COMPUTER MODEL CALIBRATION WITH TIME SERIES DATA USING DEEP LEARNING AND QUANTILE REGRESSION[*]

SAUMYA BHATNAGAR[†], WON CHANG [‡], SEONJIN KIM[§], AND JIALI WANG[¶]

Abstract. Computer models play a key role in many scientific and engineering problems. One major source of uncertainty in computer model experiments is input parameter uncertainty. Computer model calibration is a formal statistical procedure to infer input parameters by combining information from model runs and observational data. The existing standard calibration framework suffers from inferential issues when the model output and observational data are high-dimensional dependent data, such as large time series, due to the difficulty in building an emulator and the non-identifiability between effects from input parameters and data-model discrepancy. To overcome these challenges, we propose a new calibration framework based on a deep neural network (DNN) with Long short-term memory layers that directly emulates the inverse relationship between the model output and input parameters. Adopting the 'learning with noise' idea, we train our DNN model to filter out the effects from data model discrepancy on input parameter inference. We also formulate a new way to construct interval predictions for DNN using quantile regression to quantify the uncertainty in input parameter estimates. Through a simulation study and real data application with the Weather Research and Forecasting Model Hydrological modeling system (WRF-Hydro), we show our approach can yield accurate point estimates and well-calibrated interval estimates for input parameters.

Key words. Computer Model Calibration, Deep Learning, Long short-term Memory Network, Data-Model Discrepancy

AMS subject classifications. 62P12

1. Introduction.

Computer models play an important role in almost every field of science and engineering. These models are typically a collection of a large number of partial differential equations designed to capture the behavior of a real world process. These models typically have a set of uncertain input parameters that need to be properly calibrated using real data to generate realistic simulation. Since the seminal paper by [22] there has been a considerable growth in the literature of computer model calibration [2, 6, 19, 30, 35, 36, 42]).

The methodological challenges in this area can be summarized into two aspects. The first aspect stems from the fact that computer model runs are often available only at a limited number of design points. This leads to the need of a statistical surrogate (emulator) for the computer model in question, typically done by constructing a Gaussian process (GP) model that interpolates computer model outputs at input parameter settings for which the model runs are not obtained [28]. This issue is further complicated by the fact that modern computer model outputs are usually in the form of high-dimensional data with a complicated dependence structure, such as large time series or spatial data. Building a GP emulator for such data poses considerable statistical and inferential challenges [7, 17, 19, 30] and the amount of effort to address these challenges often exceeds that to solve the calibration problem itself.

The second aspect comes from the fact that most computer models are imper-

[*]Submitted to the editors Nov 25th 2020.

Funding: This research is partially supported by the University of Cincinnati Charles Phelps Taft Research Center and the Ohio Super Computing Center (OSC).

[†]University of Cincinnati, Cincinnati, OH (bhatnasa@mail.uc.edu).
[‡]University of Cincinnati, Cincinnati, OH (changwn@ucmail.uc.edu, http://www.wonchang.net).
[§]Miami University, Oxford OH (kims20@miamioh.edu).
[¶]Argonne National Laboratory, LeMont, IL (jialiwang@anl.gov).

This manuscript is for review purposes only.

fect in representing the reality, and hence, one can reasonably expect that there is considerable discrepancy between the computer model output and the corresponding real-world observation. When the model output is in the form of complicated dependent data, such as time series, the corresponding data-model discrepancy also likely has a complex dependent structure. If not handled properly, this problem can cause significant bias in input parameter estimation. The existing methods rely on problem-specific solutions such as assuming a prior distribution [5] or regularizing the complexity of the discrepancy term [7, 36]. However, such solutions require substantial knowledge or specific assumptions about the form of discrepancy, which are not always available or justifiable.

In this paper, we propose an alternative framework to the existing calibration approach that takes advantage of the recent development in deep neural network (DNN) methodologies. Our focus is on calibration using time series data, which are one of the most common forms of computer model output [2, 19], but the basic framework can be easily modified to other types of data, such as spatial data. The main idea is to build a DNN model that can predict the optimal input parameter values for a given observational data by emulating the inverse relationship between the model output and input parameter values. To effectively filter out the effect of possible data-model discrepancy without imposing a strong assumption on the discrepancy term we adopt the idea of 'learning with noise' [1, 4, 20, 23, 37]. In combination with the feature extraction capability of the modern DNN architecture, this approach allows us to train a DNN model that can focus on the features that are relevant to parameter estimation while negating the effect of discrepancies.

In addition to the new calibration framework, we propose a new way to quantify uncertainty in prediction using DNN. Computer model calibration requires not only estimating the optimal values for the input parameters but also quantifying the surrounding uncertainties. Uncertainty quantification for DNN predictions is, in general, challenging because a DNN typically contains a large number of model parameters and it has been unclear how to reflect uncertainties in those parameters when constructing interval predictions without relying on some variational approximation to the likelihood function [10]. We propose a quantile regression approach based on the observation that a DNN can be viewed as a linear regression with basis functions that are created by hidden layers. Our simulation study shows that this approach provides a better way to quantify the uncertainty as it is not prone to overconfidence issues that variational approximation-based approaches typically suffer from. To demonstrate that our method can efficiently estimate input parameters in a complicated modern computer model, we apply our method to WRF-Hydro, a recently developed hydrologic module for the weather research and forecast (WRF) model [13].

The remainder of the paper is organized as follows: Section 2 describes the existing standard calibration framework and explains the common inferential challenges faced by the approach. Section 3 introduces our new inverse model-based framework using DNN that can overcome the challenges described in Section 2. Section 4 describes the details of inference procedure for our calibration method including regularized optimization and uncertainty quantification with quantile regression. Section 5 describes simulation study and Section 6 shows an example application of our approach to WRF-Hydro model. Section 7 summarizes the findings from our work and discusses future research directions.

