Information fusion and machine learning for sensitivity analysis using physics knowledge and experimental data

Berkcan Kapusuzoglu, Sankaran Mahadevan*

Department of Civil and Environmental Engineering, Vanderbilt University, TN 37235

Abstract

When computational models (either physics-based or data-driven) are used for the sensitivity analysis of engineering systems, the sensitivity estimate is affected by the accuracy and uncertainty of the model. If the physics-based computational model is expensive, an inexpensive surrogate model is built and used to compute the Sobol' indices in variance-based global sensitivity analysis (GSA), and the surrogate model introduces further approximation in the sensitivity estimate. This paper considers GSA for situations where both a physics-based model and a small number of experimental observations are available, and investigates strategies to effectively combine the two sources of information in order to maximize the accuracy and minimize the uncertainty of the sensitivity estimate. Physics-informed and hybrid machine learning strategies are proposed to achieve these objectives. Two representative machine learning (ML) techniques are considered, namely, deep neural networks (DNN) and Gaussian process (GP) modeling, and two strategies for incorporating physics knowledge within these ML techniques are investigated, namely: (i) incorporating loss functions in the ML models to enforce physics constraints, and (ii) pre-training and updating the ML model using simulation data and experimental data respectively. Four different models are built for each type (DNN and GP), and the uncertainties in these models are included in the Sobol' indices computation. The DNN-based models, with many degrees of freedom in terms of model parameters and training options, are found to result in smaller bounds on the sensitivity estimates when compared to the GP-based

*Corresponding author

Phone: 1-615-322-3040. Fax: 1-615-322-3365. Postal address: Vanderbilt University, VU Station B 356077, Nashville, TN 37235-6077, United States.

E-mail address: sankaran.mahadevan@vanderbilt.edu (S. Mahadevan).

models. The proposed methods are illustrated for an additive manufacturing example.

Keywords: Global sensitivity analysis, Sobol' index, Deep learning, Physics-informed machine learning, Additive manufacturing, Fused filament fabrication

1. Introduction

Computational models are often used to analyze the response of an engineering system for a variety of input realizations, since conducting experiments to directly measure the true response for many input realizations is often not affordable. However, the computational model is often an incomplete representation of the complex physical system, thus the system response prediction is affected by model uncertainty. In general, the uncertainty sources affecting system response prediction include (a) epistemic uncertainty due to lack of knowledge (arising from either data or model inadequacies), and (b) aleatory uncertainty due to the inherent variability in the system properties or the external inputs. Global sensitivity analysis (GSA) [1] aims to provide a quantitative assessment of the relative contribution of each uncertainty source to the uncertainty in the model response [2–4].

Much of the GSA literature has focused on variability in the inputs and their effects on output variability; the extension of GSA to include epistemic uncertainty sources (data, model) is recent and sparse [5–9]. Model outputs can have uncertainty even for a fixed input when there exists model uncertainty. When the model is computationally expensive, it is often replaced with a surrogate model to facilitate the estimation of Sobol' indices, since such computation requires many input-output samples from the model; the surrogate model introduces additional uncertainty. Several types of surrogate models are used in the literature, e.g., polynomial chaos expansion (PCE), Gaussian process (GP) regression, neural networks, etc., to train a parametric relationship between the inputs and the outputs. The quality and quantity of the training data affect the accuracy of these surrogate models, which directly affects the uncertainty in the model output [7–9]. Thus, it is important to also include the contribution of surrogate model uncertainty to the output uncertainty in GSA. In Le Gratiet et al [7] for example, the Gaussian

process surrogate model uncertainty is included in the Sobol' index estimates with a Monte Carlo (MC) sampling procedure using multiple realizations of the GP model prediction, which helps to construct prediction intervals for the Sobol' index estimates.

In high dimensional problems, even the use of a surrogate model for GSA, which repeatedly executes the code by suppressing some variables and running through the range of other variables, may be computationally demanding since the number of executions of the code increases rapidly with the number of inputs [7, 10–12].

Expanding GSA to consider both aleatory and epistemic uncertainty sources is beneficial in supporting resource allocation decisions. If the contribution of epistemic uncertainty is found to be significant, then it may be valuable to collect more data or refine the physics model to reduce the epistemic uncertainty and thus its contribution to the output uncertainty. Several GSA studies have developed auxiliary variable-based approaches to include both aleatory and epistemic uncertainty sources at a single level instead of using nested simulations, thus achieving both computational efficiency and direct ranking of the different sources of uncertainty to support resource allocation decision-making. The auxiliary variable is used to transform one-to-many input-output mapping to one-to-one mapping, thus facilitating the computation of Sobol' indices for both aleatory and epistemic sources [5]. This idea is expanded in [6] to include several epistemic sources, such as input statistical uncertainty, surrogate model error, physics model discrepancy, and numerical solution error, and to systems with time series inputs and outputs.

Three scenarios of model and data availability can be considered for GSA: (1) use of a physics-based computational model alone, (2) use of available input-output data alone (either from experiments or previous simulations), or (3) use of both physics model and available experimental data. A straightforward model-based approach to estimate Sobol' indices is to use a double-loop Monte Carlo simulation (MCS) [13]. In order to reduce the cost associated with the double-loop MCS, analytical, spectral and efficient sampling-based methods have been developed. The methodology developed by Sudret [14] approximates the original physics model by a PCE and estimates the Sobol' indices by using the PCE coefficients. Chen et al. [15] proposed analytical

formulas to compute Sobol' indices using a GP surrogate model with input variables that follow normal or uniform distributions. The improved FAST method [16] combines the classical FAST method [2] with random balanced design [17] for generating samples to evaluate Sobol' indices.

In some problems, input-output data may be already available instead of having to simulate a physics model expressly for the purpose of GSA. Such data may be available from experiments, or real-world observations, or Markov Chain Monte Carlo (MCMC) sampling during Bayesian model calibration, or MC sampling during reliability analysis, etc. In such cases, data-driven methods have been proposed to directly compute the Sobol' indices based on available input-output samples instead of simulation runs of the physics model. A GSA method based on ANOVA using factorial design of experiments is developed by Ginot et al. [18]. The proposed method results in same values as the Sobol' index since the variance decomposition used in the Sobol' index estimations is same as the one used in the classical ANOVA [19]. The computational cost of most sample-based methods is proportional to the number of model inputs. Li and Mahadevan [20] proposed a modularized method, which has a computational cost that is not proportional to the model input dimension, to estimate the first-order Sobol' indices based on stratification of available input-output samples. DeCarlo et al. [21] proposed an importance sampling approach by introducing weights to different data points to estimate Sobol' indices from available data using Sobol' sequences to reduce the number of simulations; this method computes both first-order and higher order indices, and is able to include correlated inputs. Approximations to the joint probability distribution of inputs and outputs such as multivariate Gaussian, Gaussian copula, and Gaussian mixture have recently been found to give rapid estimation of Sobol' indices [10].

The third scenario is of interest in this paper, where both a physics-based model and some experimental or real-world data are available. One option, if adequate data is available, is to simply build a regression or machine learning (ML) model based on the observation data, and use this model to perform GSA. Multiple recent studies have pursued data-driven ML models in situations where abundant experimental data or real-world observations are available due to advances in modern sensing techniques. Generally, the construction of data-driven ML models

does not require in-depth knowledge of the complex physics inherent in the physical process. ML models can learn complex systems using available observations, but the accuracy of these models depends on the quality and quantity of the data. If the available data is sparse, then the complexity of the process may not be fully captured. Further, since purely data-driven ML models do not explicitly consider physical laws, they can produce physically inconsistent results. In such cases, incorporating physics knowledge within ML models may improve the accuracy and efficiency of GSA computations. The combined use of physics-based and ML models has been shown to achieve more accurate and physically consistent predictions by leveraging the advantages of each method [22–24].

In this work, we incorporate physics knowledge into the ML models to better capture the physics of the process by leveraging physical laws while improving the generalization performance of data-driven models. Two types of strategies are considered for incorporating physics knowledge within ML models: (1) incorporating loss functions in the ML model training to enforce physics constraints, and (2) pre-training the ML model with data generated by the physics model and then updating it with experimental data. Note that the first strategy does not use the physics model but only constraints for the output to obey physical requirements; whereas, the second strategy explicitly uses the physics computational model. Two types of ML models are considered in this paper, namely, Gaussian process (GP) and deep neural network (DNN). These two models are selected in order to represent two different kinds of available ML techniques; the GP model is one of the surrogate models commonly used in uncertainty quantification (UQ) studies, and DNN belongs to the emerging class of deep learning algorithms revolutionizing the field of artificial intelligence, spurred by recent advances in sensing, communication and computational resources. Four different physics-informed machine learning (PIML) models are developed for each type (i.e., GP or DNN) to predict the output quantity of interest (QoI), through combinations of the two strategies. The resulting GSA procedure incorporates the effect of uncertainty in the ML or PIML model, and the various models and strategies are compared in terms of accuracy and uncertainty in the GSA results and their computational demand.

In summary, the contributions of this paper are as follows:

- Physics knowledge and experimental observations are fused in order to maximize the accuracy and minimize the uncertainty of sensitivity estimates.
- Two PIML strategies and their combinations are investigated for global sensitivity analysis using both physics knowledge and experimental data.
- Four different models are built for each of GP and DNN, and the uncertainties in these models are included in the Sobol' indices computation.
- The accuracy, uncertainty and computational effort of different options for the ML and PIML models are evaluated and compared.

The outline of the rest of the paper is as follows. Section 2 provides background information on related methods. Section 3 presents the proposed methodology. A numerical example is presented in Section 4 to illustrate the proposed methodology and draw insights on the performance of various PIML strategies and models. Concluding remarks are provided in Section 5.

2. Background

This section introduces each of the basic techniques used in developing the proposed methodology, namely variance-based GSA, Gaussian process surrogate modeling, and deep neural networks (DNN). These techniques are well established with extensive literature, therefore only a brief introduction is given here.

2.1. Variance-based GSA

Consider a deterministic real integrable one-to-one system response function $\mathrm{Y} = f(\mathbf{X})$, where $f(\cdot)$ is the computational model, $\mathbf{X} = \{\mathrm{X}_1, ..., \mathrm{X}_k\}$ are mutually independent model inputs, and Y is the model output. As shown in [13], the variance of Y can be decomposed as

$$V(\mathrm{Y}) = \sum_{i}^{k} V_i + \sum_{i_1}^{k} \sum_{i_2=i_1+1}^{k} V_{i_1 i_2} + \sum_{i_1}^{k} \sum_{i_2=i_1+1}^{k} \sum_{i_3=i_2+1}^{k} V_{i_1 i_2 i_3} + ... + V_{12...k} \tag{1}$$

where V_i is the variance of Y due to $\mathrm{X_i}$ alone, and $V_{i_1 \ldots i_p} (p \geq 2)$ indicates the variance of Y due to $\{\mathrm{X_{i_1}}, ..., \mathrm{X_{i_p}}\}$.

