Inference for Neural Networks, in: Advances in Neural Information Processing Systems 24 (NIPS 2011), Granada, Spain, 2348–2356, 2011.

[134] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight Uncertainty in Neural Networks, in: Proceedings of the 32nd International Conference on Machine Learning, vol. 37, 1613–1622, 2015.

[135] A. O'Hagan, C. E. Buck, A. Daneshkhah, J. R. Eiser, P. H. Garthwaite, D. J. Jenkinson, J. E. Oakley, T. Rakow, Uncertain Judgements: Eliciting Experts' Probabilities, John Wiley & Sons, Ltd, Chichester, UK, doi: https://doi.org/10.1002/0470033312, 2006.

[136] H. Jeffreys, An invariant form for the prior probability in estimation problems, Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences 186 (1007) (1946) 453–461, doi: https://doi.org/10.1098/rspa.1946.0056.

[137] E. T. Jaynes, Prior Probabilities, IEEE Transactions on Systems Science and Cybernetics 4 (3) (1968) 227–241, doi: https://doi.org/10.1109/TSSC.1968.300117.

[138] V. Fortuin, Priors in Bayesian Deep Learning: A Review, International Statistical Review 90 (3) (2022) 563–591, doi: https://doi.org/10.1111/insr.12502.

[139] C. Andrieu, N. de Freitas, A. Doucet, M. I. Jordan, An Introduction to MCMC for Machine Learning, Machine Learning 50 (2003) 5–43, doi: https://doi.org/10.1023/A:1020281327116.

[140] S. Brooks, A. Gelman, G. Jones, X.-L. Meng (Eds.), Handbook of Markov Chain Monte Carlo, Chapman & Hall/CRC, doi: https://doi.org/10.1201/b10905, 2011.

[141] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, Equation of State Calculations by Fast Computing Machines, The Journal of Chemical Physics 21 (6) (1953) 1087–1092, doi: https://doi.org/10.1063/1.1699114.

[142] W. K. Hastings, Monte Carlo sampling methods using Markov chains and their applications, Biometrika 57 (1) (1970) 97–109, doi: https://doi.org/10.1093/biomet/57.1.97.

[143] R. M. Neal, MCMC Using Hamiltonian Dynamics, in: Handbook of Markov Chain Monte Carlo, 113–162, doi: https://doi.org/10.1201/b10905-6, 2011.

[144] M. Betancourt, A Conceptual Introduction to Hamiltonian Monte Carlo, arXiv preprint arXiv:1701.02434 doi: https://doi.org/10.48550/arXiv.1701.02434.

[145] T. Chen, E. B. Fox, C. Guestrin, Stochastic Gradient Hamiltonian Monte Carlo, in: Proceedings of the 31st International Conference on Machine Learning, vol. 32, Beijing, 1683–1691, 2014.

[146] C. Zhang, B. Shahbaba, H. Zhao, Variational Hamiltonian Monte Carlo via Score Matching, Bayesian Analysis 13 (2) (2018) 485–506, doi: https://doi.org/10.1214/17-BA1060.

[147] D. M. Blei, A. Kucukelbir, J. D. McAuliffe, Variational Inference: A Review for Statisticians, Journal of the American Statistical Association 112 (518) (2017) 859–877, doi: https://doi.org/10.1080/01621459.2017.1285773.

[148] C. Zhang, J. Butepage, H. Kjellstrom, S. Mandt, Advances in Variational Inference, IEEE Transactions on Pattern Analysis and Machine Intelligence 41 (8) (2019) 2008–2026, doi: https://doi.org/10.1109/TPAMI.2018.2889774.

[149] D. J. Rezende, S. Mohamed, Variational inference with normalizing flows, in: 32nd International Conference on Machine Learning, ICML 2015, vol. 2, 1530–1538, 2015.

[150] Y. Marzouk, T. Moselhy, M. Parno, A. Spantini, Sampling via Measure Transport: An Introduction, in: Handbook of Uncertainty Quantification, Springer International Publishing, Cham, 1–41, doi: https://doi.org/10.1007/978-3-319-11259-6_23-1, 2016.

[151] Q. Liu, D. Wang, Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm, in: Advances in Neural Information Processing Systems 29 (NIPS 2016), Barcelona, Spain, 2378–2386, 2016.

[152] G. Detommaso, T. Cui, A. Spantini, Y. Marzouk, R. Scheichl, A Stein variational Newton method, in: Advances in Neural Information Processing Systems, 9169–9179, doi: https://doi.org/10.48550/arXiv.1806.03085, 2018.

[153] A. Leviyev, J. Chen, Y. Wang, O. Ghattas, A. Zimmerman, A stochastic Stein Variational Newton method, arXiv preprint arXiv:2204.09039 (2016) (2022) 1–17, doi: https://doi.org/10.48550/arXiv.2204.09039.

[154] P. Chen, O. Ghattas, Projected stein variational gradient descent, in: Advances in Neural Information Processing Systems, doi: https://doi.org/10.48550/arXiv.2002.03469, 2020.

[155] T. P. Minka, Expectation propagation for approximate Bayesian inference, in: Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence, UAI'01, AUAI Press, Seattle, Washington, USA, 362–369, doi: https://doi.org/10.48550/arXiv.1301.2294, 2001.

[156] S. L. Lauritzen, Propagation of probabilities, means, and variances in mixed graphical association models, Journal of the American Statistical Association 87 (420) (1992) 1098–1108, doi: https://doi.org/10.2307/2290647.

[157] M. Opper, O. Winther, A Bayesian approach to on-line learning doi: https://doi.org/10.2277/0521652634.

[158] G. Shen, X. Chen, Z. Deng, Variational learning of Bayesian neural networks via Bayesian dark knowledge, in: Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, 2037–2043, doi: https://doi.org/10.24963/ijcai.2020/282, 2021.

[159] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958.

[160] Y. Gal, Z. Ghahramani, A theoretically grounded application of dropout in recurrent neural networks, in: Advances in Neural Information Processing Systems, 1019–1027, doi: https://doi.org/10.48550/arXiv.1512.05287, 2016.

[161] Y. Gal, Z. Ghahramani, Bayesian convolutional neural networks with Bernoulli approximate variational inference, arXiv preprint arXiv:1506.02158 Doi: https://doi.org/10.48550/arXiv.1506.02158.

[162] I. Osband, Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout, in: NIPS Workshop on Bayesian Deep Learning, vol. 192, 2016.

[163] I. Alarab, S. Prakoonwit, M. I. Nacer, Illustrative discussion of mc-dropout in general dataset: Uncertainty estimation in bitcoin, Neural Processing Letters 53 (2) (2021) 1001–1011, doi: https://doi.org/10.1007/s11063-021-10424-x.

[164] J. Caldeira, B. Nord, Deeply uncertain: comparing methods of uncertainty quantification in deep learning algorithms, Machine Learning: Science and Technology 2 (1) (2020) 015002, doi: https://doi.org/10.1088/2632-2153/aba6f3.

[165] A. Foong, D. Burt, Y. Li, R. Turner, On the expressiveness of approximate inference in Bayesian neural networks, Advances in Neural Information Processing Systems 33 (2020) 15897–15908, doi: https://doi.org/10.48550/arXiv.1909.00719.

[166] F. Verdoja, V. Kyrki, Notes on the behavior of MC dropout doi: https://doi.org/10.48550/arXiv.2008.02627.

[167] D. Opitz, R. Maclin, Popular ensemble methods: An empirical study, Journal of Artificial Intelligence Research 11 (1999) 169–198, doi: https://doi.org/10.1613/jair.614.

[168] T. G. Dietterich, Ensemble methods in machine learning, in: International Workshop on Multiple Classifier Systems, Springer, 1–15, doi: https://doi.org/10.1007/3-540-45014-9_1, 2000.

[169] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140, doi: https://doi.org/10.1007/BF00058655.

[170] R. E. Schapire, Y. Freund, Boosting: Foundations and algorithms, Kybernetes doi: https://doi.org/10.7551/mitpress/8291.001.0001.

[171] X. Zhang, S. Mahadevan, Ensemble machine learning models for aviation incident risk prediction, Decision Support Systems 116 (2019) 48–63, doi: https://doi.org/10.1016/j.dss.2018.10.009.

[172] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, J. Snoek, Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift, Advances in Neural Information Processing Systems 32, doi: https://doi.org/10.48550/arXiv.1906.02530.

[173] D. A. Nix, A. S. Weigend, Estimating the mean and variance of the target probability distribution, in: Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94), vol. 1, IEEE, 55–60, doi: https://doi.org/10.1109/ICNN.1994.374138, 1994.

[174] S. Fort, H. Hu, B. Lakshminarayanan, Deep ensembles: A loss landscape perspective doi: https://doi.org/10.48550/arXiv.1912.02757.

[175] J. Dodson, A. Downey, S. Laflamme, M. D. Todd, A. G. Moura, Y. Wang, Z. Mao, P. Avitabile, E. Blasch, High-rate structural health monitoring and prognostics: An overview, Data Science in Engineering, Volume 9 (2022) 213–217doi: https://doi.org/10.1007/978-3-030-76004-5_23.

[176] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, P. Pérez, Deep reinforcement learning for autonomous driving: A survey, IEEE Transactions on Intelligent Transportation Systems 23 (6) (2021) 4909–4926, doi: https://doi.org/10.1109/TITS.2021.3054625.

[177] J. Van Amersfoort, L. Smith, Y. W. Teh, Y. Gal, Uncertainty estimation using a single deep deterministic neural network, in: International Conference on Machine Learning, PMLR, 9690–9700, doi: https://doi.org/10.48550/arXiv.2003.02037, 2020.

[178] J. Mukhoti, A. Kirsch, J. van Amersfoort, P. H. Torr, Y. Gal, Deterministic neural networks with appropriate inductive biases capture epistemic and aleatoric uncertainty, arXiv preprint arXiv:2102.11582 .

[179] J. van Amersfoort, L. Smith, A. Jesson, O. Key, Y. Gal, On feature collapse and deep kernel learning for single forward pass uncertainty doi: https://doi.org/10.48550/arXiv.2102.11409.

[180] J. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax Weiss, B. Lakshminarayanan, Simple and principled uncertainty estimation with deterministic deep learning via distance awareness, Advances in Neural Information Processing Systems 33 (2020) 7498–7512, doi: https://doi.org/10.48550/arXiv.2006.10108.

[181] V. Fortuin, M. Collier, F. Wenzel, J. Allingham, J. Liu, D. Tran, B. Lakshminarayanan, J. Berent, R. Jenatton, E. Kokiopoulou, Deep classifiers with label noise modeling and distance awareness doi: https://doi.org/10.48550/arXiv.2110.02609.

[182] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. C. Courville, Improved training of wasserstein gans, Advances in Neural Information Processing Systems 30, doi: https://doi.org/10.48550/arXiv.1704.00028.

[183] T. Miyato, T. Kataoka, M. Koyama, Y. Yoshida, Spectral normalization for generative adversarial networks doi: https://doi.org/10.48550/arXiv.1802.05957.

[184] J. Postels, M. Segu, T. Sun, L. Van Gool, F. Yu, F. Tombari, On the practicality of deterministic epistemic uncertainty doi: https://doi.org/10.48550/arXiv.2107.00649.

[185] J. Van Landeghem, M. Blaschko, B. Anckaert, M.-F. Moens, Benchmarking scalable predictive uncertainty in text classification, IEEE Access 10 (2022) 43703–43737, doi: https://doi.org/10.1109/ACCESS.2022.3168734.

[186] M. H. DeGroot, S. E. Fienberg, The comparison and evaluation of forecasters, Journal of the Royal Statistical Society: Series D (The Statistician) 32 (1-2) (1983) 12–22, doi: https://doi.org/10.2307/2987588.

[187] B. Zadrozny, C. Elkan, Transforming classifier scores into accurate multiclass probability estimates, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge discovery and Data Mining, 694–699, doi: https://doi.org/10.1145/775047.775151, 2002.

[188] A. Niculescu-Mizil, R. Caruana, Predicting good probabilities with supervised learning, in: Proceedings of the 22nd International Conference on Machine Learning, 625–632, doi: https://doi.org/10.1145/1102351.1102430, 2005.

[189] Y. Liu, W. Chen, P. Arendt, H.-Z. Huang, Toward a better understanding of model validation metrics, Journal of Mechanical Design 133 (7), doi: https://doi.org/10.1115/1.4004223.

[190] M. P. Naeini, G. Cooper, M. Hauskrecht, Obtaining well calibrated probabilities using Bayesian binning, in: Twenty-Ninth AAAI Conference on Artificial Intelligence, doi: https://doi.org/10.1609/aaai.v29i1.9602, 2015.

[191] V. Kuleshov, N. Fenner, S. Ermon, Accurate uncertainties for deep learning using calibrated regression, in: International Conference on Machine Learning, PMLR, 2796–2804, doi: https://doi.org/10.48550/arXiv.1807.00263, 2018.

[192] J. Platt, et al., Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods, Advances in Large Margin Classifiers 10 (3) (1999) 61–74.

[193] C. Guo, G. Pleiss, Y. Sun, K. Q. Weinberger, On calibration of modern neural networks, in: International Conference on Machine Learning, PMLR, 1321–1330, doi: `https://doi.org/10.48550/arXiv.1706.04599`, 2017.

[194] D. Roman, S. Saxena, V. Robu, M. Pecht, D. Flynn, Machine learning pipeline for battery state-of-health estimation, Nature Machine Intelligence 3 (5) (2021) 447–456, doi: `https://doi.org/10.1038/s42256-021-00312-3`.

[195] S. Ferson, W. L. Oberkampf, L. Ginzburg, Model validation and predictive capability for the thermal challenge problem, Computer Methods in Applied Mechanics and Engineering 197 (29-32) (2008) 2408–2430, doi: `https://doi.org/10.1016/j.cma.2007.07.030`.

[196] C. Kondermann, R. Mester, C. Garbe, A statistical confidence measure for optical flows, in: European Conference on Computer Vision, Springer, 290–301, doi: `https://doi.org/10.1007/978-3-540-88690-7_22`, 2008.

[197] A. Amini, W. Schwarting, A. Soleimany, D. Rus, Deep evidential regression, Advances in Neural Information Processing Systems 33 (2020) 14927–14937, doi: `https://doi.org/10.48550/arXiv.1910.02600`.

[198] E. Ilg, O. Cicek, S. Galesso, A. Klein, O. Makansi, F. Hutter, T. Brox, Uncertainty estimates and multi-hypotheses networks for optical flow, in: Proceedings of the European Conference on Computer Vision (ECCV), 652–667, doi: `https://doi.org/10.1007/978-3-030-01234-2_40`, 2018.

[199] T. Hastie, R. Tibshirani, J. H. Friedman, J. H. Friedman, The elements of statistical learning: data mining, inference, and prediction, vol. 2, Springer, doi: `https://doi.org/10.1007/978-0-387-84858-7`, 2009.