2. Standard Calibration Framework and its Challenges. We first define notation for the model output, input parameters and observational data to facilitate our discussion on the methodological development. Let $\mathbf{Y}(\boldsymbol{\theta})$ be a p-dimensional model output at an input parameter setting $\boldsymbol{\theta} \in \mathcal{R}^{d_\theta}$. The output $\mathbf{Y}(\boldsymbol{\theta})$ is typically in the form of spatial or temporal or spatio-temporal data. We also let $\mathbf{Z} = [Z_1, \ldots, Z_p]^T$ be the p-dimensional observational data that have the same format as the model output $\mathbf{Y}(\boldsymbol{\theta})$. Throughout the rest of this paper, we focus on the situation where both $\mathbf{Y}(\boldsymbol{\theta})$ and $\mathbf{Z}$ are temporal data. Since in most scientific applications obtaining the model output $\mathbf{Y}(\boldsymbol{\theta})$ at each input parameter setting $\boldsymbol{\theta}$ is computationally expensive, model outputs are obtained at a limited number of design points $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$, with n being typically hundreds or thousands. The resulting collection of model outputs $\mathbf{Y}(\boldsymbol{\theta}_1), \ldots, \mathbf{Y}(\boldsymbol{\theta}_n)$ is often called a perturbed physics ensemble.

The objective of statistical computer model calibration is to find the best input parameter setting $\boldsymbol{\theta}^*$ for $\mathbf{Z}$ given the observed relationship between $\boldsymbol{\theta}$ and $\mathbf{Y}(\boldsymbol{\theta})$ from the perturbed physics ensemble. This problem therefore can be viewed as a classification problem with continuous labels $\boldsymbol{\theta}$. For our scientific problem described in Section 6 the number of model runs is 400 ($n = 400$), the size of each model run and observational data is 480 ($p = 480$), and the dimensionality of individual input parameter setting d_θ is 5 ($d_\theta = 5$).

2.1. Existing Forward Model-based Approach. In this section, we describe the existing standard computer model calibration framework that is currently widely used in the statistical literature. The standard computer model calibration model described in [22] can be written as

$$\mathbf{Z} = \mathbf{Y}(\boldsymbol{\theta}^*) + \boldsymbol{\delta}, \tag{2.1}$$

where $\boldsymbol{\delta}$ represents the data-model discrepancy often modeled by a p-dimensional GP model. The discrepancy includes both the structural error in the computer model (i.e. misrepresentation of the reality by the computer model) and the measurement error in observational data. The data type for $\mathbf{Z}$ and $\mathbf{Y}(\boldsymbol{\theta})$ determines the form of covariance function for $\boldsymbol{\delta}$. Here we assume that $\mathbf{Z}$ and $\mathbf{Y}(\boldsymbol{\theta})$ are time series, and hence, the discrepancy term is also a time series that can be denoted as $\boldsymbol{\delta} = [\delta_1, \delta_2, \ldots, \delta_t, \ldots, \delta_p]$. In this case a 1-dimensional Matérn class or an autoregessive model can be used as a model for $\boldsymbol{\delta}$. The likelihood function based on (2.1) can be used for inferring $\boldsymbol{\theta}^*$, while accounting for possible data-model discrepancies and observational errors. Evaluating the likelihood function based on the model in (2.1) requires running the forward model $\mathbf{Y}(\cdot)$ for the given value of $\boldsymbol{\theta}^*$ and hence we call this method a forward model-based calibration. If the forward model $\mathbf{Y}(\cdot)$ is computationally expensive and the evaluated model output $\mathbf{Y}(\boldsymbol{\theta})$ is available at only a limited number of input parameter settings, which is the case for most scientific problems including the problem described in Section 6, an emulator $\boldsymbol{\eta}(\boldsymbol{\theta})$ that approximates the forward model $\mathbf{Y}(\boldsymbol{\theta})$ is used instead. The emulator is typically constructed based on model runs $\mathbf{Y}(\boldsymbol{\theta}_1), \ldots, \mathbf{Y}(\boldsymbol{\theta}_n)$ obtained at pre-specified design points $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ using a GP model [28].

2.2. Challenges in Existing Framework. The forward model-based calibration framework described above often faces two important inferential and computational challenges. First, in most calibration problems, we need to construct an emulator $\boldsymbol{\eta}(\boldsymbol{\theta})$ that can accurately predict the model output $\mathbf{Y}(\boldsymbol{\theta})$ at any given new $\boldsymbol{\theta}$ that is not tried in the existing ensemble $\mathbf{Y}(\boldsymbol{\theta}_1), \ldots, \mathbf{Y}(\boldsymbol{\theta}_n)$. This is often challenging, especially when the model output $\mathbf{Y}(\boldsymbol{\theta})$ exhibits a complicated dependence structure. Such a problem is often complicated by the usual big data issues for GP-based methods, i.e. the likelihood evaluation becomes computationally slow or even infeasible due to the difficulty in taking a Cholesky decomposition of a large covariance matrix [7, 8, 19] when the model output is in the form of high-dimensional, dependent data, such as large time series. The computational complexity for each likelihood evaluation scales as $\mathcal{O}(p^3)$.

Second, the effects from the input parameter $\boldsymbol{\theta}^*$ and the effects from the data-model discrepancy $\boldsymbol{\delta}$ cannot be identifiable in general, and hence, lead to biased or overly uncertain estimates for $\boldsymbol{\theta}^*$ [5, 30, 36]. In particular, if the observational data appear to be quite different from any of the model runs due to data-model discrepancy, parameter estimation results can be severely biased (defined as a terminal case in [30]), as a zero-mean discrepancy term $\boldsymbol{\delta}$ cannot easily capture such a trend. This also often leads to incorrect uncertainty quantification with poorly calibrated interval estimates for target input parameters, potentially resulting in a severe undercoverage of interval estimates.

3. Inverse Model-Based Calibration using DNN.

3.1. Inverse Model-Based Calibration Framework. In this section, we propose our new inverse model-based calibration method using a deep neural network that can overcome the aforementioned challenges in the existing forward model-based calibration method. The main idea is to find the inverse function $\boldsymbol{g}$ that provides the best input parameter setting $\boldsymbol{\theta}^*$ when the observational data $\mathbf{Z}$ is given, i.e.,

$$\boldsymbol{\theta}^* = \boldsymbol{g}(\mathbf{Z}) + \boldsymbol{\epsilon}, \tag{3.1}$$

with some d-dimensional prediction error term $\boldsymbol{\epsilon}$. Finding such function $\boldsymbol{g}$ can be thought as finding a function that satisfies

$$\boldsymbol{\theta} = \boldsymbol{g}(\mathbf{Y}(\boldsymbol{\theta}) + \boldsymbol{\delta}) + \boldsymbol{\epsilon}, \tag{3.2}$$