The Sobol' indices are defined by dividing both sides of Eq. (1) with $V(\mathrm{Y})$

$$1 = \sum_{i}^{k} S_i + \sum_{i_1}^{k} \sum_{i_2=i_1+1}^{k} S_{i_1 i_2} + \sum_{i_1}^{k} \sum_{i_2=i_1+1}^{k} \sum_{i_3=i_2+1}^{k} S_{i_1 i_2 i_3} + ... + S_{12 \ldots k} \tag{2}$$

where S_i is the first-order or main effects index that assesses the contribution of $\mathrm{X_i}$ individually to the variance of the output Y without considering interactions with other inputs. The higher-order indices $S_{i_1 \ldots i_p} (p \geq 2)$ in Eq. (2) measure the contributions of individual plus interactive effects of $\{\mathrm{X_{i_1}}, ..., \mathrm{X_{i_p}}\}$.

The first-order index S_i is defined as follows:

$$S_i = \frac{V_i}{V(\mathrm{Y})} = \frac{V_{X_i}(E_{\mathbf{X}_{-i}}(\mathrm{Y}|\mathrm{X}_i))}{V(\mathrm{Y})} \tag{3}$$

where $\mathbf{X}_{-i}$ are all the model inputs other than X_i.

The overall contribution of $\mathrm{X_i}$ considering an individual input and its interactions with all other inputs is measured by the total effects index S_i^T:

$$S_i^T = 1 - \frac{V_{-i}}{V(\mathrm{Y})} = \frac{V_{\mathbf{X}_{-i}}(E_{\mathrm{x}_i}(\mathrm{Y}|\mathbf{X}_{-i}))}{V(\mathrm{Y})}. \tag{4}$$

The computation of S_i analytically is nontrivial since $E_{\mathbf{X}_{-i}}(\cdot)$ requires multi-dimensional integrals. A basic sampling-based approach is to use double-loop sampling [13]. Several approaches to reduce the computational cost were mentioned in Section 1. One of these approaches of particular relevance to this paper is to replace the original computational model $f(\cdot)$ by a surrogate model and use this surrogate model in GSA [25–29]. This approach will be addressed further in Section 3.

2.2. Gaussian process surrogate modeling

The GP surrogate model provides a prediction $f(\mathbf{u})$ at a given input $\mathbf{u}$ as

$$f(\mathbf{u}) = \mathbf{h}(\mathbf{u})^T \boldsymbol{\beta} + z(\mathbf{u}) \tag{5}$$

where $\mathbf{h}(\cdot)$ is the trend function, $\boldsymbol{\beta}$ is the vector of trend coefficients, and $z(\cdot)$ is a random variable from a zero mean GP with covariance function, which describes the deviation of the model from the trend. The covariance between the outputs of the Gaussian process $Z(\cdot)$ at points $\mathbf{a}$ and $\mathbf{b}$ is defined as:

$$\mathrm{Cov}[Z(\mathbf{a}), Z(\mathbf{b})] = \sigma_Z^2 R(\mathbf{a}, \mathbf{b}) \tag{6}$$

where σ_Z^2 is the process variance and $R(\cdot, \cdot)$ is the correlation function. A squared exponential correlation function with separate length scale parameters l_i for each input dimension has often been used in the literature:

$$R(\mathbf{a}, \mathbf{b}) = \exp\left[-\sum_{i=1}^{\mathrm{M}} \frac{(a_i - b_i)^2}{l_i} \right] \tag{7}$$

The hyperparameters of the GP model, i.e., $\boldsymbol{\Theta} = \{l, \sigma_Z, \sigma_{obs}\}$, where σ_{obs} is the observation error, are inferred from the training data. A common method is to maximize the log marginal likelihood function, which is defined as

$$\log\, p(\mathbf{Y}|\mathbf{X}; \boldsymbol{\Theta}) = -\frac{1}{2}\mathbf{Y}(\mathbf{K}_{TT} + \sigma_{obs}^2 \mathbf{I})^{-1}\mathbf{Y} - \frac{1}{2}\log|\mathbf{K}_{TT} + \sigma_{obs}^2 \mathbf{I}| + \frac{n}{2}\log 2\pi. \tag{8}$$

The outputs of the GP model are the mean prediction $\mu_G(\cdot)$ and the variance of the prediction $\sigma_G^2(\cdot)$, defined as:

$$\mu_G(\mathbf{u}) = \mathbf{h}(\mathbf{u})^T \boldsymbol{\beta} + \mathbf{r}(\mathbf{u})^T \mathbf{R}^{-1}(\mathbf{g} - \mathbf{F}\boldsymbol{\beta}) \tag{9}$$

$$\sigma_G^2(\mathbf{u}) = \sigma_Z^2 - \mathbf{A} \begin{bmatrix} \mathbf{0} & \mathbf{F}^T \\ \mathbf{F} & \mathbf{R} \end{bmatrix}^{-1} \mathbf{A}^T \tag{10}$$

where $\mathbf{r}(\mathbf{u})$ is a vector containing the covariance between $\mathbf{u}$ and each of the training points $\{x_1, x_2, ..., x_n\}$, $i \in \{1, ..., n\}$, $\mathbf{R}$ is an $n \times n$ matrix containing the correlation between each pair of training points, $\mathbf{R}(x_i,\ x_j) = \mathrm{Cov}[Z(x_i), Z(x_j)]$; $\mathbf{g}$ is the vector of original physics model outputs at each of the training points, $\mathbf{F}$ is a $n \times q$ matrix with rows $\mathbf{h}(\mathbf{u}_i)^T$, and $\mathbf{A} = \left[\mathbf{h}(\mathbf{u})^T\ \mathbf{r}(\mathbf{u})^T\right]$.

2.3. Deep neural networks

In recent years, due to the confluence of advanced sensing and imaging techniques, big data processing techniques, enormous computational power and the internet, rapid advances are being made in developing sophisticated data-driven machine learning models, particularly neural networks. A deep neural network (DNN) is composed of multiple hidden layers and has four major components: neuron, activation function, cost function, and optimization. Figure 1 shows a neural network consisting of three inputs, two hidden layers, each having four neurons, and two output neurons. The values of various input variables of a particular neuron are multiplied by their associated weights, then the sum of the products of the neuron weights and the inputs are calculated at each neuron. The summed value is passed through an activation function that maps the summed value to a fixed range before passing these signals on to the next layer of neurons.

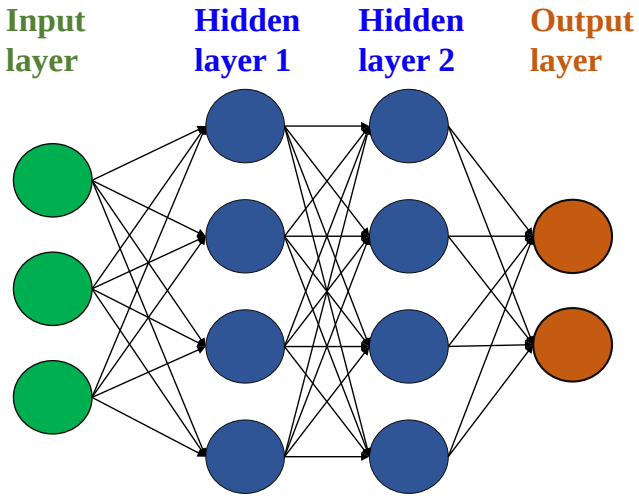

Fig. 1. A deep neural network with two hidden layers.

The predictions of the DNN after a forward propagation, $\hat{\mathbf{Y}}$, are compared against the true response of the system, $\mathbf{Y}$, by defining a loss function (e.g., root mean squared error (RMSE);

$\mathcal{L}_{\text{RMSE}}(\mathbf{Y}, \hat{\mathbf{Y}}) = \sqrt{\sum_{i=1}^{n}(\mathrm{y}_i - \hat{\mathrm{y}}_i)^2/n})$, which measures how far off the predictions are from the observations for the n training samples. Backpropagation algorithms are employed to keep track of small perturbations to the weights that affect the error in the output and to distribute this error back through the network layers by computing gradients for each layer using the chain rule. In order to minimize the value of the loss function, necessary adjustments are applied at each iteration to the neuron weights in each layer of the network. These procedures are performed at each iteration until the loss function converges to a stable value.

3. Proposed methodology

The proposed methodology for sensitivity analysis, using both physics knowledge and experimental data, consists of the following steps:

1. Identification of PIML strategies
2. Implementation of PIML strategies in ML models
3. Variance quantification in ML model prediction
4. Sobol' indices computation with ML model prediction variance

The following subsections describe these steps in detail.

3.1. Identification of PIML strategies

PIML models seek to incorporate physics knowledge or constraints within the data-driven ML models. When a mechanistic, physics-based model is also available, complementary strengths of both mechanistic and ML models can be leveraged in a synergistic manner [24]. In the latter case, the aim is to improve the predictions beyond that of physics-based models or ML models alone by coupling physics-based models with ML models. Thus two different strategies to combine physics knowledge and ML models can be considered: (1) incorporate physics constraints in the ML models, and (2) pre-train and update the ML models using physics model input-output and experimental data, respectively.

3.1.1. Strategy 1: Enforcing physics constraints

A direct strategy to enforce physics constraints in ML model predictions is by including the constraints within the loss function used in training the ML model [22]. Thus, while training a PIML model with inputs $\boldsymbol{X}$ and outputs $\hat{\boldsymbol{Y}}$, the physical constraints can be incorporated as additional penalty terms in the loss function:

$$\mathcal{L} = \mathcal{L}_{\text{ML}} + \lambda_{\text{phy}} \mathcal{L}_{\text{phy}}(\hat{\boldsymbol{Y}}), \tag{11}$$

where $\mathcal{L}_{\text{ML}}$ is the log marginal likelihood of the data for a GP model:

$$\mathcal{L}_{\text{GP}} = \log\, p(\mathbf{Y}|\mathbf{X};\boldsymbol{\Theta}) \tag{12}$$

and training loss function for a DNN that evaluates a supervised error, e.g., root mean squared error (RMSE):

$$\mathcal{L}_{\text{DNN}}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \sqrt{\sum_{i=1}^{n} \frac{(Y_i - \hat{Y}_i)^2}{n}} \tag{13}$$

which measures the accuracy of predictions $\hat{\boldsymbol{Y}}$ for n training samples. (Note that for the GP model, the likelihood is maximized, whereas for the DNN model, the RMSE is minimized). The additional physics constraint loss function $\mathcal{L}_{\text{phy}}$ in the second term of Eq. 11 is weighted by a hyperparameter λ_{phy}; the value of λ_{phy} controls the strength of the physics constraint enforcement. The inclusion of the second term ensures physically consistent model predictions and helps to reduce the generalization error, which is a measure of how accurately a model is able to predict the output QoI for previously unseen data [22].

The physical inconsistencies in the model predictions are evaluated using the physics constraint loss term. The generic forms of these physical relationships can be expressed using the

following constraints:

$$\mathcal{F}_1(\hat{\boldsymbol{Y}}, \boldsymbol{\Gamma}) = 0, \tag{14}$$
$$\mathcal{F}_2(\hat{\boldsymbol{Y}}, \boldsymbol{\Gamma}) \leq 0.$$

where $\boldsymbol{\Gamma}$ denotes other variables or thresholds that define the physics constraint regarding the model output $\hat{\boldsymbol{Y}}$. Equation. 14 indicates that the constraints may take the form of equalities or inequalities. These equations can involve algebraic relationships or partial differentials of $\hat{\boldsymbol{Y}}$ and/or $\boldsymbol{\Gamma}$. The physics-based loss functions for these equations can be defined as:

$$\mathcal{L}_{\text{phy}}(\hat{\boldsymbol{Y}}) = ||\mathcal{F}_1(\hat{\boldsymbol{Y}}, \boldsymbol{\Gamma})|| + \text{ReLU}(\mathcal{F}_2(\hat{\boldsymbol{Y}}, \boldsymbol{\Gamma})), \tag{15}$$

where $\text{ReLU(x)} = \max(0, \text{x})$ represents the rectified linear unit function and it acquires value when a threshold is violated in the inequality constraint, i.e., it penalizes the optimization when $\mathcal{F}_2 > 0$. It can also be used to penalize deviations from a desired physically consistent relationship among multiple outputs $\hat{\boldsymbol{Y}}$ [30].