[200] F. D'Angelo, V. Fortuin, Repulsive deep ensembles are Bayesian, Advances in Neural Information Processing Systems 34 (2021) 3451–3465, doi: `https://doi.org/10.48550/arXiv.2106.11642`.

[201] C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals, Understanding deep learning (still) requires rethinking generalization, Communications of the ACM 64 (3) (2021) 107–115, doi: `https://doi.org/10.1145/3446776`.

[202] O. Fink, Q. Wang, M. Svensén, P. Dersin, W.-J. Lee, M. Ducoffe, Potential, Challenges and Future Directions for Deep Learning in Prognostics and Health Management Applications, doi: `https://doi.org/10.1016/j.engappai.2020.103678`, 2020.

[203] L. Biggio, I. Kastanis, Prognostics and health management of industrial assets: Current progress and road ahead, Frontiers in Artificial Intelligence 3 (2020) 578613, doi: `https://doi.org/10.3389/frai.2020.578613`.

[204] B. Wang, Y. Lei, N. Li, T. Yan, Deep separable convolutional network for remaining useful life prediction of machinery, Mechanical systems and signal processing 134 (2019) 106330, doi: https://doi.org/10.1016/j.ymssp.2019.106330.

[205] J. Lee, E. Lapira, B. Bagheri, H.-a. Kao, Recent advances and trends in predictive manufacturing systems in big data environment, Manufacturing letters 1 (1) (2013) 38–41, doi: https://doi.org/10.1016/j.mfglet.2013.09.005.

[206] J. Lee, E. Lapira, S. Yang, A. Kao, Predictive manufacturing system-Trends of next-generation production systems, IFAC proceedings volumes 46 (7) (2013) 150–156, doi: https://doi.org/10.3182/20130522-3-BR-4036.00107.

[207] A. Saxena, J. Celaya, E. Balaban, K. Goebel, B. Saha, S. Saha, M. Schwabacher, Metrics for evaluating performance of prognostic techniques, in: 2008 International Conference on Prognostics and Health Management, IEEE, 1–17, doi: https://doi.org/10.1109/PHM.2008.4711436, 2008.

[208] L. Biggio, T. Bendinelli, C. Kulkarni, O. Fink, Dynaformer: A Deep Learning Model for Ageing-aware Battery Discharge Prediction doi: https://doi.org/10.48550/arXiv.2206.02555.

[209] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, P. Hennig, Laplace redux-effortless Bayesian deep learning, Advances in Neural Information Processing Systems 34 (2021) 20089–20103, doi: https://doi.org/10.48550/arXiv.2106.14806.

[210] A. G. Wilson, The Case for Bayesian Deep Learning, doi: https://doi.org/10.48550/arXiv.2001.10995, 2020.

[211] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, M. Bennamoun, Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users, IEEE Computational Intelligence Magazine 17 (2) (2022) 29–48, doi: https://doi.org/10.1109/MCI.2022.3155327.

[212] M. Teye, H. Azizpour, K. Smith, Bayesian Uncertainty Estimation for Batch Normalized Deep Networks, doi: https://doi.org/10.48550/arXiv.1802.06455, 2018.

[213] H. Ritter, A. Botev, D. Barber, A Scalable Laplace Approximation for Neural Networks, in: International Conference on Learning Representations, 2018.

[214] Y. Wang, Y. Zhao, S. Addepalli, Remaining useful life prediction using deep learning approaches: A review, Procedia Manufacturing 49 (2020) 81–88, doi: https://doi.org/10.1016/j.promfg.2020.06.015.

[215] P. Rokhforoz, B. Gjorgiev, G. Sansavini, O. Fink, Multi-agent maintenance scheduling based on the coordination between central operator and decentralized producers in an electricity market, Reliability Engineering & System Safety 210 (2021) 107495, doi: https://doi.org/10.1016/j.ress.2021.107495.

[216] P. Rokhforoz, M. Montazeri, O. Fink, Safe multi-agent deep reinforcement learning for joint bidding and maintenance scheduling of generation units, Reliability Engineering & System Safety 232 (2023) 109081, doi: https://doi.org/10.48550/arXiv.2112.10459.

[217] E. Zio, Prognostics and Health Management (PHM): Where are we and where do we (need to) go in theory and practice, Reliability Engineering & System Safety 218 (2022) 108119, doi: https://doi.org/10.1016/j.ress.2021.108119.

[218] A. Saxena, J. Celaya, B. Saha, S. Saha, K. Goebel, Metrics for Offline Evaluation of Prognostic Performance, International Journal of Prognostics and Health Management 1 (1), doi: https://doi.org/10.36001/ijphm.2010.v1i1.1336.

[219] C. Louizos, M. Welling, Multiplicative Normalizing Flows for Variational Bayesian Neural Networks, URL https://arxiv.org/abs/1703.01961, 2017.

[220] L. L. Folgoc, V. Baltatzis, S. Desai, A. Devaraj, S. Ellis, O. E. M. Manzanera, A. Nair, H. Qiu, J. Schnabel, B. Glocker, Is MC Dropout Bayesian?, doi: https://doi.org/10.48550/arXiv.2110.04286, 2021.

[221] K. A. Severson, P. M. Attia, N. Jin, N. Perkins, B. Jiang, Z. Yang, M. H. Chen, M. Aykol, P. K. Herring, D. Fraggedakis, et al., Data-driven prediction of battery cycle life before capacity degradation, Nature Energy 4 (5) (2019) 383–391, doi: https://doi.org/10.1038/s41560-019-0356-8.

[222] P. M. Attia, A. Grover, N. Jin, K. A. Severson, T. M. Markov, Y.-H. Liao, M. H. Chen, B. Cheong, N. Perkins, Z. Yang, et al., Closed-loop optimization of fast-charging protocols for batteries with machine learning, Nature 578 (7795) (2020) 397–402, doi: https://doi.org/10.1038/s41586-020-1994-5.

[223] M. Arias Chao, C. Kulkarni, K. Goebel, O. Fink, Aircraft engine run-to-failure dataset under real flight conditions for prognostics and diagnostics, Data 6 (1) (2021) 5, doi: https://doi.org/10.3390/data6010005.

[224] M. A. Chao, C. Kulkarni, K. Goebel, O. Fink, Fusing physics-based and deep learning models for prognostics, Reliability Engineering & System Safety 217 (2022) 107961, doi: https://doi.org/10.1016/j.ress.2021.107961.

[225] Y. Tian, M. A. Chao, C. Kulkarni, K. Goebel, O. Fink, Real-time model calibration with deep reinforcement learning, Mechanical Systems and Signal Processing 165 (2022) 108284, doi: https://doi.org/10.1016/j.ymssp.2021.108284.

[226] T. Song, C. Liu, R. Wu, Y. Jin, D. Jiang, A hierarchical scheme for remaining useful life prediction with long short-term memory networks, Neurocomputing 487 (2022) 22–33, doi: https://doi.org/10.1016/j.neucom.2022.02.032.

[227] H. Mo, G. Iacca, Multi-Objective Optimization of Extreme Learning Machine for Remaining Useful Life Prediction, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 191–206, doi: https://doi.org/10.1007/978-3-031-02462-7_13, 2022.

[228] M. A. Chao, C. Kulkarni, K. Goebel, O. Fink, Fusing physics-based and deep learning models for prognostics, Reliability Engineering & System Safety 217 (2022) 107961, doi: https://doi.org/10.1016/j.ress.2021.107961.

[229] I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE transactions on neural networks 9 (5) (1998) 987–1000.

[230] J. Cursi, A. Koscianski, Physically constrained neural network models for simulation, in: Advances and Innovations in Systems, Computing Sciences and Software Engineering, Springer, 567–572, 2007.

[231] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707, doi: https://doi.org/10.1016/j.jcp.2018.10.045.

[232] T. Ritto, F. Rochinha, Digital twin, physics-based model, and machine learning applied to damage detection in structures, Mechanical Systems and Signal Processing 155 (2021) 107614, doi: https://doi.org/10.1016/j.ymssp.2021.107614.

[233] F. Oviedo, Z. Ren, S. Sun, C. Settens, Z. Liu, N. T. P. Hartono, S. Ramasamy, B. L. DeCost, S. I. Tian, G. Romano, et al., Fast and interpretable classification of small X-ray diffraction datasets using data augmentation and deep neural networks, npj Computational Materials 5 (1) (2019) 60.

[234] B. Kapusuzoglu, S. Mahadevan, Physics-informed and hybrid machine learning in additive manufacturing: application to fused filament fabrication, Jom 72 (12) (2020) 4695–4705, doi: https://doi.org/10.1007/s11837-020-04438-4.

[235] Y. A. Yucesan, F. A. Viana, A physics-informed neural network for wind turbine main bearing fatigue, International Journal of Prognostics and Health Management 11 (1), doi: https://doi.org/10.36001/ijphm.2020.v11i1.2594.

[236] C. Jiang, M. A. Vega, M. D. Todd, Z. Hu, Model correction and updating of a stochastic degradation model for failure prognostics of miter gates, Reliability Engineering & System Safety 218 (2022) 108203, doi: https://doi.org/10.1016/j.ress.2021.108203.

[237] M. L. Thompson, M. A. Kramer, Modeling chemical processes using prior knowledge and neural networks, AIChE Journal 40 (8) (1994) 1328–1340, doi: https://doi.org/10.1002/aic.690400806.

[238] J.-X. Wang, J.-L. Wu, H. Xiao, Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data, Physical Review Fluids 2 (3) (2017) 034603, doi: https://doi.org/10.1103/PhysRevFluids.2.034603.

[239] A. Thelen, Y. H. Lui, S. Shen, S. Laflamme, S. Hu, H. Ye, C. Hu, Integrating physics-based modeling and machine learning for degradation diagnostics of lithium-ion batteries, Energy Storage Materials 50 (2022) 668–695, doi: https://doi.org/10.1016/j.ensm.2022.05.047.

[240] M.-J. Azzi, C. Ghnatios, P. Avery, C. Farhat, Acceleration of a Physics-Based Machine Learning Approach for Modeling and Quantifying Model-Form Uncertainties and Performing Model Updating, Journal of Computing and Information Science in Engineering 23 (1) (2023) 011009, doi: https://doi.org/10.1115/1.4055546.

[241] W. Chen, Q. Wang, J. S. Hesthaven, C. Zhang, Physics-informed machine learning for reduced-order modeling of nonlinear problems, Journal of Computational Physics 446 (2021) 110666, doi: https://doi.org/10.1016/j.jcp.2021.110666.

[242] H. Gong, S. Cheng, Z. Chen, Q. Li, Data-enabled physics-informed machine learning for reduced-order modeling digital twin: application to nuclear reactor physics, Nuclear Science and Engineering 196 (6) (2022) 668–693, doi: https://doi.org/10.1080/00295639.2021.2014752.

[243] Y. A. Yucesan, F. A. Viana, A hybrid physics-informed neural network for main bearing fatigue prognosis under grease quality variation, Mechanical Systems and Signal Processing 171 (2022) 108875, doi: https://doi.org/10.1016/j.ymssp.2022.108875.

[244] V. Ramadesigan, K. Chen, N. A. Burns, V. Boovaragavan, R. D. Braatz, V. R. Subramanian, Parameter estimation and capacity fade analysis of lithium-ion batteries using reformulated models, Journal of the Electrochemical Society 158 (9) (2011) A1048, doi: https://doi.org/10.1149/1.3609926.

[245] A. Downey, Y.-H. Lui, C. Hu, S. Laflamme, S. Hu, Physics-based prognostics of lithium-ion battery using non-linear least squares with dynamic bounds, Reliability Engineering & System Safety 182 (2019) 1–12, doi: https://doi.org/10.1016/j.ress.2018.09.018.

[246] Y. H. Lui, M. Li, A. Downey, S. Shen, V. P. Nemani, H. Ye, C. VanElzen, G. Jain, S. Hu, S. Laflamme, et al., Physics-based prognostics of implantable-grade lithium-ion battery for remaining useful life prediction, Journal of Power Sources 485 (2021) 229327, doi: https://doi.org/10.1016/j.jpowsour.2020.229327.

[247] P. Ramuhalli, L. Udpa, S. S. Udpa, Finite-element neural networks for solving differential equations, IEEE Transactions on Neural Networks 16 (6) (2005) 1381–1392, doi: https://doi.org/10.1109/TNN.2005.857945.

[248] J. Darbon, T. Meng, On some neural network architectures that can represent viscosity solutions of certain high dimensional Hamilton–Jacobi partial differential equations, Journal

of Computational Physics 425 (2021) 109907, doi: https://doi.org/10.1016/j.jcp.2020.109907.

[249] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via Deep-ONet based on the universal approximation theorem of operators, Nature Machine Intelligence 3 (3) (2021) 218–229, doi: https://doi.org/10.1038/s42256-021-00302-5.

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

[251] S. Cai, Z. Mao, Z. Wang, M. Yin, G. E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: A review, Acta Mechanica Sinica 37 (12) (2021) 1727–1738.

[252] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, Journal of Computational Physics 394 (2019) 56–81, doi: https://doi.org/10.1016/j.jcp.2019.05.024.

[253] L. Yang, D. Zhang, G. E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, SIAM Journal on Scientific Computing 42 (1) (2020) A292–A317, doi: https://doi.org/10.1137/18M1225409.

[254] L. Sun, J.-X. Wang, Physics-constrained bayesian neural network for fluid flow reconstruction with sparse and noisy data, Theoretical and Applied Mechanics Letters 10 (3) (2020) 161–169, doi: https://doi.org/10.1016/j.taml.2020.01.031.

[255] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics–informed neural networks: Where we are and what's next, Journal of Scientific Computing 92 (3) (2022) 88, doi: https://doi.org/10.1007/s10915-022-01939-z.

[256] C. Soize, R. G. Ghanem, C. Safta, X. Huan, Z. P. Vane, J. C. Oefelein, G. Lacaze, H. N. Najm, Q. Tang, X. Chen, Entropy-based closure for probabilistic learning on manifolds, Journal of Computational Physics 388 (1) (2019) 518–533, doi: https://doi.org/10.1016/j.jcp.2018.12.029.

[257] C. Soize, R. Ghanem, Probabilistic learning on manifolds constrained by nonlinear partial differential equations for small datasets, Computer Methods in Applied Mechanics and Engineering 380 (2021) 113777, doi: https://doi.org/10.1016/j.cma.2021.113777.

[258] C. Soize, R. Ghanem, C. Safta, X. Huan, Z. P. Vane, J. C. Oefelein, G. Lacaze, H. N. Najm, Enhancing Model Predictability for a Scramjet Using Probabilistic Learning on Manifolds, AIAA Journal 57 (1) (2019) 365–378, doi: https://doi.org/10.2514/1.J057069.