for any $\boldsymbol{\theta} \in \Theta$ where Θ is the possible range for $\boldsymbol{\theta}^*$. In other words, our objective is to find a function $\boldsymbol{g}$ that can filter out the discrepancy $\boldsymbol{\delta}$ and accurately estimate $\boldsymbol{\theta}$ that originally generated $\mathbf{Y}(\boldsymbol{\theta})$ in any given observation $\mathbf{Y}(\boldsymbol{\theta}) + \boldsymbol{\delta}$. Given the estimated function $\hat{\boldsymbol{g}}$ based on the model in (3.2), the best predicted parameter setting $\boldsymbol{\theta}^*$ can be simply computed by

$$\hat{\boldsymbol{\theta}}^* = \hat{\boldsymbol{g}}(\mathbf{Z}).$$

The approximation function $\hat{\boldsymbol{g}}$ has to possess the following properties: First of all, $\hat{\boldsymbol{g}}$ needs to be able to capture a highly nonlinear relationship, which is almost always expected in computer model calibration problems. In addition, $\hat{\boldsymbol{g}}$ needs to be able to handle high-dimensional predictor variables with a complicated dependence structure such as long time series or large spatial data (see, e.g., [2, 6, 7, 17, 18, 19, 35]) because modern computer models commonly generate such types of data as their output. Another consideration is noise filtering: the function $\hat{\boldsymbol{g}}$ needs to be able to recover $\boldsymbol{\theta}$ from a noisy model output $\mathbf{Y}(\boldsymbol{\theta}) + \boldsymbol{\delta}$ by filtering out the effects from the discrepancy $\boldsymbol{\delta}$.

In this paper, we use a DNN to find the approximation function $\hat{\boldsymbol{g}}$. This choice is natural because DNN models possess all three required characteristics above. The main feature of DNN is its ability to approximate highly complicated nonlinear functions, which has been proven in a wide range of applications and also discussed in some approximation theory point of view [9, 26, 31]. Moreover, the recently developed architectures in DNN such as Long short-term memory (LSTM) network [21] can provide a well-proven recipe for extracting important features from large time series data. The recently developed computational machineries including back-propagation and stochastic gradient descent algorithms facilitate easy implementation of DNN with a highly complicated structure. In the following subsections, we explain the details of our DNN-based method for computer model calibration.

3.2. DNN for Nonlinear Regression with Feature Extraction. The most commonly used DNN architecture consists of two components: feature extraction layers and nonlinear regression layers. The feature extraction layers apply a series of transformations to the input data set to find the features that are most relevant to predicting the response variables. For our calibration problem, the features found by the feature extraction layers can be interpreted as transformed data that are most relevant to estimating the input parameter setting. The nonlinear regression layers create a nonlinear function that links the extracted features to the response variables. In our calibration problem, the nonlinear regression layers estimate the best input parameter setting given the extracted features from data.

The form of feature extraction layers is determined by the data type of the model output $\mathbf{Y}(\boldsymbol{\theta})$ and the observational data $\mathbf{Z}$. Since our focus here is on time series data, the suitable feature extraction model will be a bidirectional LSTM network [21]. This structure combines information from the forward and backward LSTM units, where forward LSTM units model the information flow in time order and backward LSTM units model the information flow in reverse time order. This structure has been proven to be useful in capturing important features for sequence classification. The overall structure of the DNN structure described in this section is illustrated in Figure 1.

One important advantage of this approach is computational complexity, which is scaled as $\mathcal{O}(p^2)$ [29]. The difference in computing time between the DNN based method with LSTM and the GP-based method described in Section 2.1 grows exponentially as the size of model output p grows, because the computational complexity of the GP-Fwd method scales as $\mathcal{O}(p^3)$, as discussed in Section 2.2.

3.2.1. Long short-term Memory for Feature Extraction. Recurrent neural networks (RNN) are neural networks specialized in handling sequential data. The hidden layers in Recurrent neural network (RNN) are connected in a cyclic pattern or self-connected loop. The LSTM (Hochreiter and Schmidhuber, 1997; Gers et al.,2000) network is currently the most widely used recurrent neural network for various applications including speech recognition, natural language processing, and sentiment analysis [16, 29, 39]. The main advantage of LSTM network is its ability to handle both short-range and long-range dependence in a computationally efficient manner. Moreover, the gated structure of LSTM that regulates the information flow within the network is helpful for avoiding computational issues. (See Section S1 for further discussion. Note that sections, figures and tables referred with prefix S henceforth can be found in the Supplementary Document.)

An LSTM takes a sequence as input and passes it through connected hidden layers to yield estimated values as output at each time point. To be more specific, for a given d_x dimensional input vector $\mathbf{x}_t$ at each time point t, the d_c-dimensional cell vector $\overrightarrow{\mathbf{c}}_t$ and its corresponding d_c-dimensional output vector $\overrightarrow{\mathbf{h}}_t$ are computed as

$$\begin{aligned} \overrightarrow{\mathbf{c}}_t &= \overrightarrow{\mathbf{u}}_t^{(f)} * \overrightarrow{\mathbf{c}}_{t-1} + \overrightarrow{\mathbf{u}}_t^{(i)} * \mathbf{f}^{(c)}\left(\overrightarrow{\mathbf{W}}_x^{(c)}\mathbf{x}_t + \overrightarrow{\mathbf{W}}_h^{(c)}\overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{a}}^{(c)}\right), \\ \overrightarrow{\mathbf{h}}_t &= \overrightarrow{\mathbf{u}}_t^{(o)} * \mathbf{f}^{(h)}(\overrightarrow{\mathbf{c}}_t), \end{aligned} \tag{3.3}$$

where $\overrightarrow{\mathbf{W}}_x^{(c)}$ and $\overrightarrow{\mathbf{W}}_h^{(c)}$ are, respectively, $d_c \times d_x$ and $d_c \times d_c$ weight matrices for input