3.1.2. Strategy 2: Pre-training and Updating

The ML model output accuracy and uncertainty are dependent on the quality and quantity of the available training data. In some systems, the high cost associated with conducting experiments makes it infeasible to have adequate amount of training data to build purely data-driven models. Thus, it may be desirable to combine the physics-based model and available experimental data in seeking to maximize the accuracy and minimize the uncertainty of the sensitivity estimates. When the experiments are expensive, they can only be conducted for a few values of the inputs, whereas it might be possible to run the physics-based model for a larger set of input values. In that case, the simulation data can be used to pre-train an ML model, which is used as the initial model to be updated with experimental observations. Further, training of ML models requires the choice of initial values of the model parameters. The transfer of physical knowledge using a pre-trained ML model can prevent poor initialization due to lack

of knowledge regarding the initial choice of ML model parameters prior to training.

Since the pre-training based on the physics-based model can use a large amount of training data (with multiple input parameter combinations) over a wide range of values, the pre-training may also help the eventual ML model to have wider generalization beyond experimental data. In the numerical example in Section 4, the pre-training strategy exercises the physics model over 1310 input combinations, whereas only 39 experiments are available. However, if the physics model is computationally expensive, then the advantage of the pre-training strategy in using a larger input data set (for physics model runs) compared to the experiments becomes limited.

The two proposed strategies to predict the QoI are shown in Fig. 2. Figure 2(a) shows the first method, where the physical knowledge is included through constraints within the loss function of an ML trained with only experimental data. Figure 2(b) shows the second method, where an ML model is first trained with data generated using the physics-based model and then updated using experimental data. Figures 2(c) and 2(d) show the trained ML model predictions ($\hat{\boldsymbol{Y}}$) for the two proposed strategies, respectively. The proposed PIML strategies can be applied to any physical system by leveraging the related physical constraints or physics-based models.

3.2. Implementation of PIML strategies in ML models

Based on the proposed two strategies to incorporate physics knowledge into the ML model, four separate ML models can be constructed for each type of surrogate model considered here (i.e., GP and DNN):

1. GP
2. $\text{GP}^{\mathcal{L}_{\text{phy}}}$
3. GP^{upd}
4. $\text{GP}^{\text{upd},\mathcal{L}_{\text{phy}}}$
5. DNN
6. $\text{DNN}^{\mathcal{L}_{\text{phy}}}$
7. DNN^{upd}
8. $\text{DNN}^{\text{upd},\mathcal{L}_{\text{phy}}}$

These different models cover the following options: model trained with experimental data alone, models trained with PIML strategies 1 or 2 alone, and models trained with both PIML strategies together. The implementations of PIML strategies 1, 2, and their combination are

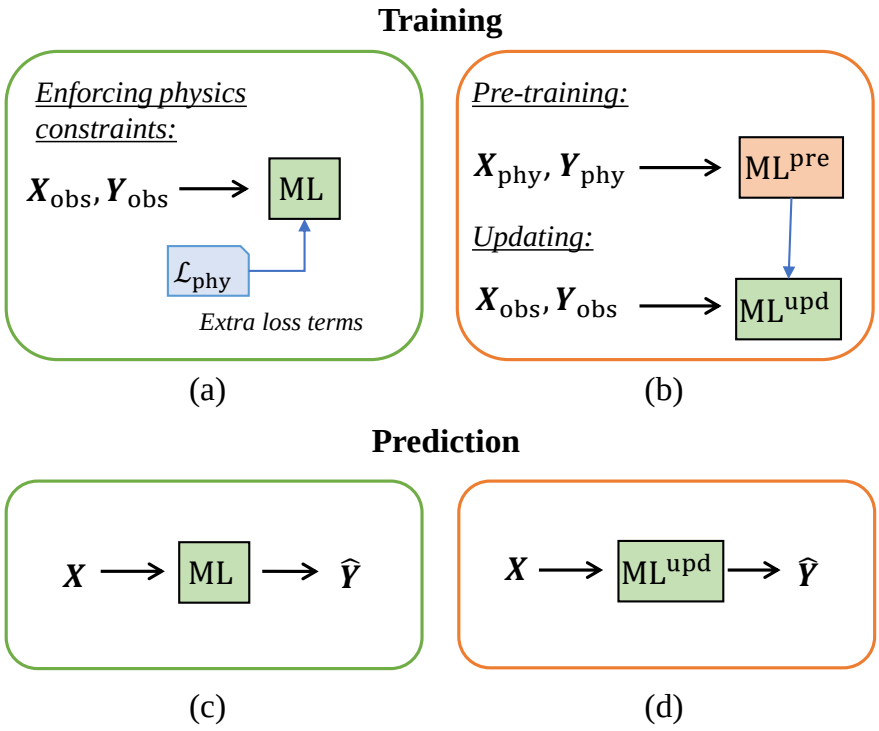

Fig. 2. PIML strategies: (a) incorporating physics-based loss functions in the ML models to enforce physics constraints, (b) pre-training an ML model with physics model input-output ($\mathbf{X}_{\text{phy}}$, $\mathbf{Y}_{\text{phy}}$) and updating it with experimental training data ($\mathbf{X}_{\text{obs}}$, $\mathbf{Y}_{\text{obs}}$), (c) the trained ML model prediction ($\hat{\boldsymbol{Y}}$), (d) updated ML model prediction ($\hat{\boldsymbol{Y}}$).

different for the GP models vs. the DNN models. The following subsections describe how the PIML strategies can be implemented for each of the above models.

3.2.1. Implementation of PIML in GP models

In Model 1, denoted as GP, only experimental observations are used for training. The hyperparameters of the GP model (process variance, correlation length scale along each input dimension, and trend function coefficients, and also measurement error variance if unknown) are optimized during training by maximizing the log marginal likelihood function shown in Eq. 8. In calculating the likelihood, the difference between the true response of the system $\mathbf{Y}_{\text{true}}$ and the observed response $\mathbf{Y}_{\text{obs}}$ is attributed to the observation error ϵ_{obs}, which is often treated as a zero-mean Gaussian random variable with variance σ^2_{obs}.

Model 2, denoted as $\text{GP}^{\mathcal{L}_{\text{phy}}}$, incorporates the first PIML strategy by enforcing physics constraints during the optimization of the GP model hyperparameters. More specifically, the physics

constraints are included during the maximization of the log marginal likelihood function (Eq. 8) while inferring the hyperparameters of the GP model; thus constrained optimization needs to be implemented. Specifically, the training of Model 2 is achieved by maximizing the function Eq. 16:

$$\mathcal{L}_{\text{GP}} = \log\, p(\mathbf{Y}|\mathbf{X};\boldsymbol{\Theta}) - \lambda_{\text{phy}}\mathcal{L}_{\text{phy}}(\hat{\boldsymbol{Y}}), \tag{16}$$

where $\hat{\boldsymbol{Y}}$ is the GP model prediction. Note that since $\mathcal{L}_{\text{GP}}$ is to be maximized, the second term corresponding to the physics constraint has a negative sign.

Model 3, denoted as GP^{upd}, pursues the second PIML strategy, i.e., it pre-trains a GP surrogate model with data generated from the physics model, then improves the surrogate using experimental data. Consider a physics model $G(\cdot)$ that maps input variables $\mathbf{X}$ and model parameters $\boldsymbol{\theta_m}$ to the numerical model output $\mathbf{Y}_m$:

$$\mathbf{Y}_m = G\big(\mathbf{X};\ \boldsymbol{\theta}_m(\mathbf{X})\big). \tag{17}$$

Let n_D be the number of collected observation data $\mathbf{Y}_{\text{obs}}$ from experiments with input variable settings $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n_D)}$, where $\mathbf{x}^{(i)}$ is the input variable setting for the ith experiment. The physics model prediction is inaccurate due to missing physics or due to other approximations. Thus, a model discrepancy term $\boldsymbol{\delta}(\mathbf{X})$ as a function of model inputs is introduced to capture the difference between $\mathbf{Y}_{\text{true}}$ and $\mathbf{Y}_m$ [31]. The true system response $\mathbf{Y}_{\text{true}}$ can be described as

$$\mathbf{Y}_{\text{true}}(\mathbf{X}) = \mathbf{Y}_{\text{obs}}(\mathbf{X}) + \epsilon_{\text{obs}}(\mathbf{X}) = \mathbf{Y}_m(\mathbf{X}) + \boldsymbol{\delta}(\mathbf{X}) = G\big(\mathbf{X};\ \boldsymbol{\theta}_m(\mathbf{X})\big) + \boldsymbol{\delta}(\mathbf{X}). \tag{18}$$

When the physics model is computationally expensive, it is replaced by a cheaper surrogate model. In Model 3, a GP surrogate model is used to approximate the original physics model. The accuracy of the surrogate model prediction depends on the quality and quantity of the training data generated by the original physics model. The surrogate model error ($\epsilon_\delta(\mathbf{X})$) can

be incorporated as follows:

$$\mathbf{Y}_m(\mathbf{X}) = \hat{\mathbf{Y}}_m(\mathbf{X}) + \epsilon_\delta(\mathbf{X}), \tag{19}$$

where $\hat{\mathbf{Y}}_m$ is the surrogate model prediction.

A common approach to estimate the discrepancy term $\boldsymbol{\delta}(\mathbf{X})$ is the one formulated by Kennedy and O'Hagan [31], which is applicable in the context of Bayesian calibration. In that case, physics model parameters are sought to be calibrated, and a discrepancy term is added in the calibration equation. The discrepancy term can be expressed in multiple ways, such as constant, Gaussian random variable with unknown parameters (either input-dependent or not), or Gaussian process (either stationary or non-stationary) [32]. The hyperparameters of the discrepancy term are then estimated along with the physics model parameters using Bayesian calibration.