[259] R. G. Ghanem, C. Soize, C. Safta, X. Huan, G. Lacaze, J. C. Oefelein, H. N. Najm, Design optimization of a scramjet under uncertainty using probabilistic learning on manifolds, Journal of Computational Physics 399 (2019) 108930, doi: https://doi.org/10.1016/j.jcp.2019.108930.

[260] R. Ghanem, C. Soize, L. Mehrez, V. Aitharaju, Probabilistic learning and updating of a digital twin for composite material systems, International Journal for Numerical Methods in Engineering 123 (13) (2022) 3004–3020, doi: https://doi.org/10.1002/nme.6430.

[261] A. Thelen, X. Zhang, O. Fink, Y. Lu, S. Ghosh, B. D. Youn, M. D. Todd, S. Mahadevan, C. Hu, Z. Hu, A comprehensive review of digital twin—part 2: roles of uncertainty quantification and optimization, a battery digital twin, and perspectives, Structural and Multidisciplinary Optimization 66 (1) (2023) 1, doi: https://doi.org/10.1007/s00158-022-03476-7.

[262] D. Angelis, F. Sofos, T. E. Karakasidis, Artificial Intelligence in Physical Sciences: Symbolic Regression Trends and Perspectives, Archives of Computational Methods in Engineering (2023) 1–21Doi: https://doi.org/10.1007/s11831-023-09922-z.

[263] S. L. Brunton, J. L. Proctor, J. N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proceedings of the National Academy of Sciences 113 (15) (2016) 3932–3937, doi: https://doi.org/10.1073/pnas.1517384113.

[264] S. H. Rudy, S. L. Brunton, J. L. Proctor, J. N. Kutz, Data-driven discovery of partial differential equations, Science Advances 3 (4) (2017) e1602614, doi: https://doi.org/10.1098/10.1126/sciadv.1602614.

[265] K. Kaheman, J. N. Kutz, S. L. Brunton, SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics, Proceedings of the Royal Society A 476 (2242) (2020) 20200279, doi: https://doi.org/10.1098/rspa.2020.0279.

[266] S. M. Hirsh, D. A. Barajas-Solano, J. N. Kutz, Sparsifying priors for Bayesian uncertainty quantification in model discovery, Royal Society Open Science 9 (2) (2022) 211823, doi: https://doi.org/10.1098/rsos.211823.

[267] N. M. Mangan, T. Askham, S. L. Brunton, J. N. Kutz, J. L. Proctor, Model selection for hybrid dynamical systems via sparse regression, Proceedings of the Royal Society A 475 (2223) (2019) 20180534, doi: https://doi.org/10.1098/rspa.2018.0534.

[268] N. Wiener, The homogeneous chaos, American Journal of Mathematics 60 (4) (1938) 897–936, doi: https://doi.org/10.2307/2371268.

[269] R. G. Ghanem, P. D. Spanos, Stochastic finite elements: a spectral approach, Courier Corporation, 2003.

[270] D. Xiu, G. E. Karniadakis, The Wiener–Askey polynomial chaos for stochastic differential equations, SIAM Journal on Scientific Computing 24 (2) (2002) 619–644, doi: https://doi.org/10.1137/S1064827501387826.

[271] O. P. Le Maıtre, M. T. Reagan, H. N. Najm, R. G. Ghanem, O. M. Knio, A stochastic projection method for fluid flow: II. Random process, Journal of Computational Physics 181 (1) (2002) 9–44, doi: https://doi.org/10.1006/jcph.2002.7104.

[272] M. Berveiller, B. Sudret, M. Lemaire, Stochastic finite element: a non intrusive approach by regression, European Journal of Computational Mechanics/Revue Européenne de Mécanique Numérique 15 (1-3) (2006) 81–92, doi: https://doi.org/10.3166/remn.15.81-92.

[273] S. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions, Dokl. Akad. Nauk SSSR 148 (5) (1963) 1042–1045.

[274] P. G. Constantine, M. S. Eldred, E. T. Phipps, Sparse pseudospectral approximation method, Computer Methods in Applied Mechanics and Engineering 229-232 (2012) 1–12, ISSN 00457825, doi: https://doi.org/10.1016/j.cma.2012.03.019.

[275] P. R. Conrad, Y. M. Marzouk, Adaptive Smolyak Pseudospectral Approximations, SIAM Journal on Scientific Computing 35 (6) (2013) A2643–A2670, ISSN 1064-8275, doi: https://doi.org/10.1137/120890715.

[276] G. Blatman, B. Sudret, Adaptive sparse polynomial chaos expansion based on least angle regression, Journal of Computational Physics 230 (6) (2011) 2345–2367, doi: https://doi.org/10.1016/j.jcp.2010.12.021.

[277] J. Hampton, A. Doostan, Compressive sampling of polynomial chaos expansions: Convergence analysis and sampling strategies, Journal of Computational Physics 280 (2015) 363–386, doi: https://doi.org/10.1016/j.jcp.2014.09.019.

[278] P. Tsilifis, X. Huan, C. Safta, K. Sargsyan, G. Lacaze, J. C. Oefelein, H. N. Najm, R. G. Ghanem, Compressive sensing adaptation for polynomial chaos expansions, Journal of Computational Physics 380 (2019) 29–47, doi: https://doi.org/10.1016/j.jcp.2018.12.010.

[279] G. Blatman, B. Sudret, An adaptive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis, Probabilistic Engineering Mechanics 25 (2) (2010) 183–197, doi: https://doi.org/10.1016/j.probengmech.2009.10.003.

[280] C. Hu, B. D. Youn, Adaptive-sparse polynomial chaos expansion for reliability analysis and design of complex engineering systems, Structural and Multidisciplinary Optimization 43 (2011) 419–442, doi: https://doi.org/10.1007/s00158-010-0568-9.

[281] Q. Pan, D. Dias, Sliced inverse regression-based sparse polynomial chaos expansions for reliability analysis in high dimensions, Reliability Engineering & System Safety 167 (2017) 484–493, doi: https://doi.org/10.1016/j.ress.2017.06.026.

[282] J. Xu, F. Kong, A cubature collocation based sparse polynomial chaos expansion for efficient structural reliability analysis, Structural Safety 74 (2018) 24–31, doi: https://doi.org/10.1016/j.strusafe.2018.04.001.

[283] B. Bhattacharyya, Structural reliability analysis by a Bayesian sparse polynomial chaos expansion, Structural Safety 90 (2021) 102074, doi: https://doi.org/10.1016/j.strusafe.2020.102074.

[284] N. Lüthen, S. Marelli, B. Sudret, Sparse polynomial chaos expansions: Literature survey and benchmark, SIAM/ASA Journal on Uncertainty Quantification 9 (2) (2021) 593–649, doi: https://doi.org/10.1137/20M1315774.

[285] R. Schobi, B. Sudret, J. Wiart, Polynomial-chaos-based Kriging, International Journal for Uncertainty Quantification 5 (2), doi: https://doi.org/10.1615/Int.J.UncertaintyQuantification.2015012467.

[286] P. Kersaudy, B. Sudret, N. Varsier, O. Picon, J. Wiart, A new surrogate modeling technique combining Kriging and polynomial chaos expansions–Application to uncertainty analysis in computational dosimetry, Journal of Computational Physics 286 (2015) 103–117, doi: https://doi.org/10.1016/j.jcp.2015.01.034.

[287] B. Pavlack, J. Paixão, S. Da Silva, A. Cunha Jr, D. Garcia Cava, Polynomial Chaos-Kriging metamodel for quantification of the debonding area in large wind turbine blades, Structural Health Monitoring 21 (2) (2022) 666–682, doi: https://doi.org/10.1177/14759217211007956.

[288] X. Shang, P. Ma, M. Yang, T. Chao, An efficient polynomial chaos-enhanced radial basis function approach for reliability-based design optimization, Structural and Multidisciplinary Optimization 63 (2021) 789–805, doi: https://doi.org/10.1007/s00158-020-02730-0.

[289] E. Torre, S. Marelli, P. Embrechts, B. Sudret, Data-driven polynomial chaos expansion for machine learning regression, Journal of Computational Physics 388 (2019) 601–623, doi: https://doi.org/10.1016/j.jcp.2019.03.039.

[290] Z. Nado, N. Band, M. Collier, J. Djolonga, M. W. Dusenberry, S. Farquhar, Q. Feng, A. Filos, M. Havasi, R. Jenatton, et al., Uncertainty Baselines: Benchmarks for uncertainty & robustness in deep learning, arXiv preprint arXiv:2106.04015 URL: https://doi.org/10.48550/arXiv.2106.04015.

[291] H. Li, J. Yin, X. Du, Uncertainty Quantification of Physics-Based Label-Free Deep Learning and Probabilistic Prediction of Extreme Events, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 86236, American Society of Mechanical Engineers, V03BT03A001, doi: https://doi.org/10.1115/DETC2022-88277, 2022.

[292] R. M. Neal, Bayesian Learning for Neural Networks, Springer-Verlag New York, New York, NY, doi: https://doi.org/10.1007/978-1-4612-0745-0, 1996.

[293] C. Williams, Computing with infinite networks, Advances in Neural Information Processing Systems 9.

[294] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, J. Sohl-Dickstein, Deep Neural Networks as Gaussian Processes, in: ICLR, 2018.

[295] R. Novak, L. Xiao, J. Lee, Y. Bahri, G. Yang, J. Hron, D. A. Abolafia, J. Pennington, J. Sohl-Dickstein, Bayesian deep convolutional networks with many channels are Gaussian processes, in: NIPS Workshop on Bayesian Deep Learning, 2018.

[296] A. Garriga-Alonso, C. E. Rasmussen, L. Aitchison, Deep convolutional networks as shallow Gaussian processes, in: ICLR, 2019.

[297] Y. Cho, L. Saul, Kernel methods for deep learning, Advances in Neural Information Processing Systems 22.

[298] A. G. Wilson, Z. Hu, R. Salakhutdinov, E. P. Xing, Deep kernel learning, in: Artificial intelligence and statistics, PMLR, 370–378, 2016.

[299] A. Damianou, N. D. Lawrence, Deep Gaussian processes, in: Artificial intelligence and statistics, PMLR, 207–215, doi: https://doi.org/10.48550/arXiv.1211.0358, 2013.

[300] T. Bui, D. Hernández-Lobato, J. Hernandez-Lobato, Y. Li, R. Turner, Deep Gaussian processes for regression using approximate expectation propagation, in: International Conference on Machine Learning, PMLR, 1472–1481, 2016.

[301] H. Salimbeni, M. Deisenroth, Doubly stochastic variational inference for deep Gaussian processes, Advances in Neural Information Processing Systems 30.

[302] M. Havasi, J. M. Hernández-Lobato, J. J. Murillo-Fuentes, Inference in deep Gaussian processes using stochastic gradient Hamiltonian Monte Carlo, Advances in Neural Information Processing Systems 31.

[303] M. Fuge, B. Peters, A. Agogino, Machine learning algorithms for recommending design methods, Journal of Mechanical Design 136 (10) (2014) 101103, doi: https://doi.org/10.1115/1.4028102.

[304] J. H. Panchal, M. Fuge, Y. Liu, S. Missoum, C. Tucker, Machine learning for engineering design, Journal of Mechanical Design 141 (11), doi: https://doi.org/10.1115/1.4044690.

[305] C. A. Vale, K. Shea, et al., A machine learning-based approach to accelerating computational design synthesis, in: DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm, 183–184, 2003.

[306] C. Fan, L. Zeng, Y. Sun, Y.-Y. Liu, Finding key players in complex networks through deep reinforcement learning, Nature Machine Intelligence 2 (6) (2020) 317–324, doi: https://doi.org/10.1038/s42256-020-0177-2.

[307] J. Jiang, Y. Xiong, Z. Zhang, D. W. Rosen, Machine learning integrated design for additive manufacturing, Journal of Intelligent Manufacturing (2020) 1–14Doi: https://doi.org/10.1007/s10845-020-01715-6.

[308] S. M. Moosavi, K. M. Jablonka, B. Smit, The role of machine learning in the understanding and design of materials, Journal of the American Chemical Society 142 (48) (2020) 20273–20287, doi: https://doi.org/10.1021/jacs.0c09105.

[309] Q. Tao, P. Xu, M. Li, W. Lu, Machine learning for perovskite materials design and discovery, NPJ Computational Materials 7 (1) (2021) 1–18, doi: https://doi.org/10.1038/s41524-021-00495-8.

[310] M. Moustapha, B. Sudret, Surrogate-assisted reliability-based design optimization: a survey and a unified modular framework, Structural and Multidisciplinary Optimization 60 (5) (2019) 2157–2176, doi: https://doi.org/10.1007/s00158-019-02290-y.

[311] A. Perera, P. Wickramasinghe, V. M. Nik, J.-L. Scartezzini, Machine learning methods to assist energy system optimization, Applied Energy 243 (2019) 191–205, doi: https://doi.org/10.1016/j.apenergy.2019.03.202.

[312] X. Lei, C. Liu, Z. Du, W. Zhang, X. Guo, Machine learning-driven real-time topology optimization under moving morphable component-based framework, Journal of Applied Mechanics 86 (1) (2019) 011004, doi: https://doi.org/10.1115/1.4041319.

[313] G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507, doi: https://doi.org/10.1126/science.1127647.

[314] C. Qian, R. K. Tan, W. Ye, An adaptive artificial neural network-based generative design method for layout designs, International Journal of Heat and Mass Transfer 184 (2022) 122313, doi: https://doi.org/10.1016/j.ijheatmasstransfer.2021.122313.

[315] L. Regenwetter, A. H. Nobari, F. Ahmed, Deep generative models in engineering design: A review, Journal of Mechanical Design 144 (7) (2022) 071704, doi: https://doi.org/10.1115/1.4053859.

[316] K. M. Hamdia, H. Ghasemi, Y. Bazi, H. AlHichri, N. Alajlan, T. Rabczuk, A novel deep learning based method for the computational material design of flexoelectric nanostructures with topology optimization, Finite Elements in Analysis and Design 165 (2019) 21–30, doi: https://doi.org/10.1016/j.finel.2019.07.001.

[317] Z. Yang, X. Li, L. Catherine Brinson, A. N. Choudhary, W. Chen, A. Agrawal, Microstructural materials design via deep adversarial learning methodology, Journal of Mechanical Design 140 (11), doi: https://doi.org/10.1115/1.4041371.

[318] R. Alizadeh, J. K. Allen, F. Mistree, Managing computational complexity using surrogate models: a critical review, Research in Engineering Design 31 (3) (2020) 275–298, doi: https://doi.org/10.1007/s00163-020-00336-7.

[319] M. C. Kennedy, A. O'Hagan, Bayesian calibration of computer models, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63 (3) (2001) 425–464, doi: https://doi.org/10.1111/1467-9868.00294.

[320] K. Cheng, Z. Lu, C. Ling, S. Zhou, Surrogate-assisted global sensitivity analysis: an overview, Structural and Multidisciplinary Optimization 61 (3) (2020) 1187–1213, doi: https://doi.org/10.1007/s00158-019-02413-5.