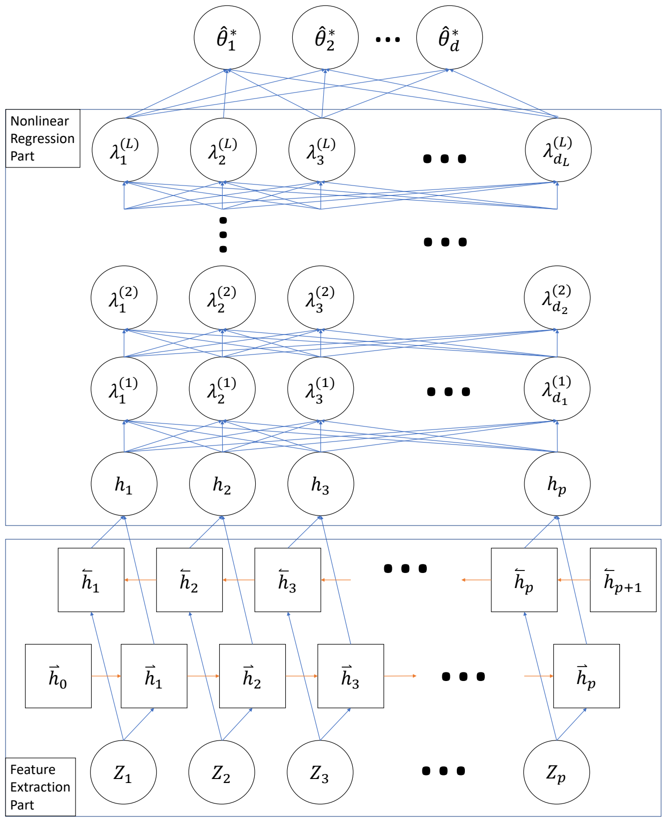

FIG. 1. *Illustration of DNN with LSTM layers. The lower rectangular area represents the bidirectional LSTM layers, which extract relevant features from time series. The upper rectangular area represents the fully connected layers of the model, which are for capturing the nonlinear relationship between the extracted features from the LSTM layers and the output* $\hat{\boldsymbol{\theta}}^*$.

$\mathbf{x}_t$ and output from previous time step $\overrightarrow{\mathbf{h}}_{t-1}$; $\overrightarrow{\mathbf{a}}^{(c)}$ is a d_c-dimensional intercept vector (often called bias in the deep learning literature); $\mathbf{f}^{(c)}|R^{d_c} \to R^{d_c}$ and $\mathbf{f}^{(h)}|R^{d_c} \to R^{d_c}$ are activation functions for nonlinear transformation. Here the arrow $\overrightarrow{\cdot}$ is used to emphasize that the matrices and vectors are for a network that models information flow going forward in time. (Below, a network for backward flow will be introduced as well.) The initial values $\overrightarrow{\mathbf{c}_0}$ and $\overrightarrow{\mathbf{h}_0}$ are set to be zeros. The operator $*$ denotes element-wise multiplication, and $\overrightarrow{\mathbf{u}}_t^{(f)}$, $\overrightarrow{\mathbf{u}}_t^{(i)}$, and $\overrightarrow{\mathbf{u}}_t^{(o)}$ are respectively the *forget*, *input* and *output* gate vectors (thereafter shortened as gate). The gates are defined in a similar fashion as a usual neural network node:

$$\begin{aligned}
\overrightarrow{\mathbf{u}}_t^{(f)} &= \mathbf{f}^{(f)}\left(\overrightarrow{\mathbf{W}}_x^{(f)}\mathbf{x}_t + \overrightarrow{\mathbf{W}}_h^{(f)}\overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{a}}^{(f)}\right),\\
\overrightarrow{\mathbf{u}}_t^{(i)} &= \mathbf{f}^{(i)}\left(\overrightarrow{\mathbf{W}}_x^{(i)}\mathbf{x}_t + \overrightarrow{\mathbf{W}}_h^{(i)}\overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{a}}^{(i)}\right),\\
\overrightarrow{\mathbf{u}}_t^{(o)} &= \mathbf{f}^{(o)}\left(\overrightarrow{\mathbf{W}}_x^{(o)}\mathbf{x}_t + \overrightarrow{\mathbf{W}}_h^{(o)}\overrightarrow{\mathbf{h}}_{t-1} + \overrightarrow{\mathbf{a}}^{(o)}\right),
\end{aligned} \tag{3.4}$$

where matrices denoted as $\overrightarrow{\mathbf{W}}_x^{(.)}$ and $\overrightarrow{\mathbf{W}}_h^{(.)}$ are, respectively, $d_c \times d_x$ and $d_c \times d_c$ weight matrices that link input variables $\mathbf{x}_t$ and previous output $\overrightarrow{\mathbf{h}}_{t-1}$ to each gate vector;

vectors denoted as $\overrightarrow{\mathbf{a}}^{(\cdot)}$ are d_c-dimensional intercept vectors for each gate; functions denoted as $\mathbf{f}^{(\cdot)}|R^{d_c} \to R^{d_c}$ are activation functions for each gate. These gates control how the information flows within the LSTM network, and including them improves numerical stability as well as prediction accuracy [11]. The input vector $\mathbf{x}_t$ is defined as the current and lagged variables of observed sequence, i.e. $\mathbf{x}_t = [Z_{t-d_t}, \dots, Z_t]^T$, which supplies information from short range time dependence (or short term memory) to the network. The sequential cells $\overrightarrow{\mathbf{c}}_1, \dots, \overrightarrow{\mathbf{c}}_p$ are designed to capture the long range dependence (or long term memory) in the modeled time sequence.

If our goal was to make predictions on the observed sequence Z_t, the models described in (3.3) and (3.4) would be enough. However, since our goal here is to extract features from the observed sequence and use it for finding the best value for $\boldsymbol{\theta}$, a bidirectional LSTM network is more suitable [21]. In addition to the forward LSTM layers described in (3.3) and (3.4), we have the following backward LSTM layers at each time step t:

$$\begin{aligned}\overleftarrow{\mathbf{c}}_t &= \overleftarrow{\mathbf{u}}_t^{(f)} * \overleftarrow{\mathbf{c}}_{t+1} + \overleftarrow{\mathbf{u}}_t^{(i)} * \mathbf{f}^{(c)}\left(\overleftarrow{\mathbf{W}}_x^{(c)}\mathbf{x}_t + \overleftarrow{\mathbf{W}}_h^{(c)}\overleftarrow{\mathbf{h}}_{t+1} + \overleftarrow{\mathbf{a}}^{(c)}\right),\\ \overleftarrow{\mathbf{h}}_t &= \overleftarrow{\mathbf{u}}_t^{(o)} * \mathbf{f}^{(h)}(\overleftarrow{\mathbf{c}}_t),\end{aligned} \tag{3.5}$$