However, the situation considered here is much simpler. There is no calibration of the physics model parameters here; only the discrepancy term is needed. (In other words, the physics model parameters are already established). In that case, the model discrepancy can be evaluated for different input values of experimental tests and realizations of observation errors as follows:

$$\boldsymbol{\delta}(\mathbf{X}) = \mathbf{Y}_{\text{obs}}(\mathbf{X}) + \epsilon_{\text{obs}}(\mathbf{X}) - \hat{\mathbf{Y}}_m(\mathbf{X}) - \epsilon_\delta(\mathbf{X}). \tag{20}$$

Moving the surrogate model error $\epsilon_\delta(\mathbf{X})$ to the left hand side, we can express the difference between the actual response and GP model prediction as

$$\hat{\boldsymbol{\delta}}(\mathbf{X}) = \boldsymbol{\delta}(\mathbf{X}) + \epsilon_\delta(\mathbf{X}) = \mathbf{Y}_{\text{obs}}(\mathbf{X}) + \epsilon_{\text{obs}}(\mathbf{X}) - \hat{\mathbf{Y}}_m(\mathbf{X}). \tag{21}$$

In this work, a second GP model is trained for $\hat{\boldsymbol{\delta}}(\mathbf{X})$ in terms of the inputs. Thus two GP models are trained in Model 3. The first GP model is constructed using the physics model input-output data, and predicts $\hat{\mathbf{Y}}_m$. The second GP model is constructed using the experimental data and the corresponding surrogate model predictions, and predicts $\hat{\boldsymbol{\delta}}$ (the difference between the surrogate model prediction and actual system response).

The GP model for the model discrepancy captures the combined contribution of measurement

error, physics and surrogate model errors for a given prediction. Thus, the predictions of the first GP model (pre-trained) are corrected with the second GP model predictions ($\hat{\boldsymbol{\delta}}$) representing the model discrepancy term and can be written as

$$\hat{\mathbf{Y}}(\mathbf{X}) = \hat{\mathbf{Y}}_m(\mathbf{X}) + \hat{\boldsymbol{\delta}}. \tag{22}$$

Model 4, denoted as $\text{GP}^{\text{upd},\mathcal{L}_{\text{phy}}}$, combines both PIML strategies. The pre-trained model is corrected with $\hat{\boldsymbol{\delta}}$ and the physics constraints are enforced within the correction. The first GP model is trained using the input-output samples from the physics model to predict $\hat{\mathbf{Y}}_m$. Then, using the experimental data, the second GP model is constructed by enforcing the physics constraints during the optimization of its hyperparameters. Thus constrained optimization is implemented while estimating the hyperparameters of the second GP model, i.e., the constraint is added to the objective through a penalty term as in Eq. 16. The prediction of the updated model is $\hat{\mathbf{Y}}$, given by the sum of the predictions of the two GP models, as shown in Eq. 22.

3.2.2. Implementation of PIML in DNN models

In Model 5, denoted simply as DNN, a deep neural network is trained using only experimental data. In order to train the model, an optimization algorithm is used to find a set of model parameters (weights and biases) that best map inputs to outputs. The number of epochs, which is the number of complete passes through a batch of training dataset, layers and neurons needs to be optimized to reduce cost and improve model accuracy. The model accuracy is evaluated using a validation dataset not used for training; the number of epochs is gradually increased until the accuracy improvement is insignificant. The accuracy and generalization performance of the DNN are also affected by the dropout rate used in training the model; the dropout concept will be discussed in Section 3.3 below, and the selection of the dropout rate will be discussed in Section 4.

Model 6, denoted as $\text{DNN}^{\mathcal{L}_{\text{phy}}}$, extends Model 5 by implementing the first PIML strategy, i.e., physical knowledge related to the physical process is enforced through constraints within the loss function of the DNN, as shown in Eq. 11. The physics-based loss function terms are

evaluated for given experimental inputs at every optimization iteration during the training of $\text{DNN}^{\mathcal{L}_{\text{phy}}}$; this makes the optimization process slower during the training. In particular, the multipliers in the physics constraint penalty terms ($\lambda_{\text{phy}}^{\text{DNN}}$) affect the training speed; a higher value of the multiplier makes the penalty stronger (i.e., the physics constraints more stringent), thus requiring an increased number of iterations to converge to the optimum solution.

Model 7, denoted as DNN^{upd}, pursues the second PIML strategy, where a DNN model is pre-trained using the coupled multi-physics model input-output and then updated with experimental data. The pre-trained model is first trained with physics model input-output data consisting of input combinations over a range of values. Then, the weights and biases (model parameters) of this pre-trained network are updated using the experimental data. Note that the model parameters in this case are updated using a least squares approach, whereas in the GP approach (i.e., in Model 3 (GP^{upd}) and Model 4 ($\text{GP}^{\text{upd},\mathcal{L}_{\text{phy}}}$)), a second GP model for the model error $\hat{\boldsymbol{\delta}}$ is constructed using the experimental data and added to the first GP model trained with physics model input-output. Thus the second PIML strategy is implemented differently in the GP and DNN models.

Model 8, denoted as $\text{DNN}^{\text{upd},\mathcal{L}_{\text{phy}}}$, combines both PIML strategies for DNN, where the optimized model parameters of the pre-trained model based on the physics model input-output are used as the initial values. These model parameters are updated using the experimental data by minimizing the augmented loss function shown in Eq. 13 which consists of both the training loss function and the physics constraint loss terms. Similar to Model 6, the inclusion of physics constraints makes it slower for the optimization to converge to optimal model parameter values.

3.3. Variance of GP and DNN prediction

In general, every surrogate model has uncertainty in prediction, whether acknowledged or not. In the GP models, the prediction at a given input is expressed by a normal distribution with a mean and variance. In order to quantify the uncertainty in the GP prediction, we can sample multiple realizations of the Gaussian process term. Note that this only captures the variance of the GP prediction, not the bias, which can be evaluated by comparing against validation data.

In the DNN models, the estimates of the model parameters (neuron weights $\mathbf{w}$) have uncertainty, and this uncertainty depends on the available training data. When the neural network parameters are represented using distributions (to reflect the epistemic uncertainty) instead of deterministic values, the model is referred to as a Bayesian neural network (BNN) [33–35]. In this Bayesian context, the model parameter uncertainty is first described using a prior distribution $p(\mathbf{w})$, and the likelihood function is $p(\mathbf{Y}|\mathbf{X}, \mathbf{w})$. Following Bayes' theorem, a posterior distribution over the model parameters given the training data $\{\mathbf{X}, \mathbf{Y}\} = \{\{\mathbf{x}_1, ..., \mathbf{x}_N\}, \{\{\mathbf{y}_1, ..., \mathbf{y}_N\}\}$ is defined by

$$p(\mathbf{w}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{Y}|\mathbf{X})}. \tag{23}$$

In this context, the predictive distribution of the model outputs for a given input $\mathbf{x}^*$ is given by:

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int_{\Omega} p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w})p(\mathbf{w}|\mathbf{X}, \mathbf{Y})d\mathbf{w}. \tag{24}$$

The posterior distribution of model parameters $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ is challenging to evaluate over the entire parameter space Ω due to the high dimensionality of Ω in a DNN model, and the highly non-linear behavior caused by the non-linear activation functions and their combinations across multiple hidden layers. Therefore, different approximate inference techniques can be considered to infer the posterior distribution $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ [36–39]. One such approximation is variational inference, which fits a simple and tractable distribution $q_\theta(\mathbf{w})$ to the posterior, parametrized by a variational parameter θ [36]. This approximates the intractable problem by optimizing the parameters of $q_\theta(\mathbf{w})$. The quality of the variational inference can be assessed by the Kullback-Leibler (KL) divergence between the approximate distribution $q_\theta(\mathbf{w})$ and the true model posterior $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$.

The term *dropout* refers to randomly dropping out neurons (along with their connections) with a given dropout rate during the training phase in a neural network. Dropout is a common regularization approach in neural network training, which prevents over-fitting and reduces generalization error. A Monte Carlo (MC) dropout technique has been developed in recent years in the context of Bayesian neural networks [40], which has been shown to be equivalent

to performing approximate variational inference. In MC dropout, dropout is not only applied while training a model but also during prediction. Randomly chosen neurons are temporarily removed from the network along with their connections. Next, the gradients of neuron weights are calculated on a sub-neural network for each training data and these gradients are then averaged over the training sets to obtain the weights for the overall network. A Bayesian neural network with MC dropout generates random samples following a binomial distribution (0 or 1) for each neuron in the input and hidden layers during prediction. The neuron that takes the value 0 is dropped with probability p_d. The outputs of the network are predicted using the collection of generated random samples from the posterior predictive distribution and the uncertainty in the prediction of a new data is quantified with the trained network. Thus the MC dropout strategy provides an efficient way of Bayesian inference to quantify the model prediction variance, and can be applied to a variety of neural networks, such as feedforward neural networks, convolutional neural networks, and recurrent neural networks [41].

The sensitivity estimate results depend on the dropout rate. The main reason for this is that the model is regularized and it underfits the data as it is over-regularized. Further, both the accuracy and uncertainty of the sensitivity estimates also depend on the number of training epochs, and the architecture of the network. If the model is not fully trained, it will also result in underfitting, leading to larger bias and variance in the prediction.

3.4. Sobol' indices computation with model uncertainty

This section discusses the incorporation of ML model prediction variance within the estimation of Sobol' indices using the GP and DNN models.

When the training data is noise-free, the GP predictions at the training points have zero variance and at other points the variance is non-zero. The prediction at any point is given by a normal distribution with a mean and variance. This prediction uncertainty can be captured by sampling multiple realizations of the GP model, which can then be used in GSA. The model uncertainty pertaining to the GP model is propagated to the Sobol' index calculations using the

following estimator (see [7]):

$$
\begin{aligned}
S_m^{\mathrm{GP}} &= \frac{V_{X_i}(E_{\mathbf{X}}(\mathbf{Y}(Z(\mathbf{X}))|X_i))}{V(\mathbf{Y}(Z(\mathbf{X})))} \\
&= \frac{\frac{1}{m}\sum_{k=1}^{m}\hat{\mathbf{Y}}(Z(\mathbf{X}^{(k)}))\hat{\mathbf{Y}}(Z(X_i^{(k)})) - \frac{1}{m}\sum_{k=1}^{m}\hat{\mathbf{Y}}(Z(\mathbf{X}^{(k)}))\sum_{k=1}^{m}\hat{\mathbf{Y}}(Z(X_i^{(k)}))}{\frac{1}{m}\sum_{k=1}^{m}\hat{\mathbf{Y}}(Z(\mathbf{X}^{(k)}))^2 - (\frac{1}{m}\sum_{k=1}^{m}\hat{\mathbf{Y}}(Z(\mathbf{X}^{(k)})))^2},
\end{aligned}
\tag{25}
$$

where Z represents the ensemble of realizations from a Gaussian process (trained with n experimental data) and $\mathbf{X}^{(k)}$ and $X_i^{(k)}$ are the kth Monte Carlo samples of the random vectors $\mathbf{X}$ and $\mathbf{X}_i$.

The distribution of S_m^{GP} can be computed by sampling N_Z realizations of Z numerically using Algorithm 1.

Algorithm 1 Estimation of the distribution of S_m^{GP} using GP models.

1: Generate samples $\mathbf{X}^{(k)}$ and $X_i^{(k)}$ $(k = 1, \ldots, m)$ of the random vectors $\mathbf{X}$ and $\mathbf{X}_i$.
2: **for** $p = 1, 2, \ldots, N_Z$ **do**
3: Sample a realization $z(\mathbf{x})$ of $Z(\mathbf{x})$ with $\mathbf{x} = \{(x^{(k)})_{k=1,\ldots,m}, (x_i^{(k)})_{k=1,\ldots,m}\}$.
4: Calculate the model prediction $\hat{\mathbf{Y}}(z)$.
5: Compute $\hat{S}_{m,p}^{\mathrm{GP}}$ using Eq. 25.
6: **end for**
return $(\hat{S}_{m,p}^{\mathrm{GP}})_{p=1,2,\ldots,N_Z}$.