[321] T. Chatterjee, S. Chakraborty, R. Chowdhury, A critical review of surrogate assisted robust design optimization, Archives of Computational Methods in Engineering 26 (1) (2019) 245–274, doi: https://doi.org/10.1007/s11831-017-9240-5.

[322] F. A. Viana, R. T. Haftka, V. Steffen, Multiple surrogates: how cross-validation errors can help us to obtain the best predictor, Structural and Multidisciplinary Optimization 39 (4) (2009) 439–457, doi: https://doi.org/10.1007/s00158-008-0338-0.

[323] R. Jin, X. Du, W. Chen, The use of metamodeling techniques for optimization under uncertainty, Structural and Multidisciplinary Optimization 25 (2) (2003) 99–116, doi: https://doi.org/10.1007/s00158-002-0277-0.

[324] Z. Hu, S. Mahadevan, A single-loop kriging surrogate modeling for time-dependent reliability analysis, Journal of Mechanical Design 138 (6), doi: https://doi.org/10.1115/1.4033428.

[325] B. Gaspar, A. P. Teixeira, C. G. Soares, Assessment of the efficiency of Kriging surrogate models for structural reliability analysis, Probabilistic Engineering Mechanics 37 (2014) 24–34, doi: https://doi.org/10.1016/j.probengmech.2014.03.011.

[326] X. Zhang, L. Wang, J. D. Sørensen, REIF: a novel active-learning function toward adaptive Kriging surrogate models for structural reliability analysis, Reliability Engineering & System Safety 185 (2019) 440–454, doi: https://doi.org/10.1016/j.ress.2019.01.014.

[327] L. Yan, T. Zhou, Adaptive multi-fidelity polynomial chaos approach to Bayesian inference in inverse problems, Journal of Computational Physics 381 (2019) 110–128, doi: https://doi.org/10.1016/j.jcp.2018.12.025.

[328] Y. Zhang, D. W. Apley, W. Chen, Bayesian optimization for materials design with mixed quantitative and qualitative variables, Scientific Reports 10 (1) (2020) 1–13, doi: https://doi.org/10.1038/s41598-020-60652-9.

[329] US NSTC, Materials Genome Initiative for global competitiveness, Executive Office of the President, National Science and Technology Council, 2011.

[330] E. Lander, K. Koizumi, J. Christodoulou, L. Sapochak, L. E. Friedersdorf, J. Warren, Materials genome initiative strategic plan (2021), National Science And Technology Council .

[331] D. McDowell, J. Scott, et al., Creating the Next-Generation Materials Genome Initiative Workforce, Tech. Rep., The Minerals Metals and Materials Society, 2019.

[332] J. J. de Pablo, N. E. Jackson, M. A. Webb, L.-Q. Chen, J. E. Moore, D. Morgan, R. Jacobs, T. Pollock, D. G. Schlom, E. S. Toberer, et al., New frontiers for the materials genome

initiative, NPJ Computational Materials 5 (1) (2019) 1–23, doi: https://doi.org/10.1038/s41524-019-0173-4.

[333] J. Christodoulou, L. E. Friedersdorf, L. Sapochak, J. A. Warren, The second decade of the Materials Genome Initiative, doi: https://doi.org/10.1007/s11837-021-05008-y, 2021.

[334] H. Sasaki, H. Igarashi, Topology optimization accelerated by deep learning, IEEE Transactions on Magnetics 55 (6) (2019) 1–5, doi: https://doi.org/10.1109/TMAG.2019.2901906.

[335] N. A. Kallioras, G. Kazakis, N. D. Lagaros, Accelerated topology optimization by means of deep learning, Structural and Multidisciplinary Optimization 62 (3) (2020) 1185–1212, doi: https://doi.org/10.1007/s00158-020-02545-z.

[336] Y. Xiao, S. Nazarian, P. Bogdan, Self-optimizing and self-programming computing systems: A combined compiler, complex networks, and machine learning approach, IEEE transactions on Very Large Scale Integration (VLSI) Systems 27 (6) (2019) 1416–1427, doi: https://doi.org/10.1109/TVLSI.2019.2897650.

[337] Z. Hu, S. Mahadevan, Global sensitivity analysis-enhanced surrogate (GSAS) modeling for reliability analysis, Structural and Multidisciplinary Optimization 53 (3) (2016) 501–521, doi: https://doi.org/10.1007/s00158-015-1347-4.

[338] J. Li, B. Wang, Z. Li, Y. Wang, An improved active learning method combing with the weight information entropy and Monte Carlo simulation of efficient structural reliability analysis, Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 235 (19) (2021) 4296–4313, doi: https://doi.org/10.1177/0954406220973233.

[339] U. Alibrandi, L. V. Andersen, E. Zio, Informational probabilistic sensitivity analysis and active learning surrogate modelling, Probabilistic Engineering Mechanics (2022) 103359Doi: https://doi.org/10.1016/j.probengmech.2022.103359.

[340] M. K. Sadoughi, C. Hu, C. A. MacKenzie, A. T. Eshghi, S. Lee, Sequential exploration-exploitation with dynamic trade-off for efficient reliability analysis of complex engineered systems, Structural and Multidisciplinary Optimization 57 (1) (2018) 235–250, doi: https://doi.org/10.1007/s00158-017-1748-7.

[341] S. S. Afshari, F. Enayatollahi, X. Xu, X. Liang, Machine learning-based methods in structural reliability analysis: A review, Reliability Engineering & System Safety 219 (2022) 108223, doi: https://doi.org/10.1016/j.ress.2021.108223.

[342] P. I. Frazier, Bayesian optimization, in: Recent advances in optimization and modeling of contemporary problems, INFORMS, 255–278, doi: https://doi.org/10.1287/educ.2018.0188, 2018.

[343] W. Shen, X. Huan, Bayesian sequential optimal experimental design for nonlinear models using policy gradient reinforcement learning, arXiv preprint arXiv:2110.15335 Doi: https://doi.org/10.48550/arXiv.2110.15335.

[344] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Communications of the ACM 63 (11) (2020) 139–144, doi: https://doi.org/10.1145/3422622.

[345] T. Guo, D. J. Lohan, R. Cang, M. Y. Ren, J. T. Allison, An indirect design representation for topology optimization using variational autoencoder and style transfer, in: 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 0804, doi: https://doi.org/10.2514/6.2018-0804, 2018.

[346] J. Chen, C. Chen, Z. Xing, X. Xia, L. Zhu, J. Grundy, J. Wang, Wireframe-based UI design search through image autoencoder, ACM Transactions on Software Engineering and Methodology (TOSEM) 29 (3) (2020) 1–31, doi: https://doi.org/10.1145/3391613.

[347] X. Li, C. Xie, Z. Sha, A Predictive and Generative Design Approach for Three-Dimensional Mesh Shapes Using Target-Embedding Variational Autoencoder, Journal of Mechanical Design 144 (11) (2022) 114501, doi: https://doi.org/10.1115/1.4054906.

[348] S. Oh, Y. Jung, S. Kim, I. Lee, N. Kang, Deep generative design: Integration of topology optimization and generative models, Journal of Mechanical Design 141 (11), doi: https://doi.org/10.1115/1.4044229.

[349] L. Regenwetter, F. Ahmed, Towards Goal, Feasibility, and Diversity-Oriented Deep Generative Models in Design, arXiv preprint arXiv:2206.07170, Doi: https://doi.org/10.48550/arXiv.2206.07170.

[350] H. Song, K. K. Choi, I. Lee, L. Zhao, D. Lamb, Adaptive virtual support vector machine for reliability analysis of high-dimensional problems, Structural and Multidisciplinary Optimization 47 (4) (2013) 479–491, doi: https://doi.org/10.1007/s00158-012-0857-6.

[351] A. Basudhar, S. Missoum, Adaptive explicit decision functions for probabilistic design and optimization using support vector machines, Computers & Structures 86 (19-20) (2008) 1904–1917, doi: https://doi.org/10.1016/j.compstruc.2008.02.008.

[352] O. Sener, S. Savarese, Active learning for convolutional neural networks: A core-set approach, arXiv preprint arXiv:1708.00489 Doi: https://doi.org/10.48550/arXiv.1708.00489.

[353] J. M. Haut, M. E. Paoletti, J. Plaza, J. Li, A. Plaza, Active learning with convolutional neural networks for hyperspectral image classification using a new Bayesian approach, IEEE Transactions on Geoscience and Remote Sensing 56 (11) (2018) 6440–6461, doi: https://doi.org/10.1109/TGRS.2018.2838665.

[354] Z. Xiang, J. Chen, Y. Bao, H. Li, An active learning method combining deep neural network and weighted sampling for structural reliability analysis, Mechanical Systems and Signal Processing 140 (2020) 106684, doi: https://doi.org/10.1016/j.ymssp.2020.106684.

[355] Y. Bao, Z. Xiang, H. Li, Adaptive subset searching-based deep neural network method for structural reliability analysis, Reliability Engineering & System Safety 213 (2021) 107778, doi: https://doi.org/10.1016/j.ress.2021.107778.

[356] L. C. Nguyen, H. Nguyen-Xuan, Deep learning for computational structural optimization, ISA Transactions 103 (2020) 177–191, doi: https://doi.org/10.1016/j.isatra.2020.03.033.

[357] T. Asano, S. Noda, Optimization of photonic crystal nanocavities based on deep learning, Optics Express 26 (25) (2018) 32704–32717, doi: https://doi.org/10.1364/OE.26.032704.

[358] J. J. Beland, P. B. Nair, Bayesian optimization under uncertainty, in: NIPS BayesOpt 2017 workshop, 2017.

[359] A. Mathern, O. S. Steinholtz, A. Sjöberg, M. Önnheim, K. Ek, R. Rempling, E. Gustavsson, M. Jirstrand, Multi-objective constrained Bayesian optimization for structural design, Structural and Multidisciplinary Optimization 63 (2) (2021) 689–701, doi: https://doi.org/10.1007/s00158-020-02720-2.

[360] P. I. Frazier, J. Wang, Bayesian optimization for materials design, in: Information Science for Materials Discovery and Design, Springer, 45–75, doi: https://doi.org/10.1007/978-3-319-23871-5_3, 2016.

[361] C. Sharpe, C. C. Seepersad, S. Watts, D. Tortorelli, Design of mechanical metamaterials via constrained Bayesian optimization, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 51753, American Society of Mechanical Engineers, V02AT03A029, doi: https://doi.org/10.1115/DETC2018-85270, 2018.

[362] L. F. F. Miguel, R. H. Lopez, A. J. Torii, A. T. Beck, Reliability-based optimization of multiple Folded Pendulum TMDs through Efficient Global Optimization, Engineering Structures 266 (2022) 114524, doi: https://doi.org/10.1016/j.engstruct.2022.114524.

[363] D. Liu, Y. Wang, Metal Additive Manufacturing Process Design based on Physics Constrained Neural Networks and Multi-Objective Bayesian Optimization, Manufacturing Letters 33 (2022) 817–827, doi: https://doi.org/10.1016/j.mfglet.2022.07.101.

[364] L. Le Gratiet, J. Garnier, Recursive co-kriging model for design of computer experiments with multiple levels of fidelity, International Journal for Uncertainty Quantification 4 (5), doi: https://doi.org/10.1615/Int.J.UncertaintyQuantification.2014006914.

[365] M. A. Álvarez, L. Rosasco, N. D. Lawrence, Kernels for Vector-Valued Functions: A Review, Foundations and Trends in Machine Learning 4 (3) (2012) 195–266, doi: https://doi.org/10.1561/2200000036.

[366] R. P. Dwight, Z.-H. Han, Efficient uncertainty quantification using gradient-enhanced kriging, AIAA paper 2276 (2009) 2009, doi: https://doi.org/10.2514/6.2009-2276.

[367] A. Tran, M. Tran, Y. Wang, Constrained mixed-integer Gaussian mixture Bayesian optimization and its applications in designing fractal and auxetic metamaterials, Structural and Multidisciplinary Optimization 59 (2019) 2131–2154, doi: https://doi.org/10.1007/s00158-018-2182-1.

[368] C. Paciorek, M. Schervish, Nonstationary covariance functions for Gaussian process regression, Advances in Neural Information Processing Systems 16.

[369] M. Heinonen, H. Mannerström, J. Rousu, S. Kaski, H. Lähdesmäki, Non-stationary Gaussian process regression with Hamiltonian Monte Carlo, in: Artificial Intelligence and Statistics, PMLR, 732–740, 2016.

[370] S. Remes, M. Heinonen, S. Kaski, Non-stationary spectral kernels, Advances in Neural Information Processing Systems 30.

[371] M. Schwabacher, K. Goebel, A Survey of Artificial Intelligence for Prognostics., in: AAAI fall symposium: artificial intelligence for prognostics, Arlington, VA, 108–115, 2007.

[372] M. Kefalas, B. van Stein, M. Baratchi, A. Apostolidis, T. Bäck, An End-to-End Pipeline for Uncertainty Quantification and Remaining Useful Life Estimation: An Application on Aircraft Engines 7 (2022) 245–260, doi: https://doi.org/10.36001/phme.2022.v7i1.3317.

[373] J. Lee, M. Mitici, Deep reinforcement learning for predictive aircraft maintenance using Probabilistic Remaining-Useful-Life prognostics, Reliability Engineering & System Safety (2022) 108908Doi: https://doi.org/10.1016/j.ress.2022.108908.

[374] G. Mazaev, G. Crevecoeur, S. Van Hoecke, Bayesian convolutional neural networks for remaining useful life prognostics of solenoid valves with uncertainty estimations, IEEE Transactions on Industrial Informatics 17 (12) (2021) 8418–8428, doi: https://doi.org/10.1109/TII.2021.3078193.

[375] R. Zhu, Y. Chen, W. Peng, Z.-S. Ye, Bayesian deep-learning for RUL prediction: An active learning perspective, Reliability Engineering & System Safety 228 (2022) 108758, doi: https://doi.org/10.1016/j.ress.2022.108758.

[376] J. Yang, Y. Peng, J. Xie, P. Wang, Remaining Useful Life Prediction Method for Bearings Based on LSTM with Uncertainty Quantification, Sensors 22 (12) (2022) 4549, doi: https://doi.org/10.3390/s22124549.

[377] G. Li, L. Yang, C.-G. Lee, X. Wang, M. Rong, A Bayesian deep learning RUL framework integrating epistemic and aleatoric uncertainties, IEEE Transactions on Industrial Electronics 68 (9) (2020) 8829–8841, doi: https://doi.org/10.1109/TIE.2020.3009593.

[378] Y.-H. Lin, G.-H. Li, A Bayesian Deep Learning Framework for RUL Prediction Incorporating Uncertainty Quantification and Calibration, IEEE Transactions on Industrial Informatics Doi: https://doi.org/10.1109/TII.2022.3156965.