with the following gate structure that has the same form as the forward LSTM units:

$$\begin{aligned}\overleftarrow{\mathbf{u}}_t^{(f)} &= \mathbf{f}^{(f)}\left(\overleftarrow{\mathbf{W}}_x^{(f)}\mathbf{x}_t + \overleftarrow{\mathbf{W}}_h^{(f)}\overleftarrow{\mathbf{h}}_{t+1} + \overleftarrow{\mathbf{a}}^{(f)}\right),\\ \overleftarrow{\mathbf{u}}_t^{(i)} &= \mathbf{f}^{(i)}\left(\overleftarrow{\mathbf{W}}_x^{(i)}\mathbf{x}_t + \overleftarrow{\mathbf{W}}_h^{(i)}\overleftarrow{\mathbf{h}}_{t+1} + \overleftarrow{\mathbf{a}}^{(i)}\right),\\ \overleftarrow{\mathbf{u}}_t^{(o)} &= \mathbf{f}^{(o)}\left(\overleftarrow{\mathbf{W}}_x^{(o)}\mathbf{x}_t + \overleftarrow{\mathbf{W}}_h^{(o)}\overleftarrow{\mathbf{h}}_{t+1} + \overleftarrow{\mathbf{a}}^{(o)}\right),\end{aligned} \tag{3.6}$$

where the vectors and matrices in (3.5) and (3.6) with the backward arrow $\overleftarrow{\cdot}$ have the same dimensionalities as their counterparts in (3.3) and (3.4) with the forward arrow $\overrightarrow{\cdot}$. Again, the initial values $\overleftarrow{\mathbf{c}_{T+1}}$ and $\overrightarrow{\mathbf{h}_{T+1}}$ are set to be zeroes. The output vectors from the backward and forward LSTM units at each time step t are summed into one output vector $\mathbf{h}_t$,

$$\mathbf{h}_t = \overrightarrow{\mathbf{h}}_t + \overleftarrow{\mathbf{h}}_t, \tag{3.7}$$

which will be passed to the nonlinear regression layers. Figure 1 illustrates the resulting LSTM structure.

The activation functions $\mathbf{f}^{(\cdot)}$ are defined as a collection of 1-dimensional functions $f_i^{(\cdot)}|R \to R$ $(i = 1, \dots, d_c)$ as follows:

$$\mathbf{f}^{(\cdot)}(\cdot) = \left[f_1^{(\cdot)}(\cdot), f_2^{(\cdot)}(\cdot), \dots, f_{d_c}^{(\cdot)}(\cdot)\right]^T.$$

Following a typical choice in the literature, we use the hard sigmoid function for the activation functions $\mathbf{f}^{(f)}$, $\mathbf{f}^{(i)}$, $\mathbf{f}^{(o)}$ for the gate variables, i.e.

$$f_i^{(\cdot)}(x) = \max(0, \min(1, x))$$

for $f_i^{(f)}$, $f_i^{(i)}$, and $f_i^{(o)}$ $(i = 1, \dots, d_c)$. This choice sets a large number of values in the gate vectors in both forward and backward LSTM layers ($\overrightarrow{u}_t^{(f)}$, $\overrightarrow{u}_t^{(i)}$, $\overrightarrow{u}_t^{(o)}$, $\overleftarrow{u}_t^{(f)}$, $\overleftarrow{u}_t^{(i)}$ and $\overleftarrow{u}_t^{(o)}$) to be zeros, and hence imposes a strong regularization through sparsity. For the remaining activation functions $\mathbf{f}^{(c)}$ and $\mathbf{f}^{(h)}$, we use the rectified linear unit (ReLU, see e.g., [14], Chapter 6):

$$f_i^{(\cdot)}(x) = \max(0, x) \tag{3.8}$$

for individual $f_i^{(c)}$ and $f_i^{(h)}$ $(i = 1, \ldots, J_c)$. It is well known that using ReLU as activation functions greatly increases numerical stability in likelihood estimation for deep neural networks. We discuss the rationale behind this choice in detail in Section S1.

3.2.2. Fully-Connected Layers for Nonlinear Regression. The final outputs from the feature extraction layers are vectorized (often referred to as flattening in the deep learning literature) as $\boldsymbol{\lambda}^{(0)} = [\mathbf{h}_1^T, \ldots, \mathbf{h}_p^T]^T$ and supplied to the nonlinear regression layers. We use a fully connected network with L layers as our nonlinear regression layers. The model structure for the fully connected layers can be written as

$$\begin{aligned}
\boldsymbol{\lambda}^{(1)} &= \mathbf{f}^{(1)}\left(\mathbf{W}^{(0)}\boldsymbol{\lambda}^{(0)} + \mathbf{a}^{(0)}\right), \\
\boldsymbol{\lambda}^{(2)} &= \mathbf{f}^{(2)}\left(\mathbf{W}^{(1)}\boldsymbol{\lambda}^{(1)} + \mathbf{a}^{(1)}\right), \\
&\cdots \\
\boldsymbol{\lambda}^{(L)} &= \mathbf{f}^{(L)}\left(\mathbf{W}^{(L-1)}\boldsymbol{\lambda}^{(L-1)} + \mathbf{a}^{(L-1)}\right), \\
\hat{\boldsymbol{\theta}}^* &= \mathbf{W}^{(L)}\boldsymbol{\lambda}^{(L)} + \mathbf{a}^{(L)},
\end{aligned} \tag{3.9}$$

where $\boldsymbol{\lambda}^{(l)}$ is the vector for the $d_{(l)}$ different nodes in the lth layer; $\mathbf{f}^{(l)}|R^{d_{(l)}} \to R^{d_{(l)}}$ is a vector-valued activation function for the lth layer; $\mathbf{W}^{(l)}$ is a $d_{(l+1)} \times d_{(l)}$ weight matrix; $\mathbf{a}^{(l)}$ is a $d_{(l+1)}$-dimensional intercept matrix (which is often called bias in the deep learning literature). The length of $\boldsymbol{\lambda}^{(0)}$ (i.e., $d_{(0)}$) is determined as Td_c because the length of each $\mathbf{h}_t$ is d_c. The sizes of subsequent layers, $d_{(1)}, \ldots, d_{(L)}$, which are often referred to as the widths of layers, need to be determined by the user. The width of the last layer $d_{(L+1)}$ is d_θ (the dimensionality of $\hat{\boldsymbol{\theta}}^*$) and hence $\mathbf{W}^{(L)}$ is a $d_\theta \times d_{(L)}$ matrix and $\mathbf{a}^{(L)}$ is a d_θ-dimensional vector.