The output of Algorithm 1 $(\hat{S}_{m,p}^{\mathrm{GP}})_{p=1,2,\ldots,N_Z}$ is a sample of size N_Z, where m is the number of Monte Carlo samples. Thus, the mean and variance of sensitivity estimates obtained using the GP models are defined as follows, respectively:

$$
\begin{aligned}
\mu_{S_m^{\mathrm{GP}}} &= \frac{1}{N_Z}\sum_{p=1}^{N_Z}\hat{S}_{m,p}^{\mathrm{GP}}, \\
\sigma^2_{S_m^{\mathrm{GP}}} &= \frac{1}{N_Z}\sum_{p=1}^{N_Z}(\hat{S}_{m,p}^{\mathrm{GP}} - \mu_{S_m^{\mathrm{GP}}})^2.
\end{aligned}
\tag{26}
$$

A similar approach can be implemented in the DNN models with the use of MC dropout (with

a chosen dropout rate). However, in contrast to the GP models, the sampling implementation is different in the DNN models. In the DNN models, we randomly set units of the network to zero and generate predictions using the remaining units of the network as shown in Algorithm 2. Whereas, we sample from a multivariate normal distribution in GP models to quantify the uncertainty in the Sobol' index estimates.

Similar to Eq. 25, the uncertainty in the DNN model can be propagated to the sensitivity calculations using the following estimator:

$$S_m^{\text{DNN}} = \frac{\frac{1}{m}\sum_{k=1}^{m} \hat{\mathbf{Y}}(\mathcal{G}(\mathbf{X}^k))\hat{\mathbf{Y}}(\mathcal{G}(X_i^k)) - \frac{1}{m}\sum_{k=1}^{m} \hat{\mathbf{Y}}(\mathcal{G}(\mathbf{X}^k)) \sum_{k=1}^{m} \hat{\mathbf{Y}}(\mathcal{G}(X_i^k))}{\frac{1}{m}\sum_{k=1}^{m} \hat{\mathbf{Y}}(\mathcal{G}(\mathbf{X}^k))^2 - (\frac{1}{m}\sum_{k=1}^{m} \hat{\mathbf{Y}}(\mathcal{G}(\mathbf{X}^k)))^2}, \tag{27}$$

where $\mathcal{G}$ denotes the deep neural network (DNN) with MC dropout and all the other terms have the same definition as Eq. 25..

Algorithm 2 Estimation of the distribution of S_m^{DNN} using DNN models with MC dropout.

1: Generate samples $\mathbf{X}^{(k)}$ and $X_i^{(k)}$ $(k = 1, \ldots, m)$ of the random vectors $\mathbf{X}$ and $\mathbf{X}_i$.
2: **for** $p = 1, 2, \ldots, N_d$ **do**
3: Perform a stochastic forward pass through the network $\mathcal{G}(\mathbf{X})$ using MC dropout.
4: Calculate the model prediction $\hat{\mathbf{Y}}(\mathcal{G})$.
5: Compute $\hat{S}_{m,p}^{\text{DNN}}$ using Eq. 27.
6: **end for**
return $(\hat{S}_{m,p}^{\text{DNN}})_{p=1,2,\ldots,N_d}$.

The output of Algorithm 2 $(\hat{S}_{m,p}^{\text{DNN}})_{p=1,2,\ldots,N_d}$ is a sample of size N_d. Thus, the mean and variance of sensitivity estimates obtained using DNN models are defined as follows, respectively:

$$\begin{aligned} \mu_{S_m^{\text{DNN}}} &= \frac{1}{N_d}\sum_{p=1}^{N_d} \hat{S}_{m,p}^{\text{DNN}}, \\ \sigma^2_{S_m^{\text{DNN}}} &= \frac{1}{N_d}\sum_{p=1}^{N_d} (\hat{S}_{m,p}^{\text{DNN}} - \mu_{S_m^{\text{DNN}}})^2. \end{aligned} \tag{28}$$

In summary, two types of PIML strategies are proposed in this section and implemented in

two types of ML models (GP and DNN), in order to evaluate the accuracy and uncertainty of the sensitivity estimates in GSA. Eight different PIML models are developed by leveraging the two PIML strategies. The accuracy of these models can be assessed by comparison against validation data, whereas the variance of the sensitivity estimates can be quantified using Algorithms 1 and 2 and Eqs. 25 to 28. The different models have different training strategies and different numbers of parameters, both of which will affect the accuracy and uncertainty of the sensitivity estimates. These differences are assessed in detail in the next section.

4. Numerical illustration

4.1. Problem Setup

An additive manufacturing application is used to illustrate the proposed PIML models for GSA and compare their performance. A fused filament fabrication (FFF) process is considered; commercial material Ultimaker Black Acrylonitrile butadiene styrene (ABS) is extruded from an Ultimaker 2 Extended+ 3D printer to manufacture parts with unidirectionally aligned filaments, and the porosity of the manufactured part is measured. FFF is a widely used additive manufacturing (AM) process due to its easy operation, low cost, and suitability for complex geometries. As the molten filament is deposited layer upon layer through a nozzle, it cools down, solidifies and bonds with the adjacent filaments. Rectangular specimens of length 35 mm, width 12 mm, and thickness 4.2 mm are manufactured for the ABS amorphous polymer, with constant filament height, width and length (0.7, 0.8, and 35 mm, respectively).

The output QoI is porosity of the printed part, and the inputs are two process parameters, namely nozzle temperature and speed. The porosity of an FFF part is dependent on the temperature history at the interfaces between filaments. Thus, it is important to predict the temperature evolution of filaments for estimating the final mesostructure of the printed part. The analytical solution proposed by Costa et al. [42] for transient heat transfer during the printing process in FFF is used to predict the temperature evolution of filaments. A physics-based sintering model is developed, which considers realistic filament geometry, and allows the filament geometry to change during the printing process. This model is used to predict the porosity of

the FFF part using the temperature evolution of filaments, material properties, part geometry, and process parameters as inputs. Thus the mapping from input to output is a multi-physics model, i.e., models of two physical phenomena (heat transfer and sintering) are combined to predict the porosity given the values of two process parameters: extrusion temperature and extrusion speed.

The statistical properties of the QoI were observed to have negligible variability along the length of the specimens; therefore only the porosity measurements taken at the midpoint cross-section (see Fig. 3) are discussed here. These measurements were based on microscopy images processed through the ImageJ software [43]. Filaments were extruded through a nozzle with 0.8 mm diameter. The build plate temperature was constant and set to 110°C. Using Latin hypercube sampling, 39 sets of process parameters $\boldsymbol{X}$ were generated, and experiments were conducting at these 39 values. The ranges considered for the two process variables were: printer extrusion temperature T: ($210°C$ - $260°C$), and extrusion speed S: (15 mm/s - 46 mm/s).

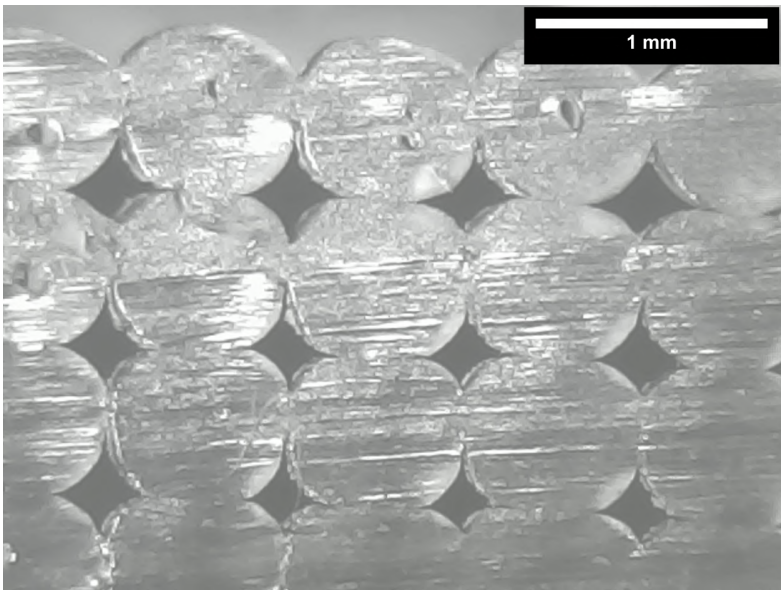

Fig. 3. Cross-sectional geometry of an FFF specimen printed with extrusion temperature 240°C and speed 42 mm/s.

4.2. Training details of the ML models

The basic ML models, namely Model 1 (for GP) and Model 5 (for DNN) are simply trained with the 39 sets of process inputs (temperature and speed) and output (porosity).

In the training of Model 2 ($\text{GP}^{\mathcal{L}_{\text{phy}}}$) and Model 6 ($\text{DNN}^{\mathcal{L}_{\text{phy}}}$), we impose two physics constraints (i.e., two separate loss function terms, $\mathcal{L}_{\text{phy},k}(\hat{\boldsymbol{Y}})$, where $k = \{1, 2\}$ and $\hat{\boldsymbol{Y}}$ is the porosity prediction). The corresponding loss function terms are defined as

$$\mathcal{L}_{\text{phy},1}(\hat{\boldsymbol{Y}}) = \frac{1}{N}\sum_{i=1}^{N} \text{ReLU}(-\hat{Y}_{\text{i}}),$$
$$\mathcal{L}_{\text{phy},2}(\hat{\boldsymbol{Y}}) = \frac{1}{N}\sum_{i=1}^{N} \text{ReLU}(\hat{Y}_{\text{i}} - \phi_{0,\text{i}}), \tag{29}$$

considering physics violations related to the porosity in all the N samples. In the first loss function, a negative value of porosity is treated as a physics violation. The second loss function penalizes the model when the predicted final porosity $\hat{Y}_i$ is greater than the initial porosity $\phi_{0,i}$ of the ith part. This is based on the physics knowledge that the total void area decreases as the bond formation takes place. Thus, the porosity predictions are ensured to be physically meaningful with the inclusion of these physics-based penalty terms.

The overall "loss" function of the GP model is

$$\mathcal{L}_{\overline{\text{GP}}} = \mathcal{L}_{\text{GP}} - \lambda^{\text{GP}}_{\text{phy},1}\mathcal{L}_{\text{phy},1}(\hat{\boldsymbol{Y}}) - \lambda^{\text{GP}}_{\text{phy},2}\mathcal{L}_{\text{phy},2}(\hat{\boldsymbol{Y}}). \tag{30}$$

Note that the GP model parameters are obtained by maximizing the above function.

The overall loss function of the DNN model is

$$\mathcal{L}_{\overline{\text{DNN}}} = \mathcal{L}_{\text{DNN}} + \lambda^{\text{DNN}}_{\text{phy},1}\mathcal{L}_{\text{phy},1}(\hat{\boldsymbol{Y}}) + \lambda^{\text{DNN}}_{\text{phy},2}\mathcal{L}_{\text{phy},2}(\hat{\boldsymbol{Y}}). \tag{31}$$

Note that the DNN model parameters are obtained by minimizing the above function.