[379] M. Wei, H. Gu, M. Ye, Q. Wang, X. Xu, C. Wu, Remaining useful life prediction of lithium-ion batteries based on Monte Carlo Dropout and gated recurrent unit, Energy Reports 7 (2021) 2862–2871, doi: https://doi.org/10.1016/j.egyr.2021.05.019.

[380] Y. Kong, X. Zhang, S. Mahadevan, Bayesian Deep Learning for Aircraft Hard Landing Safety Assessment, IEEE Transactions on Intelligent Transportation Systems 23 (10) (2022) 17062–17076, doi: https://doi.org/10.1109/TITS.2022.3162566.

[381] W. Peng, Z.-S. Ye, N. Chen, Bayesian deep-learning-based health prognostics toward prognostics uncertainty, IEEE Transactions on Industrial Electronics 67 (3) (2019) 2283–2293, doi: https://doi.org/10.1109/TIE.2019.2907440.

[382] S. Xiang, Y. Qin, J. Luo, F. Wu, K. Gryllias, A concise self-adapting deep learning network for machine remaining useful life prediction, Mechanical Systems and Signal Processing 191 (2023) 110187, ISSN 0888-3270, doi: https://doi.org/10.1016/j.ymssp.2023.110187.

[383] M. Xu, P. Baraldi, S. Al-Dahidi, E. Zio, Fault Prognostics by an Ensemble of Echo State Networks in Presence of Event Based Measurements, Engineering Applications of Artificial Intelligence 87 (2019) 103346, doi: https://doi.org/10.1016/j.engappai.2019.103346.

[384] J. Zgraggen, G. Pizza, L. G. Huber, Uncertainty Informed Anomaly Scores with Deep Learning: Robust Fault Detection with Limited Data, in: PHM Society European Conference, vol. 7, 530–540, doi: https://doi.org/10.36001/phme.2022.v7i1.3342, 2022.

[385] Y. Liao, L. Zhang, C. Liu, Uncertainty prediction of remaining useful life using long short-term memory network based on bootstrap method, in: 2018 IEEE International Conference on Prognostics and Health Management (ICPHM), IEEE, 1–8, doi: https://doi.org/10.1109/ICPHM.2018.8448804, 2018.

[386] M. G. Rigamonti, P. Baraldi, E. Zio, I. Roychoudhury, K. Goebel, S. Poll, Ensemble of optimized echo state networks for remaining useful life prediction, Neurocomputing 281 (2017) 121–138, doi: https://doi.org/10.1016/j.neucom.2017.11.062.

[387] L. Biggio, A. Wieland, M. A. Chao, I. Kastanis, O. Fink, Uncertainty-Aware Prognosis via Deep Gaussian Process, IEEE Access 9 (2021) 123517–123527, doi: https://doi.org/10.1109/ACCESS.2021.3110049.

[388] B. Ellis, P. S. Heyns, S. Schmidt, A hybrid framework for remaining useful life estimation of turbomachine rotor blades, Mechanical Systems and Signal Processing 170 (2022) 108805, doi: https://doi.org/10.1016/j.ymssp.2022.108805.

[389] M. Jankowiak, G. Pleiss, J. R. Gardner, Deep Sigma Point Processes URL https://arxiv.org/abs/2002.09112.

Appendix A. Some further discussions on Gaussian process regression

Appendix A.1. An extended discussion on kernels

The class of Matérn kernels represents a very general class of covariance functions, of which the squared exponential kernel is a special case. It offers a broad class of kernels with varying values of a smoothness parameter $\nu > 0$ that controls the smoothness of the resulting approximation of the underlying function [106]. The Matérn covariance between the function outputs at two points are described as [106]

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \frac{1}{\Gamma(\nu) 2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{\ell} dist\left(\mathbf{x}, \mathbf{x}'\right) \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}}{\ell} dist\left(\mathbf{x}, \mathbf{x}'\right) \right) \tag{A.1}$$

where $\Gamma(\cdot)$ is the Gamma function, $dist(\mathbf{x}, \mathbf{x}')$ is the Euclidean distance between points $\mathbf{x}$ and $\mathbf{x}'$, i.e., $dist(\mathbf{x}, \mathbf{x}') = |\mathbf{x} - \mathbf{x}'| = \sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}$, and K_ν is the modified Bessel function of the second kind and order ν. A larger value of ν results in a smoother appropriated function. When $\nu \to \infty$, the Matérn kernel becomes the squared exponential kernel. Another special case worth mentioning is when $\nu = 1/2$, the Matérn kernel is equivalent to the absolute exponential kernel (sometimes also called the Ornstein-Uhlenbeck process kernel), which can be expressed as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{dist\left(\mathbf{x}, \mathbf{x}'\right)}{l} \right). \tag{A.2}$$

GPR using this Matérn 1/2 kernel yields rather unsmooth (rough) functions sampled from the Gaussian process prior and posterior. Additionally, observations do not inform predictions on input points far away from the points of observations, leading to poor generalization performance of the resulting GPR model. Two other special cases of the Matérn kernels are $\nu = 3/2$ and $\nu = 5/2$. The resulting Matérn 3/2 kernel and Matérn 5/2 kernel are not infinitely differentiable, unlike the squared exponential kernel, but at least once (Matérn 3/2) or twice differentiable ($\nu = 5/2$). These two kernels may be useful in cases where intermediate solutions between the unsmooth Matérn 1/2 kernel and the perfectly smooth squared exponential kernel are needed to approximate functions that are expected to be somewhat smooth yet not perfectly smooth.

The Matérn kernel in Eq. (A.1) has a single length scale l and is of an isotropic form. Like the ARD squared exponential kernel shown in Eq. (11), an anisotropic variant of the Matérn kernel can be defined by introducing D length scales, each depicting the relevance of an input dimension. The resulting ARD Matérn kernel has a slightly modified term, $\sqrt{\sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{l_d^2}}$, in place of the original term, $\frac{\sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}}{l}$ (i.e., $\frac{dist(\mathbf{x}, \mathbf{x}')}{l}$ in Eq. (A.1)). For D-dimensional input $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$, an anisotropic kernel is composed of $(D+1)$ hyperparameters, $\sigma_f, l_1, \ldots, l_D$.

To illustrate the concept of kernels, Fig. A.24 compares GPR models built using multiple commonly used kernels in a 1D example. As demonstrated in this figure, the squared-exponential kernel produces the smoothest GPR, whereas Matérn1/2 produces the roughest GPR (where the samples drawn from the posterior are equivalent to a Brownian motion). The intuition is that the
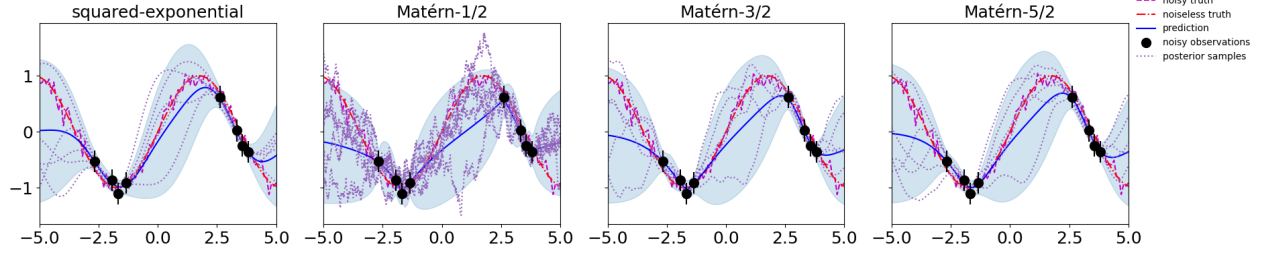
Figure A.24: Comparison of GPR models built using multiple kernels: squared-exponential ($\nu \to \infty$), Matérn1/2 ($\nu = \frac{1}{2}$), Matérn3/2 ($\nu = \frac{3}{2}$)), Matérn5/2 ($\nu = \frac{5}{2}$), with the same eight training data points, along with five samples randomly drawn from the posterior.

larger the ν value, the smoother the underlying function. Specifically, when $\nu = 1/2$, the Gaussian process sampled from posterior with this kernel (Matérn1/2) corresponds to a Brownian motion (or equivalently, a Wiener process), whereas $\nu \to \infty$ smoothens the sampled Gaussian process because the posterior mean is infinitely differentiable (i.e., $\mathcal{C}^\infty$) [106]. The noiseless ground truth, $f(x) = \sin(0.9x)$, is plotted as dot-dashed magenta lines. Each noisy observation used for training is obtained based on the following observation model: $y = f(x) + \varepsilon$, where the Gaussian noise $\varepsilon \sim \mathcal{N}(0, 0.1^2)$. Eight training observations are plotted as black dots, and five samples randomly drawn from the GPR posterior are plotted as dotted purple lines.

Appendix A.2. Parametric study on effect of hyperparameter optimization

Figure A.25 illustrates the effect of l, σ_f, and σ_ε on the Gaussian process posterior of observations $\mathbf{y}_*$ (each being function output f plus noise ε) for the 1D toy example used in Fig. 5. In each of the four cases considered, the values of the three hyperparameters and log marginal likelihood (see Eq. 20) are shown right below the regression plot. In all four cases, the observation ($\mathbf{y}_*$) posterior has the same mean curve as the function ($\mathbf{f}_*$) posterior but a slightly larger variance at any input point due to the non-zero noise variance σ_ε^2, as discussed in Sec. 3.1.1.d. The length scale determines how quickly the correlation between the function values at two input points decays as they become farther away. Too small of an l value (e.g., $l = 0.1$ in Fig. A.25) leads to an approximation that varies too quickly horizontally and yields too wide of uncertainty regions between training points. The signal amplitude σ_f depicts the maximum vertical variation of functions/observations drawn from the Gaussian process. A larger σ_f value (e.g., $\sigma_\mathrm{f} = 3$ in Fig. A.25) results in a larger maximum width of the confidence interval for a test point between or away from training points. It is an important hyperparameter for quantifying epistemic uncertainty, although it is difficult to derive an optimum value solely based on training data. The signal standard deviation σ_ε controls the amount of (input-independent) noise in the observations. Too small of a σ_ε value (e.g., $\sigma_\varepsilon = 0.05$ in Fig. A.25) results in an approximation that fails to capture the observational noise (aleatory uncertainty).

Appendix A.3. Connections with neural networks and recent development

Efforts to draw connections between GPR and neural networks dated back more than two decades, with the first study showing the equivalence between a Gaussian process and a fully-
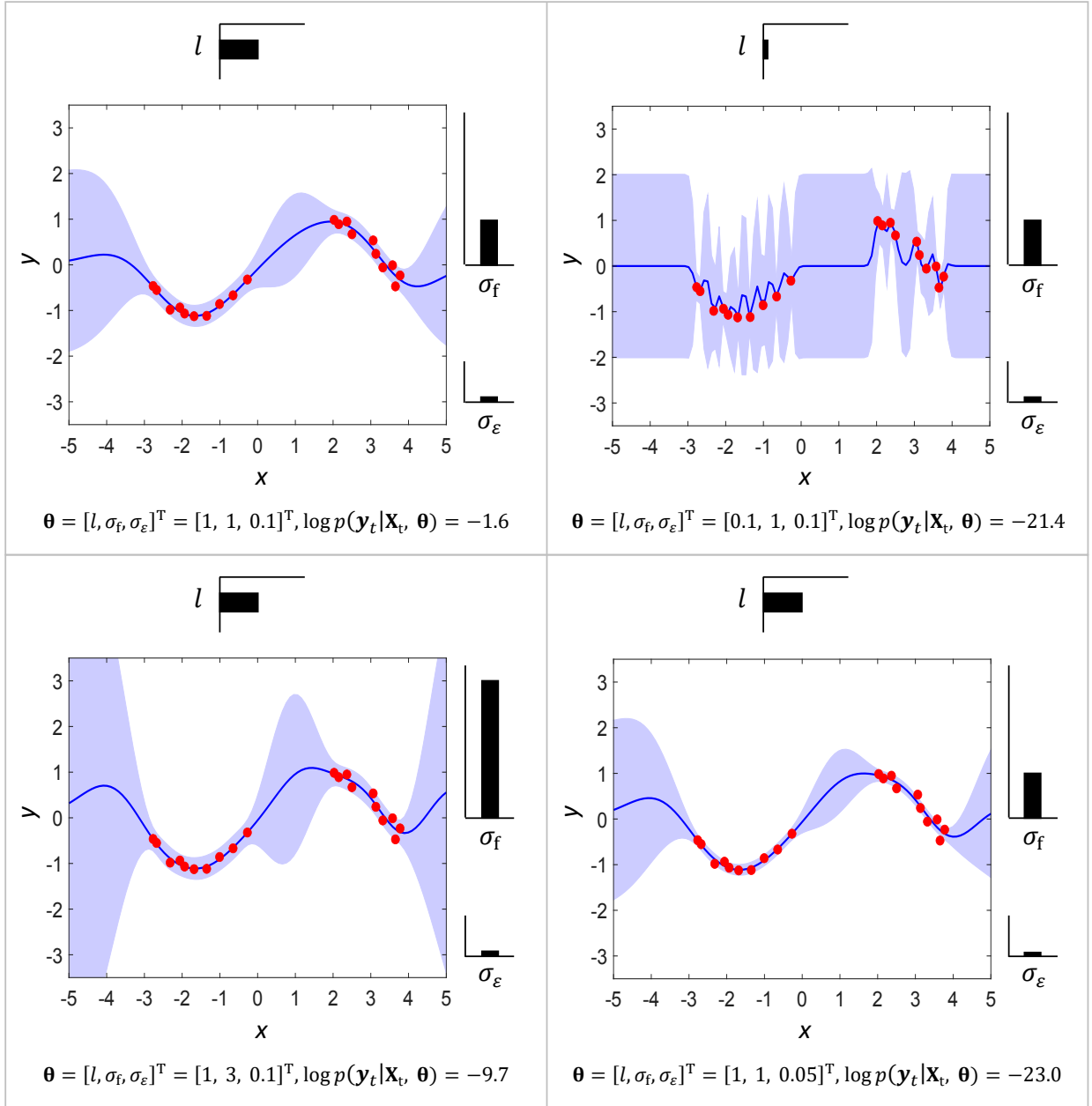
Figure A.25: Effect of hyperparameters on the Gaussian process posterior for the 1D toy example used in Fig. 5. Note that the confidence intervals shown collectively as light blue shade are derived from the posterior of (noisy) observations (function output plus noise); they are slightly wider than the confidence intervals for the underlying function shown in Fig. 5 due to the added Gaussian noise (see the discussion below Eqs. (18) and (19) in Sec. 3.1.1.d).

connected neural network with a single, *infinite-width* hidden layer and an i.i.d. prior over the network parameters (weights and biases) [292]. This equivalence is significant because using a Gaussian process prior over functions allows one to perform Bayesian inference in its exact form on neural networks using simple matrix operations (see the familiar formulae for Gaussian process

posterior in Eqs. (16) and (17)) [293]. One obvious benefit is that one does not need to resort to iterative, more computationally expensive training algorithms, such as gradient descent and stochastic gradient descent, or approximate Bayesian inference methods for Bayesian neural networks (see Sec. 3.2). As deep learning has been gaining popularity in recent years, significant extensions were made to draw such connections for standard DNNs [294] and DNNs with convolutional filters, or so-called deep convolutional neural networks [295, 296].
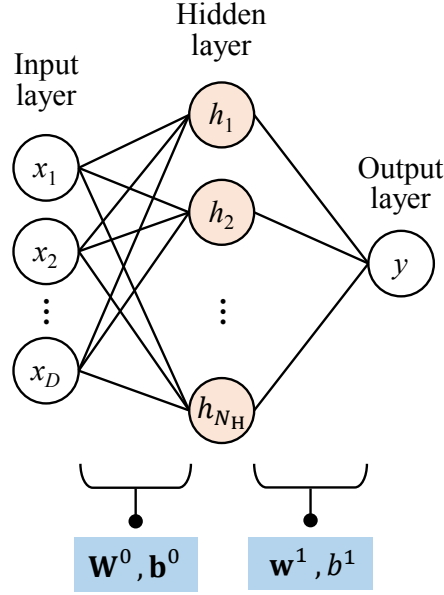

Figure A.26: A single-hidden-layer neural network where the number of hidden units N_H could approach infinity, i.e., $N_H \to \infty$. $\mathbf{W}^0$ and $\mathbf{b}^0$ conveniently denote the $N_H \times D$ matrix of input-to-hidden weights and the vector of N_H input-to-hidden biases. Similarly, $\mathbf{w}^1$ denotes the vector of N_H hidden-to-output weights, again, for notational convenience purposes.