The recent development in approximation theories (e.g., [9, 26, 31]) suggest that having multiple hidden layers (i.e., $L \gg 1$) to build a 'deep' network leads to a better prediction performance for the response variable than having a shallow network, coining the term 'deep learning'. Having a deep network, however, poses a danger of saturation or vanishing gradient, meaning that the gradient of the resulting likelihood function becomes zero for a wide range of predictor variables, and, hence, gradient-based optimization methods, such as gradient descent search, become computationally infeasible. (see Section 4.1 below for further discussion). This issue can be avoided by choosing a proper activation function: for the lth layer activation function $\mathbf{f}^{(l)}(\cdot) = \left[f_1^{(l)}(\cdot), f_2^{(l)}(\cdot), \ldots, f_{d_{(l)}}^{(l)}(\cdot)\right]^T$ we define the activation function as ReLU defined in (3.8). This choice of activation function also imposes certain level of sparsity to the network by making a large portion of $\boldsymbol{\lambda}^{(l)}$ become zeros.

3.3. Handling Data-Model Discrepancy. In our calibration approach, the main goal of statistical inference is to build an inverse function $\hat{\boldsymbol{g}}$ that can efficiently estimate $\boldsymbol{\theta}$ from $\mathbf{Y}(\boldsymbol{\theta}) + \boldsymbol{\delta}$, even under the presence of data-model discrepancy $\boldsymbol{\delta}$.

This problem resembles the problem of noisy sequence classification, except that the response variable is a continuous variable in our case. Inspired by the idea of learning with noise in the neural network literature [1, 4, 20, 23, 37] we propose to train the inverse emulator $\hat{\boldsymbol{g}}$ using contaminated model outputs instead of the original model outputs. In this way, the resulting neural network model $\hat{\boldsymbol{g}}$ can automatically extract the features $\boldsymbol{\lambda}^{(0)}$ from a noisy model output $\mathbf{Y}(\boldsymbol{\theta})+\boldsymbol{\delta}$ that is most relevant to recovering the input parameter setting $\boldsymbol{\theta}$.

To this end, we generate n_d different realizations of $\boldsymbol{\delta}$ from an assumed discrepancy distribution for each input parameter setting $\boldsymbol{\theta}_i$ $(i = 1, \ldots, n)$ to have generated discrepancy terms $\{\boldsymbol{\delta}_{ij}\}$ $(i = 1, \ldots, n$ and $j = 1, \ldots, n_d)$. We then create contaminated model outputs $\tilde{\mathbf{Y}}_1, \ldots, \tilde{\mathbf{Y}}_N$ with $N = n \times n_d$ by superimposing the generated discrepancy terms on the original model outputs as follows:

$$\tilde{\mathbf{Y}}_k = \mathbf{Y}(\boldsymbol{\theta}_i) + \boldsymbol{\delta}_{ij}$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, n_d$, where $k = n_d(i-1) + j$. We let $\tilde{\boldsymbol{\theta}}_1, \ldots, \tilde{\boldsymbol{\theta}}_N$ denote the input parameter settings used for creating $\tilde{\mathbf{Y}}_1, \ldots, \tilde{\mathbf{Y}}_N$ (i.e., $\tilde{\boldsymbol{\theta}}_k = \boldsymbol{\theta}_{\lceil k/n_d \rceil}$). This learning with error approach aims to train the DNN model with various types of data-model discrepancy patterns so that it can handle discrepancies varying in a wide range of magnitudes and time scales.

For the discrepancy model for $\boldsymbol{\delta}$ we use a zero mean Gaussian process model with the following squared exponential covariance function for generating $\boldsymbol{\delta}_{ij}$:

$$Cov(\delta_{t_1}, \delta_{t_2}) = \zeta 1(t_1 = t_2) + \kappa \exp\left(-\frac{|\delta_{t_1} - \delta_{t_2}|^2}{\phi}\right),$$

where $t_1, t_2 \in \{1, \ldots, T\}$, $1(\cdot)$ is an indicator function for the condition in $(\cdot)$; $\zeta > 0$, $\kappa > 0$, and $\phi > 0$ are respectively the nugget, partial sill, and the range parameters. To avoid imposing a too strong assumption on the discrepancy term, we allow these parameters to vary across different realizations of $\boldsymbol{\delta}_{ij}$ so that the resulting inverse function $\hat{g}$ can handle various types of $\boldsymbol{\delta}$ patterns. We generate a sample of size N for these parameter values based on a Latin hypercube design. Ranges for the parameters (preferably broad) are the only required input. The ranges for ζ and κ reflect model user's guess on the magnitudes of independent and time-dependent components in the data-model discrepancy. (See Section 5.2 for the specific parameter ranges used in our application problem.) As per the range of ϕ, one rule that can be used for a wide range of problems is to use a value between 1% and 10% of the time interval lengths (p) as the lower limit and a value between 60% and 70% of the length as the upper limit so that the generated discrepancy patterns cover various types of structured errors, including errors with short range dependence (when ϕ is near its lower limit) and overall mean shift (when ϕ is near its upper limit). One can choose a more informative sampling scheme that puts more emphasis on certain parts of the discrepancy parameter space if some prior knowledge that justifies such choice exists for the problem at hand.

4. Statistical Inference for DNN Calibration.

We now describe the details of inference for the calibration model proposed in Section 3. We illustrate how the model is fitted with a proper regularization and how the input parameters are predicted along with their uncertainty intervals.