In Model 3 (GP^{upd}) and Model 7 (DNN^{upd}), the ML models are pre-trained using the multi-physics model input-output. The pre-trained ML models are then updated using the experimental data. The training data for pre-training consists of 1310 input parameter combinations over a range of experimental values, i.e., ($210°C \leq T \leq 260°C$, 15 mm/s $\leq S \leq$ 46 mm/s). Note that there are only 39 physical experiments available; this is one of the advantages of the

pre-training/updating strategy, where the pre-training can be over a much larger set of input combinations, thus improving the generalization performance of the updated model. The input data are normalized prior to the training of the ML models (i.e., the output quantity porosity is dimensionless and between 0 and 1).

Model 4 ($\text{GP}^{\text{upd},\mathcal{L}_{\text{phy}}}$) combines both PIML strategies for GP, and consists of two GP models: (i) the first GP model is trained using the physics model input-output samples consisting of 1310 input parameter combinations; and (ii) the second GP model is built for the discrepancy between the first GP model prediction and the actual system response using the experimental data, by maximizing the function shown in Eq. 30 to optimize the hyperparameters of the second GP model.

Model 8 $\text{DNN}^{\text{upd},\mathcal{L}_{\text{phy}}}$, which is a combination of the two PIML strategies, uses the DNN model parameters trained using the physics model input-output as the initial values. Then, during the updating phase with the experimental data, these parameters are updated by minimizing the loss function shown in Eq. 31.

The four GP models (Models 1 to 4) were implemented using Python. The optimization of the hyperparameters were performed using the scikit-optimize package. The multipliers of the physics constraint terms of models 2 ($\text{GP}^{\mathcal{L}_{\text{phy}}}$) and 4 ($\text{GP}^{\text{upd},\mathcal{L}_{\text{phy}}}$) were chosen as $(\lambda^{\text{GP}}_{\text{phy},1}, \lambda^{\text{GP}}_{\text{phy},2}) = (50, 50)$ for the sake of illustration. The Automatic Relevance Determination (ARD) squared exponential function [44] was used as the covariance function for all the GP models.

The four DNN models (Models 5 to 8) were implemented using the Keras package [45] with Tensorflow in the backend. The hyperparameters of each model were tuned with grid search and the multipliers of the physics constraint terms in Model 6 ($\text{DNN}^{\mathcal{L}_{\text{phy}}}$) and Model 8 ($\text{DNN}^{\text{upd},\mathcal{L}_{\text{phy}}}$) were chosen as $(\lambda^{\text{DNN}}_{\text{phy},1}, \lambda^{\text{DNN}}_{\text{phy},2}) = (0.01, 0.01)$. Fully-connected DNN models with 2 hidden layers and 5 neurons in each hidden layer were constructed. The Rectified Linear Unit (ReLU) activation function and Adam optimizer were used to perform stochastic gradient descent for 300 epochs in learning the model parameters. The dropout rate for the DNN models was chosen to be 0.05, for reasons as explained below.

4.3. Comparison of computational effort

The computational costs of different models for training and estimation of Sobol' indices based on 5000 MC samples with a fixed number of experimental data ($n = 39$) is given in Table. 1. Among the GP models, the time it takes for training as well as computation of Sobol' indices using Models 2-4 is significantly greater than Model 1. The reason for the difference between the training time of GP and GP^{upd} is the pre-training phase, where a large amount of physics input-output samples used. Whereas, the difference between the training time of GP and $\text{GP}^{\mathcal{L}_{\text{phy}}}$ is due to the inclusion of physics constraints, which makes it harder for the optimization to find optimal hyperparameters. Interestingly, the training time for the DNN models ranges from 20 to 55 sec on the same desktop computer as used for the GP model (Intel® Xeon® CPU E5-1660 v4@3.20GHz with 32 GB RAM and GPU NVIDIA Quadro K620 with 2 GB). Among the DNN models, the reasons for the increased training time of models 6-8 compared to Model 5 are the same as for the GP models. The Sobol' index estimations based on 5000 samples take approximately 1-2 minutes for the DNN models (models 5-8), whereas the GP predictions take much longer because the covariance matrix needs to be stored and inverted.

Table 1
Computational effort of eight models for training and estimation of first-order and total-effect Sobol' indices using 5000 MC samples with $n = 39$ number of observations.

Models	Training	Sobol' indices calculation [in minutes]
1. GP	50 sec	3
2. $\text{GP}^{\mathcal{L}_{\text{phy}}}$	122 sec	5
3. GP^{upd}	165 sec	7
4. $\text{GP}^{\text{upd},\mathcal{L}_{\text{phy}}}$	195 sec	8
5. DNN	20 sec	1
6. $\text{DNN}^{\mathcal{L}_{\text{phy}}}$	45 sec	2
7. DNN^{upd}	30 sec	2
8. $\text{DNN}^{\text{upd},\mathcal{L}_{\text{phy}}}$	55 sec	2

4.4. Comparison of accuracy

In order to compare the accuracy of the eight different models, the models are trained with different amounts of experimental observations ($n = (5, 10, 15, 20, 30)$), and the remaining 9

Table 2
Effect of different amounts of training data on the RMSE of the GP models

Model	$n = 5$	$n = 10$	$n = 15$	$n = 20$	$n = 30$
1. GP	0.020($\pm$0.004)	0.026($\pm$0.005)	0.021($\pm$0.001)	0.021($\pm$0.002)	0.020($\pm$0.001)
2. GP$^{\mathcal{L}_{\text{phy}}}$	0.019($\pm$0.004)	0.026($\pm$0.004)	0.024($\pm$0.004)	0.021($\pm$0.002)	0.019($\pm$0.001)
3. GP$^{\text{upd}}$	0.021($\pm$0.003)	0.028($\pm$0.003)	0.020($\pm$0.001)	0.020($\pm$0.001)	0.019($\pm$0.001)
4. GP$^{\text{upd},\mathcal{L}_{\text{phy}}}$	0.021($\pm$0.005)	0.023($\pm$0.003)	0.022($\pm$0.003)	0.020($\pm$0.002)	0.019($\pm$0.001)

Table 3
Effect of different amounts of training data on the RMSE of the DNN models

Model	$n = 5$	$n = 10$	$n = 15$	$n = 20$	$n = 30$
5. DNN	0.027($\pm$0.007)	0.017($\pm$0.009)	0.019($\pm$0.006)	0.013($\pm$0.003)	0.013($\pm$0.003)
6. DNN$^{\mathcal{L}_{\text{phy}}}$	0.017($\pm$0.007)	0.015($\pm$0.008)	0.014($\pm$0.005)	0.014($\pm$0.003)	0.013($\pm$0.002)
7. DNN$^{\text{upd}}$	0.014($\pm$0.002)	0.009($\pm$0.002)	0.014($\pm$0.001)	0.013($\pm$0.001)	0.013($\pm$0.001)
8. DNN$^{\text{upd},\mathcal{L}_{\text{phy}}}$	0.014($\pm$0.002)	0.009($\pm$0.001)	0.014($\pm$0.001)	0.014($\pm$0.001)	0.013($\pm$0.001)

observations are used to compute the errors; the root mean square error (RMSE) based on the 9 validation samples is reported as the accuracy measure for comparison. The RMSE values for the four GP models and the four DNN models are shown in Tables 2 and 3 respectively, for different values of n. The results show that the use of PIML strategies in DNN models improves the performance, and the improvement is relatively larger as the amount of observed data n gets smaller. For n=20 and n=30, the basic DNN model is as accurate as the DNN models with the PIML strategies; whereas for smaller values of n, the DNN models with the PIML strategies are significantly more accurate. This clearly indicates the benefit of PIML over basic ML for the DNN models. However, the GP models incorporating PIML strategies do not show significant improvement. This is because the GP models have a much smaller number of parameters compared to the DNN models, and the achievable accuracy of any ML model is constrained by the number of model parameters (in addition to model form).

In addition to the increased number of parameters, the DNN models have additional advantages in terms of training epochs and dropout rate that affect the prediction accuracy. As mentioned earlier, the optimum number of training epochs was found to be 300. Regarding dropout rate, the RMSE values (based on 9 validation samples) of DNN models with MC dropout for different dropout rates are shown in Fig. 4. The lowest RMSE values for all four models are

obtained between 0.005-0.05. On the other hand, the Sobol' index estimates were found to be similar within this range of dropout rate, for each of the four DNN models. Therefore, a dropout rate of 0.05 was chosen for the reporting of GSA results, since a higher dropout rate results in smaller sub-networks, therefore a smaller number of parameters and faster training. By comparing Tables 2 and 3, it is seen that at the dropout rate of 0.05, the DNN models are more accurate compared to the GP models.

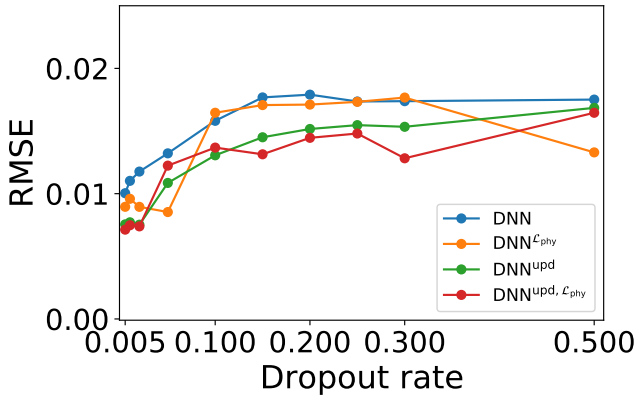

Fig. 4. RMSE values of DNN models with varying dropout rates.

4.5. GSA results using GP models

The Sobol' index computations with the GP models (1-4) are based on 5000 MC samples and 100 realizations of the Gaussian process. The effect of the number of experimental observations (used to train the GP models) on the first-order Sobol' index estimates from the four GP models is illustrated in Fig. 5. And the total-effect Sobol' index estimates from the GP models for different numbers of experimental training data are shown in Fig. 6. The mean values of sensitivity estimates based on the GP model predictions are denoted with solid dots at a given number of observations n. The sensitivity results are reported for two input variables, extrusion temperature and extrusion speed; the output quantity of interest is porosity.

The 95% prediction intervals are represented with bars above and below the solid dots for the corresponding model. The bounds in the sensitivity estimates are calculated using the mean predictions based on the 100 realizations of the GP models. As expected, the prediction intervals decrease as increasing amounts of experimental data are used to train the GP models.

Further, all four models converge to similar first-order and total-effect sensitivity estimates for both printer extrusion temperature and speed. The relative individual contribution (at n=39) of extrusion speed to the variance of the porosity (≈ 0.65) is greater than that of the extrusion temperature (≈ 0.25) and their sum is ≈ 0.9. And the sum of their total-effects indices (which capture parameter interactions) is slightly above 1.0, indicating that the interaction effect is small.