Let us now briefly review the early work in [292]. We consider a fully-connected neural network with one hidden layer, illustrated in Fig. A.26. To get to each hidden node h_j, $1 \leq j \leq N_H$, where N_H is the number of hidden units, we first apply a linear transformation of input point $\mathbf{x}$ and then a nonlinear operation using ... function $\psi(\cdot) : \mathbb{R}^D \mapsto \mathbb{R}$. The resulting j-th hidden unit takes the following form:

$$h_j(\mathbf{x}) = \psi \left(b_j^0 + \sum_{d=1}^{D} w_{dj}^0 x_d \right), \tag{A.3}$$

where w_{dj}^0 denotes the input-to-hidden weight from x_d to h_j and b_j^0 is the input-to-hidden bias for h_j. To get to the output node y (assuming zero observation noise for simplicity, i.e., $y(\mathbf{x}) = f(\mathbf{x})$), we apply another linear transformation of the hidden units with hidden-to-output weights and a bias

$$y(\mathbf{x}) = b^1 + \sum_{j=1}^{N_H} w_j^1 h_j(\mathbf{x}), \tag{A.4}$$

where w_j^1 denotes the hidden-to-output weight from h_j to y, and b^1 is the hidden-to-output bias.

We assume (1) the prior of the hidden-to-output weights w_j^1 and bias b follows independent zero-mean (often Gaussian) distributions with variances being $\sigma_{w^1}^2$ and σ_b^2, respectively, and (2) the input-to-hidden weights w_{dj}^0 and biases b_j^0 are i.i.d. It follows that the network output $y(\mathbf{x})$ in Eq. (A.4) is a summation over $(N_H + 1)$ i.i.d. random variables [292]. Based on the Central Limit Theorem, when $N_H \to \infty$, i.e., when the width of the hidden layer approaches infinity, $\widehat{y}(\mathbf{x})$ will follow a Gaussian distribution. This Gaussian prior holds regardless of the distribution types of the $(N_H + 1)$ random variables in the sum. Let us move on to look at any finite set of input points, $\mathbf{x}_1, \ldots, \mathbf{x}_{N_*}$. As $N_H \to \infty$, their network outputs, $\widehat{y}_1, \ldots, \widehat{y}_{N_*}$, will be jointly Gaussian, according to the multidimensional Central Limit Theorem. It means that the joint distribution of the network outputs at any finite collection of input points is multivariate Gaussian, which exactly matches the definition of a Gaussian process discussed in Sec. 3.1.1.a. Thus, $\widehat{y}(\mathbf{x}) \sim \mathcal{GP}(m_{nn}(\mathbf{x}), k_{nn}(\mathbf{x}, \mathbf{x}'))$, a Gaussian process with the mean function $m_{nn}(\cdot)$ and covariance function $k_{nn}(\cdot)$. Since the hidden-to-output weights w_j^1 and bias b^1 have zero means, $m_{nn} \equiv \mathbb{E}[\widehat{y}(\mathbf{x})] = 0$. The covariance function can be derived based on i.i.d. conditions and takes the following form:

$$k_{nn}(\mathbf{x}, \mathbf{x}') \equiv \mathbb{E}\left[\widehat{y}(\mathbf{x})\widehat{y}(\mathbf{x}')\right] = \sigma_{b^1}^2 + \sum_{j=1}^{N_H} \sigma_{w^1}^2 \mathbb{E}\left[h_j(\mathbf{x})h_j(\mathbf{x}')\right] = \sigma_{b^1}^2 + \underbrace{N_H \sigma_{w^1}^2}_{\omega^2} \underbrace{\mathbb{E}\left[h_j(\mathbf{x})h_j(\mathbf{x}')\right]}_{\mathcal{C}(\mathbf{x},\mathbf{x}')}, \quad \text{(A.5)}$$

where the prior variance $\sigma_{w^1}^2$ of each hidden-to-output weight is set to scale carefully as ω^2/N_H for some fixed "unscaled" variance ω^2 and $\mathcal{C}(\mathbf{x}, \mathbf{x}')$ need to be evaluated for all $\mathbf{x}$ in the training set and all $\mathbf{x}'$ in the training and test sets. $\mathcal{C}(\mathbf{x}, \mathbf{x}')$ has an analytic form for certain types of activation functions such as the error function (or Gaussian nonlinearities) [106, 293], one-sided polynomial functions [297], and ReLU (rectified linear unit) [294]. As a result, infinitely wide Bayesian neural networks give rise to a new family of GPR kernels. An interesting and attractive property of these neural networks is that all network parameters are often initialized as independent zero-mean Gaussians, some with properly scaled variances, and the kernel parameters (e.g., "unscaled" prior variances of weights and prior variances of biases) may be the only parameters that need to be optimized.

What has been discussed in this subsection represents a category of approaches for combining the strengths of GPR (exact Bayesian inference, distance awareness, etc.) with those of neural networks (feature extraction from high-dimensional inputs (large D), ability to model nonlinearities, etc.). These approaches explore the direct theoretical relationship between infinitely wide neural networks and GPR. Another category of approaches uses GPR with standard kernels (such as the squared exponential kernel in Eq. (10)) whose inputs are feature representations in the hidden space learned by a neural network [179–181, 298]. These approaches are often called *deep kernel learning*. The network weights, biases, and GPR kernel parameters can be jointly optimized end-to-end, which is straightforward to implement using gradient descent or stochastic gradient descent. These approaches excel in OOD detection thanks to the distance awareness property of GPR and offer a solution to improving the scalability of GPR to high-dimensional inputs. A drawback is that overparameterization associated with a DNN (e.g., a deep convolutional neural network) may

make the network prone to overfitting. Another issue is *feature collapse* [177], which needs to be carefully addressed to preserve input distances in the hidden space. This issue will be discussed along with a representative approach in this category called spectral-normalized neural Gaussian process (SNGP) in Sec. 3.4. A third category of approaches aims to mimic the many-layer architecture of a DNN by stacking Gaussian processes on top of one another in a hierarchical form [299–302]. The resulting deep Gaussian processes are probabilistic ML models with the UQ capability brought in by GPR and the added flexibility to learn complex mappings from datasets that can be small or large. However, the performance gains over standard GPR comes at a cost: exact Bayesian inference by deep Gaussian processes can be prohibitively expensive due to the computationally demanding need to compute the inverse and determinant of the covariance matrix. Therefore, almost all deep Gaussian process approaches adopt appropriate inference techniques for efficient model training that use only a small set of the so-called inducing points to build covariance matrixes [300–302].

Appendix B. UQ of ML models in engineering design

Appendix B.1. Needs of ML models in engineering design

In recent years, the rapid advancement of high-performance computing and data analytics techniques has made ML a game changer for engineering design. In particular, ML enables engineers and designers to relax simplifications and assumptions that are usually needed in conventional design paradigms [303, 304], accelerate the design process by shortening the required design cycles [305], and handle the design of highly complex systems with large numbers of design variables [306, 307]. These benefits provided by data-driven ML models are particularly appealing for simulation-based engineering system design, which usually entails costly simulations.

As shown in Fig. B.27, ML revolutionizes engineering design mainly through three categories of ML-enabled capabilities: feature extraction, surrogate modeling, and optimization. Approaches in each of these three categories have been applied to solve challenging engineering design problems in various applications, such as discovery and design of engineering materials [308, 309], design for reliability [310], energy system design [311], and topology optimization [312], to name a few.

i. **Feature extraction**: Extracting informative features from massive volumes of raw data is a representative use case of ML in engineering design. In this regard, ML, particularly deep learning, has become more and more prevalent in engineering design due to its salient characteristic of automatically extracting feature representations from high-dimensional data in its raw form. Specifically, in the context of engineering design, the powerful representation learning ability has been frequently utilized in two types of design activities, namely (1) dimension reduction, which is to reduce the dimensionality of design problems, and (2) generative design, which is to generate candidate designs subject to certain design constraints [313–315].

   (a) For *dimension reduction*, autoencoder, as an unsupervised learning technique, has been commonly adopted to learn efficient codings and compressed knowledge representations from unlabeled data [313]. More specifically, an autoencoder consists of an encoder and
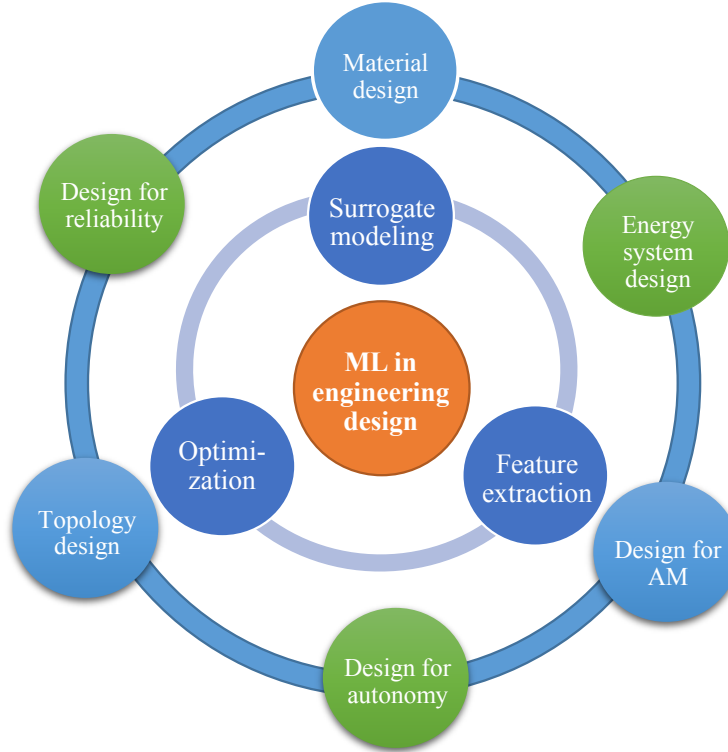
Figure B.27: ML-enabled techniques in engineering design and applications.

a decoder: the encoder transforms high-dimensional data into a low-dimensional representation through a "bottleneck" layer of neurons, while the decoder recovers the high-dimensional data from the low-dimensional code. The encoder and decoder are trained together to minimize the discrepancy between the original data and its reconstruction. Due to their powerful representation capacity, autoencoders and their variants (e.g., sparse autoencoders and variational autoencoders (VAEs)) have been actively employed to extract important features, supporting diverse engineering design tasks [316].

(b) For *generative design*, researchers have investigated ML approaches to aid the design process through automatic design synthesis. In short, generative design is an iterative process of using algorithms to facilitate the exploration of thousands of design variants as guided by the parameters outlined in the study setup to approach an optimal design that meets the performance target. Towards this end, ML has contributed substantially to automating the process of generative design, which is often referred to as automatic design synthesis in the design community. In essence, automatic design synthesis is to learn a generative model from existing designs and then generate new designs meeting design requirements (e.g., performance targets and cost constraints) based on the compact representations of training data in the hidden space. In particular, VAEs and GANs are two popular classes of ML algorithms for generative design [317].

ii. **Surrogate modeling**: It is a process of using ML models as emulators of computationally

expensive computer simulation models in engineering design [24]. With the development of computational mechanics and advanced numerical solvers, computer simulations are getting increasingly sophisticated. The high-fidelity computer simulations allow us to accurately predict complicated physical phenomena without performing large numbers of expensive physical experiments, thereby accelerating the design of engineering systems to meet mission-specific requirements. Although high-fidelity simulations significantly enhance our predictive capability, they present notable challenges to engineering design due to the high computational demand and burden often associated with them. ML models play a vital role in addressing this challenge by maintaining the same predictive capability level as high-fidelity simulations while significantly reducing the computational effort required to make high-fidelity predictions [318]. The basic idea of ML-enabled surrogate modeling is to replace an expensive-to-evaluate high-fidelity simulation model with a much "cheaper" mathematical surrogate, essentially an ML model. Over the past few decades, various surrogate modeling methods have been proposed for different purposes within engineering design, including model calibration [319], reliability analysis [28], sensitivity analysis [320], and optimization [321]. These existing surrogate modeling methods can be broadly classified into two groups:

(a) *Global surrogate modeling for general purposes:* This class of surrogate models is constructed for the general purpose of design optimization and tries to achieve a good prediction accuracy in the whole design region of interest [27, 322, 323]. More specifically, let us use $\hat{y} = \hat{G}(\mathbf{x})$ to represent the surrogate model of a computer simulation model $y = G(\mathbf{x}), \mathbf{x} \in \Omega_{\mathbf{x}}$, where $\Omega_{\mathbf{x}}$ is the prediction domain of the inputs. In global surrogate modeling, we are concerned about the prediction accuracy of $\hat{y} = \hat{G}(\mathbf{x})$ for all $\mathbf{x} \in \Omega_{\mathbf{x}}$. Because of this, the training data for ML model construction needs to spread throughout the whole prediction domain $\Omega_{\mathbf{x}}$, with those in nonlinear regions being denser and the others in relatively smoother regions being more sparse. Various sampling techniques have been developed to efficiently construct globally accurate surrogate models using ML. Some examples of the techniques include MSE-based methods, the A-optimality criterion, and maximin scaled distance approaches [24]. The goal of global surrogate modeling is to construct a surrogate that is fully representative of the original computer simulation model. Since the surrogate model is not constructed for any specific purposes and the prediction accuracy has been verified for all $\mathbf{x} \in \Omega_{\mathbf{x}}$, it can be used for any purposes, such as design optimization, uncertainty analysis, and sensitivity analysis, after its construction. In addition, the UQ calibration metrics presented in Sec. 4.1.3 and Sec. 4.3 can be used to quantify the prediction accuracy of a global surrogate model, if the test data is representative of the design domain $\Omega_{\mathbf{x}}$.