4.1. Minimizing the Stochastic Loss Function with Dropout. In this section we use $\hat{\boldsymbol{g}}(\cdot)$ to exclusively denote the approximation function constructed by the deep network explained in Sections 3.2.1 and 3.2.2. We also let $\mathbf{w} = [w_1, \ldots, w_{n_w}]^T$ denote a vector of all parameters contained in weight matrices and intercept vectors defined in (3.3), (3.4), (3.5), (3.6), and (3.9) where n_w is the total number of parameters in the deep network model. The inference problem here is to estimate quantities in $\mathbf{w}$ based on $\tilde{\mathbf{Y}}_1, \ldots, \tilde{\mathbf{Y}}_N$. The standard cost function used in the deep learning literature for continuous response variables is the *squared loss function*, given as

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} (\hat{\boldsymbol{\theta}}_i^* - \tilde{\boldsymbol{\theta}}_i)^T (\hat{\boldsymbol{\theta}}_i^* - \tilde{\boldsymbol{\theta}}_i),$$

where $\hat{\boldsymbol{\theta}}_i^* = \hat{\boldsymbol{g}}(\tilde{\mathbf{Y}}_i)$. Minimizing this cost function is equivalent to maximizing the log-likelihood function for the model in (3.1) with an assumption $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \sigma^2 \mathbf{I}_{d_\theta})$, with $\sigma^2 > 0$ (i.e., assuming equal variance for $\boldsymbol{\epsilon}$). Here the equal variance assumption for the d_θ different input parameters can be justified by rescaling the input parameters so that they have the same range (typically [0,1]) and, hence, operate in the same scale.

The deep network model described in Sections 3.2.1 and 3.2.2 is apparently overparametrized and it is often helpful to impose some regularization for a better prediction performance (see [14], Chapter 7). We implement two approaches simultaneously that are frequently used in the deep learning literature: dropout and penalized likelihood.

Dropout is a way to create a stochastic likelihood function by introducing some randomness in the structure of our deep network ([14], Chapter 7.12). To be more specific, we re-define the fully connected layers in (3.9) as

$$\begin{aligned}
\boldsymbol{\lambda}^{(1)} &= \mathbf{f}^{(1)}\left(\mathbf{W}^{(0)}\boldsymbol{\lambda}^{(0)} + \mathbf{a}^{(0)}\right),\\
\boldsymbol{\lambda}^{(2)} &= \mathbf{f}^{(2)}\left(\mathbf{W}^{(1)}\boldsymbol{\lambda}^{(1)} * \mathbf{r}^{(1)} + \mathbf{a}^{(1)}\right),\\
&\cdots\\
\boldsymbol{\lambda}^{(L)} &= \mathbf{f}^{(L)}\left(\mathbf{W}^{(L-1)}\boldsymbol{\lambda}^{(L-1)} * \mathbf{r}^{(L-1)} + \mathbf{a}^{(L-1)}\right),\\
\hat{\boldsymbol{\theta}}^* &= \mathbf{W}^{(L)}\boldsymbol{\lambda}^{(L)} * \mathbf{r}^{(L)} + \mathbf{a}^{(L)},
\end{aligned} \tag{4.1}$$

where $\mathbf{r}^{(l)}$ for $l = 1, 2, \ldots, L$ is defined as $d_{(l)}$-dimensional vectors whose elements are identically and independently distributed Bernoulli random variables with a pre-specified success probability p_{keep}. We let $\mathbf{r}$ denote a collection of all $\mathbf{r}^{(l)}$'s, i.e., $\mathbf{r} = \left[{\mathbf{r}^{(1)}}^T, {\mathbf{r}^{(2)}}^T, \ldots, \; {\mathbf{r}^{(L)}}^T\right]^T$. The operator $*$ denotes element-wise multiplication. This leads to a stochastic loss function since new values of $\mathbf{r}$ are drawn for every evaluation of the function. The loss function with dropout $\mathcal{L}_{\mathbf{r}}(\mathbf{w})$ can be redefined as

$$\mathcal{L}_{\mathbf{r}}(\mathbf{w}) = \sum_{i=1}^{N} (\hat{\boldsymbol{\theta}}_{\mathbf{r},i}^* - \tilde{\boldsymbol{\theta}}_i)^T (\hat{\boldsymbol{\theta}}_{\mathbf{r},i}^* - \tilde{\boldsymbol{\theta}}_i). \tag{4.2}$$

where $\hat{\boldsymbol{\theta}}_{\mathbf{r},i}^* = \hat{\boldsymbol{g}}_{\mathbf{r}}\left(\tilde{\mathbf{Y}}_i\right)$ and $\hat{\boldsymbol{g}}_{\mathbf{r}}$ is the deep learning-based approximation function constructed based on (4.1) instead of (3.9). The subscript $\mathbf{r}$ is used to emphasize the dependence of the predicted values on the random vector $\mathbf{r}$.

For penalization we can choose any commonly used form including lasso, ridge, and elastic net as the penalty function, which we will denote as $\mathcal{P}(\mathbf{w})$ henceforth. The resulting penalized loss function is given as

$$\ell_{\mathbf{r}}(\mathbf{w}) \propto \mathcal{L}_{\mathbf{r}}(\mathbf{w}) + \mathcal{P}(\mathbf{w}). \tag{4.3}$$

One notable choice for $\mathcal{P}(\mathbf{w})$ in the literature [10] is a ridge penalty term defined in (S1) in Section S2. When combined with dropout, the resulting penalized likelihood function (i.e., the negative penalized loss function -$\ell_{\mathbf{r}}(\mathbf{w})$) can be thought as a variational approximation to the posterior of the deep Gaussian process model corresponding to our DNN model. (See Section S4 and [10] for further details.)

Note that dropout is applied only for parameter estimation, not prediction. In other words, once the parameter $\hat{\mathbf{w}}$ is estimated by minimizing the loss function in (4.3) the predictor $\hat{\boldsymbol{\theta}}^*$ is computed by the original model in (3.9) not (4.1). An exception for this rule is when the MC dropout approach is applied [10] (see Section S4 for details). Model fitting using the penalized loss function in (4.3) can be done through a gradient descent algorithm, which is described in Section S1.

4.2. Uncertainty Quantification Using Quantile Regression. The formulation in (3.1) suggests that uncertainty quantification for the estimated input parameter $\boldsymbol{\theta}^*$ can be essentially boiled down to the problem of finding the prediction interval for the fitted DNN emulator $\hat{\boldsymbol{g}}$. However, the highly complicated structure of $\hat{\boldsymbol{g}}$ and a large number of parameters in $\mathbf{w}$ make classic approaches to finding a prediction interval for $\boldsymbol{\theta}^*$ computationally prohibitive. For example, the asymptotic variance based on information matrix [40] cannot be computed because it requires inverting an $n_w \times n_w$ matrix and the total number of parameters n_w is typically hundreds of thousands or more. Similarly, a fully Bayesian inference (as mentioned in [27]) is not applicable either because it is not possible to fully explore the n_w-dimensional parameter space using Markov Chain Monte Carlo (MCMC).