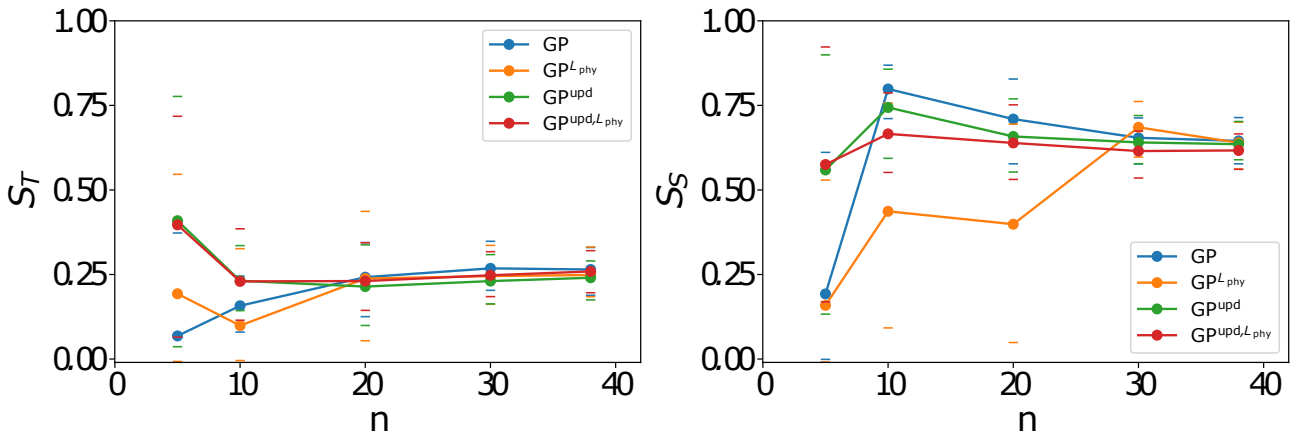

Fig. 5. First-order sensitivity index estimates for (a) extrusion temperature, and (b) extrusion speed, using the GP models.

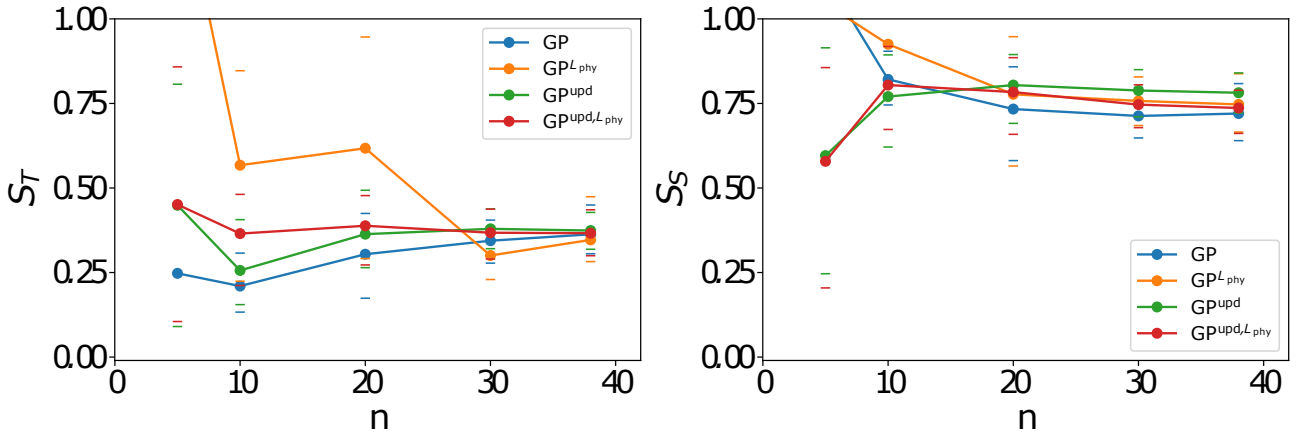

Fig. 6. Total-effect sensitivity index estimates for (a) extrusion temperature, and (b) extrusion speed, using the GP models.

For further clarity regarding uncertainty, numerical values of the prediction bounds of first-order sensitivity estimates of extrusion temperature obtained using 100 realizations of the GP models are shown in Table 4. The results indicate that prediction intervals obtained using the

PIML models 3 and 4 (GP^{upd} and $GP^{upd,\mathcal{L}_{phy}}$) converge to the bounds obtained using 39 number of observations for training the models faster than the first two models. Note that the upper 95% bound for Model 2 ($GP^{\mathcal{L}_{phy}}$) converges to its final value (0.33) within 10 experimental observations.

Table 4
First-order sensitivity estimate prediction bounds of extrusion temperature for different amounts of experimental training data, using the GP models.

Models	Lower 95% confidence limit					Upper 95% confidence limit				
	n=5	n=10	n=20	n=30	n=39	n=5	n=10	n=20	n=30	n=39
1. GP	0.00	0.08	0.13	0.20	0.19	0.37	0.25	0.34	0.35	0.33
2. $GP^{\mathcal{L}_{phy}}$	0.00	0.00	0.05	0.16	0.18	0.55	0.33	0.43	0.34	0.33
3. GP^{upd}	0.03	0.14	0.10	0.16	0.17	0.77	0.33	0.34	0.31	0.29
4. $GP^{upd,\mathcal{L}_{phy}}$	0.06	0.11	0.14	0.18	0.20	0.71	0.38	0.34	0.31	0.32

4.6. GSA results using DNN models with MC dropout

The Sobol' index computations with the DNN models (5-8) are based on 5000 MC samples and 100 stochastic forward passes through the networks. The calculated first-order sensitivity estimates of temperature and speed, S_T and S_S respectively, for different numbers of experimental observations (training data) $n = (5, 10, 20, 30, 39)$ are shown in Fig. 7. The distributions of sensitivity estimates are obtained using MC dropout predictions based on 100 stochastic forward passes through the networks for different number of observations. The mean values of sensitivity estimates are represented with solid dots and the 95% bounds are denoted with bars above and below the solid dots for the corresponding model. Similarly, the calculated total-effect sensitivity estimates S_T and S_S for different values of n are illustrated in Fig. 8. Similar to the GP results, the difference between the total-effect and first-order indices of process inputs is negligible, which indicates that their interaction is not significant.

All DNN models converge to similar first-order and total-effect sensitivity estimates for both inputs, and these values are consistent with the results obtained using GP models. For this problem with two inputs and one output, 39 experimental observations appear adequate to train even the basic DNN model to achieve similar performance as the other physics-informed models; in fact, the results are similar even at 20 observations. However, the superior accuracy

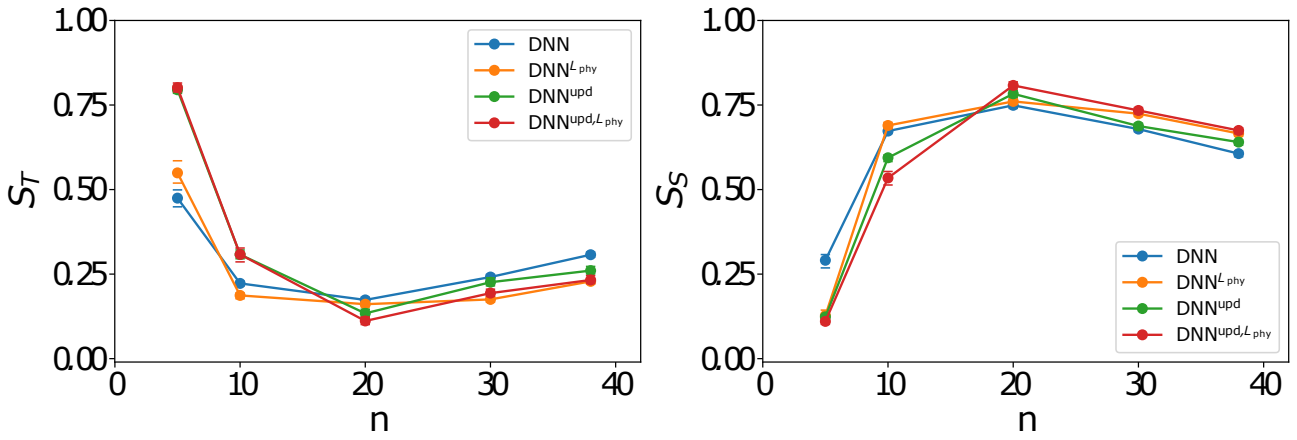

Fig. 7. First-order sensitivity index estimators for (a) extrusion temperature, and (b) extrusion speed, using DNN models with MC dropout.

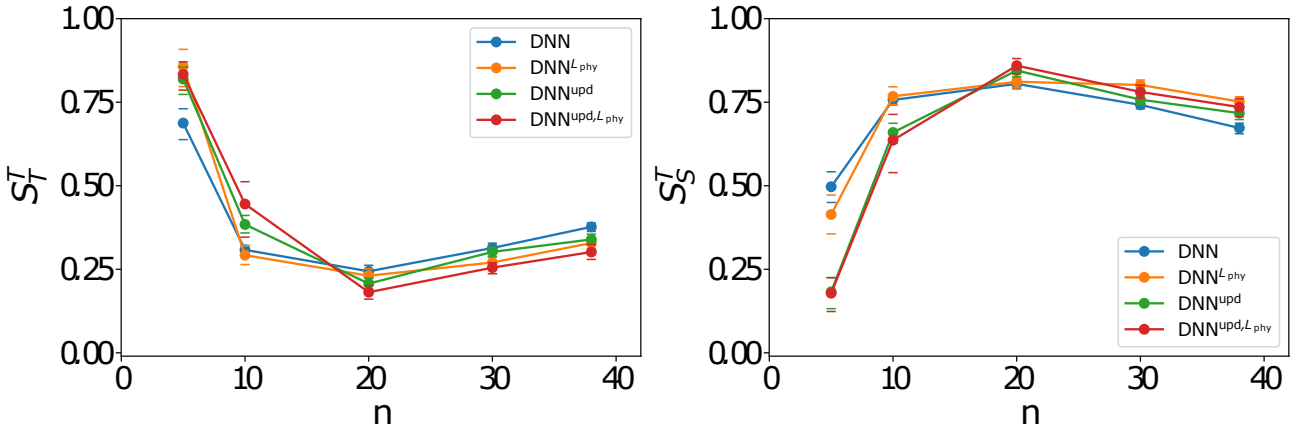

Fig. 8. Total-effect sensitivity index estimators for (a) extrusion temperature, and (b) extrusion speed, using DNN models with MC dropout.

of the physics-informed models becomes apparent if only a small number of experiments are available, say $n = 5$ or $n = 10$, as shown in Table 3 and discussed earlier in Section 4.4.

Note that the mean estimates of the GP models converge more smoothly and for a smaller number of experimental training data than the DNN models. This is because the GP models have a much smaller number of parameters (5) compared to the DNN models (23); as a result, the DNN models have more fluctuation and need more training data for convergence in the mean estimates. (Even though the full DNN model has 23 parameters, only sub-networks with smaller number of parameters are trained due to dropout).

For further clarity regarding uncertainty, numerical values of the prediction bounds of first-

order sensitivity estimates of extrusion temperature obtained using 100 forward passes through the DNN models are given in Table 5. The results show that prediction intervals obtained using all models show a similar trend. The 95% bounds for all models are significantly narrower than the ones obtained using the GP models. The uncertainty in the sensitivity estimates due to the DNN models is almost negligible when more than 10 number of observations are used to train the models.

Table 5
First-order sensitivity estimate prediction bounds of extrusion temperature for different amounts of experimental training data, using the DNN models.