(b) *Local surrogate modeling for specific purposes:* Instead of achieving good prediction accuracy in the whole design region, this group of surrogate models only focuses on prediction in very localized design regions, such as the limit state regions in design for reliability problems [28, 324–326] and important regions for model calibration purposes [327]. In

local surrogate modeling, we are concerned about the prediction accuracy of $\hat{y} = \hat{G}(\mathbf{x})$ for $\mathbf{x} \in \tilde{\Omega}_{\mathbf{x}}$, where $\tilde{\Omega}_{\mathbf{x}} \subset \Omega_{\mathbf{x}}$ is a subset of the prediction domain of the inputs. This sub-domain $\tilde{\Omega}_{\mathbf{x}}$ varies with the specific purpose of the surrogate modeling. For example, when the surrogate model is constructed for the purpose of reliability analysis, which is a classification problem, $\tilde{\Omega}_{\mathbf{x}}$ will be the regions along the limit state or classification boundary. When the surrogate model is constructed for optimization, $\tilde{\Omega}_{\mathbf{x}}$ will be the regions where the optima locate. As a result, the training data for surrogate modeling will be concentrated in those localized regions instead of spreading evenly throughout the whole prediction domain of the inputs. Because we only concentrate on a sub-domain $\tilde{\Omega}_{\mathbf{x}}$ of the input space, $\Omega_{\mathbf{x}}$, the local surrogate model $\hat{y} = \hat{G}(\mathbf{x}), \mathbf{x} \in \tilde{\Omega}_{\mathbf{x}}$ only partially represents the original simulation model (i.e., the surrogate is an accurate representation of the simulation model only in the sub-domain of the design space). Moreover, since the sub-domain $\tilde{\Omega}_{\mathbf{x}}$ is usually unknown during the construction of the surrogate model, learning functions (also called acquisition functions in some methods) are needed to identify these localized sub-domains adaptively based on the currently available information about the underlying simulation model (ground truth). Because the surrogate model is constructed for a specific purpose (e.g., model calibration, reliability analysis, or optimization), its accuracy also needs to be quantified using metrics tailored for that specific purpose. For example, a metric used to check the prediction accuracy of the surrogate model for reliability analysis may not be appropriate for constructing a surrogate model for design optimization.

iii. **Optimization**: Engineering design problems are essentially optimization problems. Conventional gradient-based optimizers often have difficulties in finding global optima. Even though evolutionary optimization methods can overcome some of the limitations of gradient-based optimizers, the former methods are likely to require much larger numbers of function evaluations, which could become prohibitively costly for high-fidelity simulation models in many engineering design problems. ML-based or ML-assisted optimization methods have been proposed to tackle this challenge, resulting in a new family of optimization methods collectively named gradient-free ML-based optimization. One representative example of this family is Bayesian optimization [30]. ML-based optimization transforms the way that engineering systems are designed in many fields, such as new materials [328]. It is worth noting that the Materials Genome Initiative [329, 330, 330–333], firstly debuted in 2011, was embedded in the context of designing new materials using ML and optimization to significantly reduce the research and development time. Moreover, the development of deep learning methods in recent years even allows designers to bypass complicated design optimization by directly generating candidate designs for a particular application. Some examples include the ML-based topology optimization [334, 335] and deep learning-enabled design of large-scale complex networks [336].

Appendix B.2. Role of UQ of ML models in engineering design

An indispensable step for the above-reviewed three categories of ML-enabled techniques (i.e., feature extraction, surrogate modeling, and optimization) is UQ of ML models. For example,

for ML-enabled feature extraction in engineering design, quantifying the predictive uncertainty of ML models play an important role in (1) ensuring the extracted features are representative of the original data sources, (2) eliminating the ill-posedness of inverse problems in generative design, and (3) accounting for variability across input features.

For surrogate modeling in engineering design, an essential step in building an accurate surrogate model (global or local surrogate) is the collection of training data. However, an initial set of training data is usually insufficient to build a surrogate model with satisfactory prediction accuracy. A subsequent refinement step sometimes is needed to improve the prediction accuracy of the surrogate model. Due to the high computational effort required to collect training data from high-fidelity simulations in engineering design, it is desirable to reduce the number of training data points or refinement iterations for surrogate modeling as much as possible. Over the past few decades, numerous refinement strategies have been developed in engineering design to minimize the number of iterations in collecting training data for the purpose of improving the performance of surrogate models. Even though these refinement strategies may differ from each other, they share one notable starting point: *quantifying the predictive uncertainty of the surrogate model for any given input.*

For instance, the most commonly used refinement method for global surrogate modeling is to identify new training data by maximizing the variance of the prediction of the surrogate model [27]. That is a mean squared error-method as mentioned above in Appendix B.1. In a GPR model, the variance of the prediction can be directly obtained from the surrogate model. For other types of surrogate models, however, the predictive uncertainty needs to be quantified using a separate UQ method. Moreover, UQ of ML models becomes particularly important, if local surrogate models need to be constructed for engineering design. In the context of local surrogate modeling, learning functions (also called acquisition functions), such as the expected improvement (EI) function in GPR-based surrogate modeling, are required to identify new training data in critical local regions (i.e., $\tilde{\Omega}_{\mathbf{x}}$ mentioned in Appendix B.1) of the input space. The new training data will then be used to refine the surrogate. Many (20+) learning functions have been proposed in recent years for local surrogate modeling of various purposes (e.g., surrogate construction, reliability analysis, and optimization). These learning functions look into multiple quantitative metrics to examine different aspects crucial to the iterative improvement of surrogate models, such as classification error [337], information entropy [338, 339], and exploitation and exploration [340], among others. A detailed review of various learning functions for local surrogate modeling for reliability analysis is available in Ref. [341]. To the best of our knowledge, nearly all the learning functions for local surrogate modeling heavily rely on UQ of ML models. Let us take a look at two well-known learning functions for local surrogate modeling in reliability-based design optimization: the expected feasibility function (EFF) [28] and the U function [29]. They are mathematically described as

follows:

$$EFF(\mathbf{x}) = \int_{e-\tau}^{e+\tau} [\tau - |e - y|] \, p_{\hat{y}(\mathbf{x})}(y) dy, \tag{B.1a}$$

$$U(\mathbf{x}) = \frac{|\mu_{\hat{y}}(\mathbf{x}) - e|}{\sigma_{\hat{y}}(\mathbf{x})}, \tag{B.1b}$$

where e is the failure threshold used to define the limit state, $y = e$, that separates the failure region ($y > e$) from the safe region ($y \leq e$), τ is half the width of a two-sided critical interval in the vincinity of the limit state ($y = e$), often set as two times the standard deviation of the ML model prediction, i.e., $\tau = 2\sigma_{\hat{y}}(\mathbf{x})$, $\mu_{\hat{y}}(\mathbf{x})$ and $\sigma_{\hat{y}}(\mathbf{x})$ are, respectively, the mean and standard deviation of the ML prediction with respect to the input $\mathbf{x}$, and $p_{\hat{y}(\mathbf{x})}(y)$ is the probability density function of y for given input $\mathbf{x}$ predicted by the ML model.

As shown in the above two equations, UQ of ML models plays an essential role in the construction of such learning functions. This observation also applies to the other learning functions in local surrogate modeling. It is commonly referred as adaptive surrogate modeling in the literature. In general, the identification of the sub-domain $\tilde{\Omega}_{\mathbf{x}}$ (see Appendix B.1) relies on the learning functions in local surrogate modeling, where UQ of ML models plays a foundational role towards the establishment of these learning functions.

Similar to local surrogate modeling, ML-enabled optimization in engineering design also depends heavily on the ability to quantify the predictive uncertainty of ML models, which is essential for ML models to exploit and explore the design domain to efficiently identify optimal designs. Examples of such ML-based optimizers include Bayesian optimization [342] and deep reinforcement learning-based optimization [343]. Specifically for Bayesian optimization, a trade-off between exploitation and exploration is balanced through a learning/acquisition function, which is very similar to that in local surrogate modeling discussed above. Some popular learning functions include the probability of improvement, EI, upper confidence bound, and knowledge gradient (a generalization of EI). Taking the EI function for a minimization problem as an example, this function is mathematically defined as [30].

$$EI(\mathbf{x}) = (f_{\min} - \mu_{\hat{y}}(\mathbf{x}))\Phi\left(\frac{f_{\min} - \mu_{\hat{y}}(\mathbf{x})}{\sigma_{\hat{y}}(\mathbf{x})}\right) + \sigma_{\hat{y}}(\mathbf{x})\phi\left(\frac{f_{\min} - \mu_{\hat{y}}(\mathbf{x})}{\sigma_{\hat{y}}(\mathbf{x})}\right), \tag{B.2}$$

where $f_{\min}$ is the current best function value obtained from the existing training data [30]. As indicated in this equation, $\mu_{\hat{y}}(\mathbf{x})$ and $\sigma_{\hat{y}}(\mathbf{x})$ are two essential elements of the EI function. UQ of ML models is needed to obtain these two terms, and more fundamentally, the probability distribution of $\hat{y}$ is required to derive a learning/acquisition function such as the EI function in Eq. (B.2). Defining such a function makes it possible to accelerate design optimization through ML. This characteristic is very similar to that of learning functions in local surrogate modeling.

In a broad sense, adaptive surrogate modeling-based design optimization can also be classified as a type of local surrogate model since a learning function is used to adaptively identify critical local regions that are important for the specific purpose of identifying a maximum or minimum.

Moreover, a global surrogate model and a local surrogate model are interchangeable during the process of ML model construction. For example, we usually start with a global surrogate model in order to construct a local surrogate model because the critical local regions are unknown and need to be identified using a learning function based on the UQ of an ML model. After constructing a local surrogate model for a specific purpose (e.g., reliability analysis, optimization), we can always convert this local surrogate into a global one if we want to expand the prediction domain to the whole design domain. Regardless of whether design optimization leverages local or global surrogate modeling, UQ of ML models is almost always the foundation of the three categories of ML-enabled capabilities in engineering design described in Appendix B.1.

Appendix B.3. State of knowledge and gaps

Driven by the increasing needs of various engineering design problems (e.g., design for reliability, design for additive manufacturing, new material design, energy system design, etc) as illustrated in Fig. B.27, the three categories of ML-enabled techniques established upon UQ of ML models (see Appendix B.1) have been extensively studied in the literature. Next, we elaborate the current state-of-the-art literature and highlight research gaps that need further investigation and efforts from three aspects: feature extraction, surrogate modeling, and optimization.

According to our literature survey, studies on feature extraction in engineering design mostly implement neural network-based approaches, such as those based on variants of autoencoders and GANs as mentioned in Appendix B.1 [344]. For example, Guo et al. [345] tackled the topology design of a heat conduction system using the latent representation produced by a VAE. Chen et al. [346] trained a wireframe image autoencoder with a large database of unlabeled real-application user interface (UI) designs to serve as a UI search engine for the purpose of supporting UI design in software development. Li et al. [347] developed a target-embedding VAE neural network and explored its usage in the design of 3D car body and mugs. In recent years, the idea of using ML for automatic design synthesis has also gained increasing popularity [315, 348, 349], especially in the mechanical design community. For instance, Zhang et al. [53] used an unsupervised VAE to learn a generative model from a corpus of existing 3D glider designs and demonstrated the utility of the VAE in the 3D outer shape design of gliders. Chen and Fuge [54] developed a generative model established upon a GAN for synthesizing smooth curves, in which the generator first synthesized parameters for rational Bézier curves, and then transformed those parameters into discrete point representations. In another study, Chen and Fuge [55] considered the interpart dependencies and proposed a GAN-based generative model for synthesizing designs by decomposing the synthesis into synthesizing each part conditioned on its corresponding parent part. The UQ methods for ML models presented in Sec. 3 can be directly applied to the aforementioned neural network models to improve the effectiveness of feature extraction in engineering design by enabling dimension reduction or generative design under uncertainty. However, as of now, only a limited number of studies have touched on topics to investigate the UQ of neural networks used in feature extraction.

For global surrogate modeling, approaches have been investigated using various ML methods, including GPR models, neural networks (both regular artificial neural networks and DNNs), sup-

port vector regression, random forest, etc. For local surrogate modeling, however, most current approaches are developed based on GPR models. This is largely attributed to the capability of GPR to analytically quantify the predictive uncertainty in the form of a Gaussian distribution that is convenient to use. In fact, most of the learning functions for local surrogate model-based reliability analysis are derived or developed based on GPR models. For example, learning functions in closed forms as given in Eqs. (B.1a) and (B.1b) have been derived for GPR models. Quantifying the predictive uncertainty of GPR models in the Gaussian form facilitates an efficient evaluation of various learning functions for the refinement of local surrogates. In addition to GPR-based local surrogate modeling methods, a few approaches have also been proposed for local surrogate modeling based on UQ of support vector regression models [350, 351]. In recent years, with the rapid development of deep learning techniques and the capability of quantifying the prediction uncertainty of deep learning models, local surrogate modeling methods have been studied for deep neural networks to achieve "active learning" [352–354]. For instance, Xiang et al. [354] proposed an active learning method for DNN-based structural reliability analysis by extending a weighted sampling method from GPR models to DNNs. This extension allows for selecting new training data for refining DNN models for reliability analysis. Similarly, Bao et al. [355] extended the subset sampling method to DNNs, resulting in an adaptive DNN method for structural reliability analysis. Even though active learning for local surrogate modeling has great potential in reducing the size of training data required to build accurate surrogate models, it is still in the early development stage for other ML models beyond GPR models. In particular, many existing UQ methods for deep learning models are still far from GPR's scientific rigor and theoretical soundness because few can stand strict UQ tests pertaining to uncertainty calibration, decomposition, and attribution. Additionally, even fewer methods offer principled ways to reduce the predictive uncertainty of deep neural networks. With UQ methods for ML models (as reviewed in Sec. 3) getting more and more mature, we foresee that active learning for local surrogate modeling will also become a very active research topic for ML models other than GPR models.

Similar to local surrogate modeling, even though some deep learning-based optimization methods have been developed recently [356, 357], ML-enabled optimization has mostly been studied using GPR models, resulting in a group of Bayesian optimization-based engineering design methods [328, 358, 359], whose applications include material design [360, 361], design for reliability [362], and design for additive manufacturing [363]. Because GPR is a flexible and versatile framework, which means it can be fairly easy to extend to other problems and applications, numerous extensions have been considered to adopt GPR models in different settings under the big umbrella of "Bayesian optimization". These extensions include, but are not limited to, using multi-fidelity strategy to reduce the required number of high-fidelity samples in GPR-based Bayesian optimization [364], Bayesian optimization for multi-output response [365], enhancing Bayesian optimization through gradient information during the construction of a GPR model [366], Bayesian optimization for problems with mixed-integer design variables (also known as mixed-variables) [367], and Bayesian optimization based on heteroscedastic or non-stationary GPR models [118, 368–370].

Based on the above reviews, we can conclude that the UQ methods for ML models reviewed

in Sec. 3 provide valuable tools to fill the gaps in the following three major activities of ML-based engineering design: ML-enabled feature extraction, surrogate modeling, and optimization.

a. **Enabling uncertainty-informed surrogate modeling and optimization**: The UQ methods for neural networks presented in Secs. 3.2 and 3.3 enable us to extend various local surrogate modeling and optimization methods, which are originally developed for GPR models, to various neural network-based ML models. This opportunity is especially important for deep neural networks that are gaining popularity in the engineering design community.

b. **Accounting for aleatory uncertainty in ML-based engineering design**: Most current methods for global surrogate modeling, local surrogate modeling, and ML-based optimization lack the capability of considering input-dependent aleatory uncertainty during the local surrogate modeling or optimization. UQ methods newly developed in the ML community such as the neural network ensemble method reviewed in Sec. 3.3 offer opportunities to address this important issue.

c. **Reducing computational cost**: Computationally efficient UQ methods are needed to quantify the predictive uncertainty of ML models, since local/global surrogate modeling and its applications to design optimization more than often require multiple UQ runs, with each run at a different input sample (e.g., for the iterative refinement of a surrogate or search for a global optimum). A computationally expensive UQ procedure could significantly increase the overhead time for surrogate modeling or design optimization, which may diminish the benefits of using an ML model in engineering design. To enable the wide adoption of UQ for ML in engineering design, the UQ method should be able to not only accurately quantify the predictive uncertainty, but also be very efficient in doing that. The methods presented in Sec. 3.2 and 3.3 have great potential to address this issue.

In summary, UQ of ML models is essential for ML-based engineering design to enable accelerated design optimization and analysis and scale design optimization to large-scale problems. The approaches presented in Sec. 3 could lead to a paradigm shift in various engineering design applications (e.g., materials, energy systems, additive manufacturing, to name a few) in the long term.

Appendix C. UQ of ML models in prognostics

Appendix C.1. Introduction to prognostics

Prognostics aims to predict the future evolution of the health condition of systems, components, or processes based on their current state, the past evolution of the health condition, and the future predicted or planned usage or operating profile [202]. If no additional information on the future usage or operating profile is available, it is often assumed that the system will be operated in the same way as it was operated in the past.

Generally, two different types of data-driven approaches for predicting the RUL can be distinguished [371]:

1. Identifying a health indicator and predicting its trend until a defined threshold is reached.

2. Directly mapping the extracted features or raw measurements as in the case of DL to the RUL.

For the first approach, the focus is on identifying a specific parameter or health indicator that is indicative of the health state of the system or component being monitored. This degradation indicator could be a physical measurement, a derived relevant feature or a combination of several degradation indicators that change over time as the system undergoes degradation. Once the health indicator is identified, the next step is to predict its trend over time. This involves using various predictive modeling techniques, such as regression or time-series analysis, to estimate how the health indicator evolves as the system degrades over time. The goal is to predict when the health indicator will reach a defined threshold, indicating that the system or component is reaching the end of its useful life.

For the second approach, instead of focusing on predicting the trend of a specific health indicator, the predictive model directly maps either the extracted features or, in the case of deep learning, directly from the raw measurements of the system or component to the RUL.

Appendix C.2. Sources of uncertainty in prognostics

In prognostics, there are several sources of uncertainty that can affect the quality of RUL predictions. These uncertainties can originate from diverse factors, and depending on the system, they can impact the RUL prediction to various degrees [47].

While measurement and model uncertainty are common sources of uncertainty in all disciplines and are also encountered in prognostics, some additional challenges for prognostics in terms of uncertainty include the uncertainty of the future usage and operating profiles, the quality and the limited availability of representative time-to-failure trajectories, high variability of operating conditions, and the dependence on external factors and environmental conditions and their impact on system degradation. Moreover, since failure modes and their mechanisms play a crucial role in the evolution of component and system degradation, the precise degradation mechanisms leading to failures may not be fully understood or may involve complex interactions. Such uncertainty in failure modes adds an additional source of uncertainties to the predictions.

Appendix C.3. DL for prognostics

The great advantage brought by DL approaches in the context of prognostics stems from their ability to automatically process high-dimensional, heterogeneous - and often noisy - sensor data in an end-to-end fashion, learn the features automatically and reduce the necessity for hand-crafted feature extraction to the minimum [203]. This concept has given rise to extensive research showcasing the prediction capabilities of modern DL algorithms in the context of prognostics. Nevertheless, most of these approaches are designed to output a single-point estimate of the RUL of the considered industrial or infrastructure assets ([202, 203, 214] and the references therein). This is the case since standard neural networks' outputs are deterministic and are not typically accompanied by a meaningful probabilistic interpretation. This is undesirable in the context of prognostics. Sensor

data are frequently distorted by multiple sources of noise and, training data is often limited in scope and fails to represent the full range of conditions that may arise in real-world scenarios. Consequently, there is a significant risk of encountering high levels of epistemic uncertainties, which must be quantified and communicated to the decision makers.

Appendix C.4. State-of-the-art uncertainty-aware DL approaches for prognostics

The emergence of DNNs has contributed to mitigating the two aforementioned issues, providing a highly expressive class of methods capable of efficiently processing large-scale datasets (see [202, 203, 214] and the references therein). Since standard DL approaches do not naturally incorporate UQ routines, using neural networks in prognostics has come at the price of neglecting UQ, hence providing simple point-estimate predictions as outputs. Only recently, thanks to recent advances in BNNs, more efforts have been spent in designing uncertainty-aware DL techniques for prognostics.

One of the simplest strategies to enable UQ of DNNs is MC dropout. As explained in Section 3.2.3, this method is based on activating dropout layers at inference time, thereby, making the neural network's forward pass stochastic. Thanks to its intuitive rationale and relatively straightforward implementation, it is not surprising that the majority of uncertainty-aware DL methods for prognostics have been established on this strategy in combination with standard neural network architectures, such as fully-connected neural networks [208, 372], CNNs [373–375], and RNNs [376–381]. Engineered systems to which MC dropout has been applied in prognostics include lithium-ion batteries [208, 375, 379], turbofan engines [372, 373, 378, 382], bearings [375, 376], solenoid valves [374], hydraulic mechanisms [377], and circuit breakers [377]. While most of the studies have applied existing MC dropout implementations to prognostics, in [377], the authors propose an adapted framework to model epistemic and aleatory uncertainty by means of MC dropout and a final aleatory layer with two nodes representing the parameters of either a Gaussian or two-parameter Weibull distribution. By appropriately sampling from the weight distribution entailed by the MC dropout and from the output distribution of the final aleatory layer, the authors are able to extract and disentangle epistemic and aleatory uncertainty.

Besides MC dropout, ensemble methods [383–386] and deep Gaussian processes [387, 388] have also been used in prognostics. In particular, in [386], an ensemble of Echo State Networks (ESNs), a type of reservoir computing method, aggregated with an additional ESN on top of the ensemble to estimate the residual variance, is used to predict the RUL and the associated prediction intervals. The model is tested both on toy cases and on real industrial datasets and is shown to yield good performance. In another research study, Deep Gaussian Processes [299, 389], have been employed for the prediction of the RUL on a dataset of turbofan engines [387]. The advantage of these techniques lies in the fact that they combine the probabilistic nature of standard GPR and the expressive power of DNNs. In addition, contrarily to vanilla GPR, they can be applied to the "big-data" regime, which is very common in prognostics. The results show that deep Gaussian processes perform well in the task of RUL prediction, outperforming a number of deep learning baseline methods.
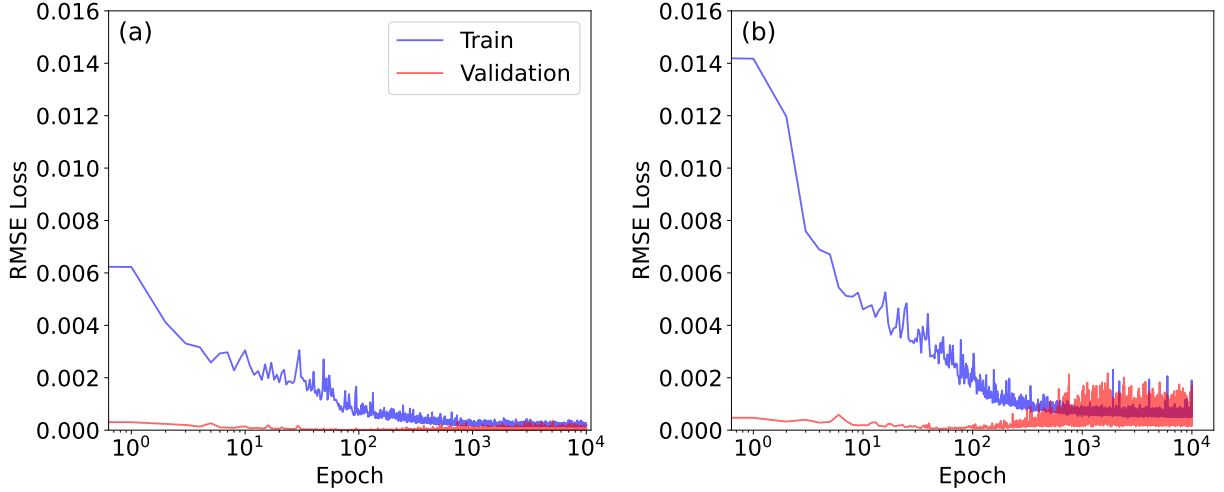
Figure D.28: Training and validation losses for MC dropout models with dropout rate of (a) 0.05 and (b) 0.2 respectively.

Appendix D. Demonstration of Instability of MC Dropout

In Section 3.2.3, we mention the instability of the MC dropout model arising from even slight variations in hyperparameters, such as model size, training epochs and dropout rate. In this appendix, we first show the training and validation losses for two MC dropout models trained with the same data of the toy example from Section 3.5 in Fig. D.28. The two MC dropout models have the same architecture (3 ResNet blocks as shown in Fig. 15), but have different dropout rates. In this case, the MC dropout model converged at around 500 epochs, but no over-fitting is observed until 10000 epochs. Next, we plot the uncertainty maps for various configurations of the MC dropout model in Table D.8. The uncertainty maps are highly inconsistent, thus leading to our conclusion about the instability of MC dropout.

		Number of training epochs	
	200	500	1000

Dropout rate: 0.05

Dropout rate: 0.1

Dropout rate: 0.2

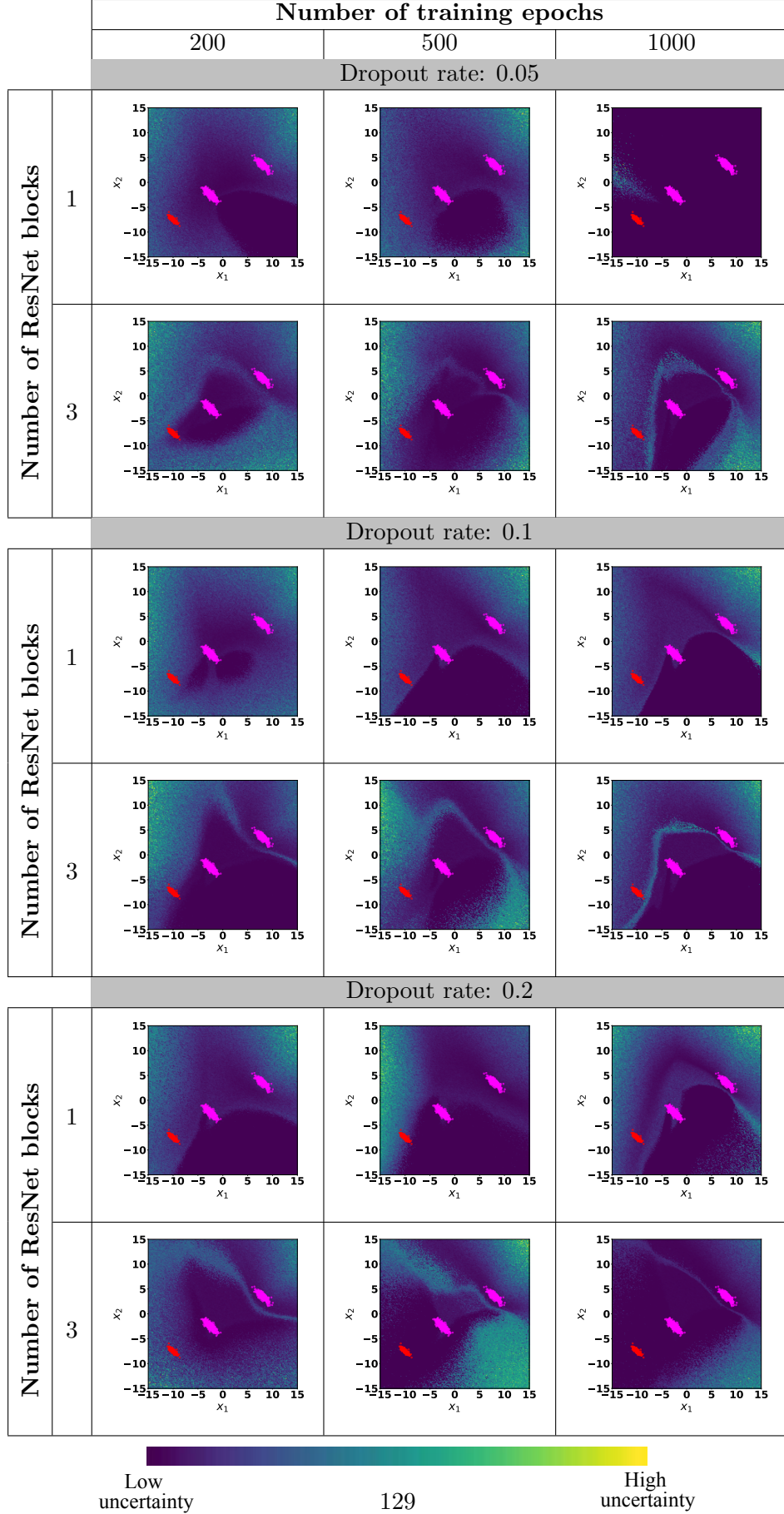Low uncertainty   129   High uncertainty

Table D.8: Demonstration of the instability associated with uncertainty maps of MC dropout with respect to dropout rate, number of training epochs, and ResNet architecture.