Models	Lower 95% confidence limit					Upper 95% confidence limit				
	n=5	n=10	n=20	n=30	n=39	n=5	n=10	n=20	n=30	n=39
5. DNN	0.45	0.22	0.17	0.23	0.30	0.50	0.23	0.18	0.25	0.32
6. $\text{DNN}^{\mathcal{L}_{\text{phy}}}$	0.52	0.18	0.15	0.17	0.22	0.59	0.19	0.17	0.18	0.24
7. DNN^{upd}	0.78	0.30	0.12	0.22	0.25	0.81	0.32	0.15	0.24	0.27
8. $\text{DNN}^{\text{upd},\mathcal{L}_{\text{phy}}}$	0.78	0.29	0.10	0.18	0.22	0.82	0.33	0.12	0.21	0.25

The 95% prediction intervals (upper limit-lower limit) decrease as increasing amounts of experimental data are used to train the DNN models. However, the prediction intervals obtained using the DNN models are much smaller than the ones obtained using the GP models since the DNN models has more degrees of freedom that can be optimized. For example, for $n = 39$, the prediction interval width is 0.02-0.03 for the DNN models, whereas it is 0.12-0.15 for the GP models. In addition to the larger number of parameters, the number of training epochs, which is the number of complete passes through a batch of training dataset, is optimized for the DNN models, thus maximizing their prediction accuracy. The prediction accuracy is further improved by choosing the appropriate dropout rate as discussed earlier. The number of parameters to be learned reduces with the use of dropout, which helps with regularization and prevents ill-conditioning. Further, the number of training epochs is also observed to affect both the accuracy and uncertainty of the sensitivity estimates. It was found that at a smaller number of epochs, the model was not fully trained resulting in underfitting, leading to larger bias and variance in the prediction. As the number of epochs was increased, both the bias and the variance were reduced.

The results of this numerical example could be summarized as follows:

- The GP models required more computational effort than the DNN models, both in training and prediction.

- The DNN models were able to achieve higher accuracy and lower uncertainty in prediction, due to the larger number of parameters and the optimization of dropout rate and number of training epochs.

- The DNN models, due to their larger number of parameters, required more training data for convergence than the GP models.

- The physics-informed ML models are able to achieve higher prediction accuracy than the basic ML models, especially when the amount of available experimental data is small.

Overall, for this numerical example, the DNN models gave higher accuracy and lower uncertainty in the GSA results than the GP models, and also required less computational effort both in training and prediction. However, this numerical example consisted of only two inputs and a single output. As the number of inputs and outputs increase, all the ML models considered above will be challenged w.r.t. adequacy of training data, computational effort in training and prediction, and the accuracy and uncertainty of the sensitivity estimates.

5. Conclusion

This paper developed methodologies for information fusion and machine learning for sensitivity analysis using both physics knowledge and experimental data, while accounting for model uncertainty. Variance-based sensitivity analysis is used to quantify the relative contribution of each uncertainty source to the variability of the output quantity. Two types of ML models were considered, namely, GP and DNN models. Several PIML models were developed by leveraging two strategies for incorporating physics knowledge into ML models: (1) incorporating physics constraints within the loss functions used in training the ML models, and (2) pre-training an ML model with simulation data and then updating it with experimental data. The first strategy does not use the physics model, whereas the second strategy does. Prediction bounds are

computed for the sensitivity index estimates to account for the model uncertainty in the trained models, and the accuracy and computational of the various PIML models are compared.

The results show that the application of PIML strategies to both GP and DNN enables accurate Sobol' index computations even with smaller amounts of experimental data while producing physically meaningful results. Thus, the proposed approach helps to fill the physics knowledge gap in the ML models while estimating the Sobol' indices, by correcting for the approximation in the physics-based models. The numerical example shows that training the GP models and estimating the Sobol' indices require more computational effort than the DNN models. The uncertainty regarding the sensitivity estimates obtained using the DNN models is smaller than the results obtained using the GP models. In this numerical example, the DNN models are found to be more accurate compared to the GP models. The higher accuracy, lower uncertainty, and lower computational effort of the DNN models is attributed to their flexibility in terms of number of parameters, training epochs and dropout rate.

In future work, the proposed PIML approaches need to be tested for problems with a larger number of dimensions both in the input and output, with multiple combinations to further analyze the convergence of Sobol' index estimates. Future work can also explore the weighting of the two sources of information, since the data produced by physics-based models and experiments have different levels of credibility.

References

[1] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, S. Tarantola, Global sensitivity analysis: the primer, John Wiley & Sons, 2008.

[2] A. Saltelli, S. Tarantola, K.-S. Chan, A quantitative model-independent method for global sensitivity analysis of model output, Technometrics 41 (1) (1999) 39–56.

[3] T. A. Mara, S. Tarantola, Variance-based sensitivity indices for models with dependent inputs, Reliability Engineering & System Safety 107 (2012) 115–121.

[4] E. Borgonovo, E. Plischke, Sensitivity analysis: a review of recent advances, European Journal of Operational Research 248 (3) (2016) 869–887.

[5] S. Sankararaman, S. Mahadevan, Separating the contributions of variability and parameter uncertainty in probability distributions, Reliability Engineering & System Safety 112 (2013) 187–199.

[6] C. Li, S. Mahadevan, Relative contributions of aleatory and epistemic uncertainty sources in time series prediction, International Journal of Fatigue 82 (2016) 474–486.

[7] 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.

[8] J. E. Oakley, A. O'Hagan, Probabilistic sensitivity analysis of complex models: a bayesian approach, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 66 (3) (2004) 751–769.

[9] A. Marrel, B. Iooss, B. Laurent, O. Roustant, Calculations of sobol indices for the gaussian process metamodel, Reliability Engineering & System Safety 94 (3) (2009) 742–751.

[10] Z. Hu, S. Mahadevan, Probability models for data-driven global sensitivity analysis, Reliability Engineering & System Safety 187 (2019) 40–57.

[11] A. Marrel, B. Iooss, F. Van Dorpe, E. Volkova, An efficient methodology for modeling complex computer codes with gaussian processes, Computational Statistics & Data Analysis 52 (10) (2008) 4731–4744.

[12] A. O'Hagan, Bayesian analysis of computer code outputs: A tutorial, Reliability Engineering & System Safety 91 (10-11) (2006) 1290–1300.

[13] 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.

[14] B. Sudret, Global sensitivity analysis using polynomial chaos expansions, Reliability engineering & system safety 93 (7) (2008) 964–979.

[15] W. Chen, R. Jin, A. Sudjianto, Analytical variance-based global sensitivity analysis in simulation-based design under uncertainty (2005).

[16] S. Tarantola, D. Gatelli, T. A. Mara, Random balance designs for the estimation of first order global sensitivity indices, Reliability Engineering & System Safety 91 (6) (2006) 717–727.

[17] F. Satterthwaite, Random balance experimentation, Technometrics 1 (2) (1959) 111–137.

[18] V. Ginot, S. Gaba, R. Beaudouin, F. Aries, H. Monod, Combined use of local and anova-based global sensitivity analyses for the investigation of a stochastic dynamic model: application to the case study of an individual-based model of a fish population, Ecological modelling 193 (3-4) (2006) 479–491.

[19] G. Archer, A. Saltelli, I. Sobol, Sensitivity measures, anova-like techniques and the use of bootstrap, Journal of Statistical Computation and Simulation 58 (2) (1997) 99–120.

[20] C. Li, S. Mahadevan, An efficient modularized sample-based method to estimate the first-order sobol' index, Reliability Engineering & System Safety 153 (2016) 110–121.

[21] E. C. DeCarlo, S. Mahadevan, B. P. Smarslok, Efficient global sensitivity analysis with correlated variables, Structural and Multidisciplinary Optimization 58 (6) (2018) 2325–2340.

[22] A. Karpatne, W. Watkins, J. Read, V. Kumar, Physics-guided neural networks (pgnn): An application in lake temperature modeling, arXiv preprint arXiv:1710.11431 (2017).

[23] X. Jia, J. Willard, A. Karpatne, J. S. Read, J. A. Zwart, M. Steinbach, V. Kumar, Physics-guided machine learning for scientific discovery: An application in simulating lake temperature profiles, arXiv preprint arXiv:2001.11086 (2020).

[24] J. Willard, X. Jia, S. Xu, M. Steinbach, V. Kumar, Integrating physics-based modeling with machine learning: A survey, arXiv preprint arXiv:2003.04919 (2020).

[25] L. L. Gratiet, S. Marelli, B. Sudret, Metamodel-based sensitivity analysis: polynomial chaos expansions and gaussian processes, arXiv preprint arXiv:1606.04273 (2016).

[26] A. Marrel, B. Iooss, S. Da Veiga, M. Ribatet, Global sensitivity analysis of stochastic computer models with joint metamodels, Statistics and Computing 22 (3) (2012) 833–847.

[27] D. Xiu, G. E. Karniadakis, The wiener–askey polynomial chaos for stochastic differential equations, SIAM journal on scientific computing 24 (2) (2002) 619–644.

[28] A. Janon, M. Nodet, C. Prieur, Uncertainties assessment in global sensitivity indices estimation from metamodels, International Journal for Uncertainty Quantification 4 (1) (2014).

[29] Z. Hu, X. Du, Mixed efficient global optimization for time-dependent reliability analysis, Journal of Mechanical Design 137 (5) (2015).

[30] N. Muralidhar, M. R. Islam, M. Marwah, A. Karpatne, N. Ramakrishnan, Incorporating prior domain knowledge into deep neural networks, in: 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018, pp. 36–45.

[31] 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.

[32] Y. Ling, J. Mullins, S. Mahadevan, Selection of model discrepancy priors in bayesian calibration, Journal of Computational Physics 276 (2014) 665–680.

[33] J. S. Denker, Y. LeCun, Transforming neural-net output levels to probability distributions, in: Advances in neural information processing systems, 1991, pp. 853–859.

[34] D. J. MacKay, A practical bayesian framework for backpropagation networks, Neural computation 4 (3) (1992) 448–472.

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

[36] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural networks, arXiv preprint arXiv:1505.05424 (2015).

[37] A. Graves, Practical variational inference for neural networks, in: Advances in neural information processing systems, 2011, pp. 2348–2356.

[38] J. M. Hernández-Lobato, Y. Li, M. Rowland, D. Hernández-Lobato, T. Bui, R. Turner, Black-box α-divergence minimization (2016).

[39] Y. Gal, Z. Ghahramani, Bayesian convolutional neural networks with bernoulli approximate variational inference, arXiv preprint arXiv:1506.02158 (2015).

[40] Y. Gal, Z. Ghahramani, Dropout as a bayesian approximation: Representing model uncertainty in deep learning, in: international conference on machine learning, 2016, pp. 1050–1059.

[41] X. Zhang, S. Mahadevan, Bayesian neural networks for flight trajectory prediction and safety assessment, Decision Support Systems (2020) 113246.

[42] S. Costa, F. Duarte, J. Covas, Estimation of filament temperature and adhesion development in fused deposition techniques, Journal of Materials Processing Technology 245 (2017) 167–179.

[43] C. A. Schneider, W. S. Rasband, K. W. Eliceiri, Nih image to imagej: 25 years of image analysis, Nature methods 9 (7) (2012) 671.

[44] C. K. Williams, C. E. Rasmussen, Gaussian processes for machine learning, Vol. 2, MIT press Cambridge, MA, 2006.

[45] F. Chollet, et al., keras. github repository, https://github. com/fchollet/keras¿. Accessed on 25 (2015) 2017.