To overcome the computational limitation, we propose a quantile regression-based approach. Quantile regression has been used to construct prediction intervals for highly complex prediction models such as the random forest [25, 44]. The last equation in (3.9) suggests that the last layer of our DNN model can be viewed as a linear mean regression model between the response variable $\boldsymbol{\theta}^*$ and the extracted feature $\boldsymbol{\lambda}^{(L)}$ up to the Lth layer, and, consequently, the predicted mean of $\boldsymbol{\theta}^*$ is

$$\hat{\boldsymbol{\theta}}^* = \mathbf{W}^{(L)}\boldsymbol{\lambda}^{(L)} + \mathbf{a}^{(L)}.$$

A similar observation on DNN as a linear model with basis functions can be also found in [24] and [41]. Instead, a predicted τth quantile of $\boldsymbol{\theta}^*$, denoted by $\hat{\boldsymbol{\theta}}^*_\tau$, can be obtained by quantile regression:

$$\hat{\boldsymbol{\theta}}^*_\tau = \mathbf{W}^{(L)}_\tau\boldsymbol{\lambda}^{(L)} + \mathbf{a}^{(L)}_\tau,$$

where $\mathbf{W}^{(L)}_\tau$ and $\mathbf{a}^{(L)}_\tau$ are the regression quantiles for a pre-specified target quantile $0 < \tau < 1$. The prediction limits are given as lower and upper tail quantiles such as the 0.025th ($\tau = 0.025$) and the 0.975th ($\tau = 0.975$) quantiles, that is $\left[\hat{\boldsymbol{\theta}}^*_{0.025}, \hat{\boldsymbol{\theta}}^*_{0.975}\right]$. Since the overall sample size N is typically thousands or larger (see Sections 5 and 6 below), these tail quantiles are reliably estimable. As noted at the end of Section 4.1 the upper and lower limits are computed without applying dropout. In addition to the interval estimates, we can also find the median estimate $\hat{\boldsymbol{\theta}}^*_{0.5}$, a more robust estimate for $\boldsymbol{\theta}^*$ than the mean, and use it as the point prediction.

We formulate the cost function to optimize based on (4.1) instead of (3.9) to apply dropout to this procedure. We propose to estimate the τth regression quantiles $\mathbf{W}_\tau^{(L)}$ and $\mathbf{a}_\tau^{(L)}$ by conducting quantile regression only in the last layer of the DNN model in (4.1),

$$\hat{\boldsymbol{\theta}}^*_\tau = \mathbf{W}_\tau^{(L)}\boldsymbol{\lambda}^{(L)} * \mathbf{r}^{(L)} + \mathbf{a}_\tau^{(L)} \tag{4.4}$$

while fixing the rest of the estimated parameters in the model at their optimal values obtained by minimizing the cost function (4.2). Therefore, the proposed quantile regression-based approach can be viewed as adding one more step in the end after completing the analysis in Section 4.1 to construct a prediction interval of $\boldsymbol{\theta}^*$. (See Section S3 for rationale behind this.)

Following the standard quantile regression procedure, we obtain the τth regression quantile estimate for $\mathbf{W}_\tau^{(L)}$ and $\mathbf{a}_\tau^{(L)}$ by minimizing the following cost function:

$$\mathcal{L}_{\mathbf{r}}(\mathbf{W}_\tau^{(L)}, \mathbf{a}_\tau^{(L)}) = \sum_{i=1}^{N} \left(\hat{\boldsymbol{\theta}}^*_{\tau,\mathbf{r},i} - \tilde{\boldsymbol{\theta}}_i\right)^T \left[\tau \mathbf{1}_{d_\theta} - \boldsymbol{I}\left(\hat{\boldsymbol{\theta}}^*_{\tau,\mathbf{r},i} - \tilde{\boldsymbol{\theta}}_i < \mathbf{0}\right)\right] \tag{4.5}$$

where $\mathbf{1}_{d_\theta}$ is a d_θ-dimensional vector of 1's and $\boldsymbol{I}\left(\hat{\boldsymbol{\theta}}^*_{\tau,\mathbf{r},i} - \tilde{\boldsymbol{\theta}}_i < \mathbf{0}\right)$ is a multivariate indicator function whose jth element is 1 if the jth element of $\hat{\boldsymbol{\theta}}^*_{\tau,\mathbf{r},i} - \tilde{\boldsymbol{\theta}}_i$ is less than 0 or 0 otherwise for $j = 1, \dots, d_\theta$. The estimated quantile $\hat{\boldsymbol{\theta}}^*_{\tau,\mathbf{r},i}$ is defined as

$$\hat{\boldsymbol{\theta}}^*_{\tau,\mathbf{r},i} = \hat{\boldsymbol{g}}_{\tau,\mathbf{r}}\left(\tilde{\mathbf{Y}}_i\right)$$

and $\hat{\boldsymbol{g}}_{\tau,\mathbf{r}}$ is the deep learning-based approximation function constructed by replacing the mean regression $\hat{\boldsymbol{\theta}}^* = \mathbf{W}^{(L)}\boldsymbol{\lambda}^{(L)} * \mathbf{r}^{(L)} + \mathbf{a}^{(L)}$ with the quantile regression (4.4).

5. Simulation Study.

In this section, we verify the performance of our proposed DNN and quantile regression-based method (DNN-Q henceforth) and compare it with three other approaches through a simulation study using a synthetic computer model output and observational data.

The first method to be compared is a DNN-based method that shares the framework introduced in Sections 3 and 4, except for the uncertainty quantification method described in Section 4.2. For uncertainty quantification, this method employs MC dropout, an existing standard uncertainty quantification method for DNN [10] based on variational Bayes approximation. We call this method DNN-MC henceforth. (See Section S4 for details.)

The second method to be compared is an inverse model-based approach that shares the same framework in Section 3.1, but finds the estimated inverse function $\hat{\boldsymbol{g}}(\cdot)$ using the random forest. The random forest-based calibration approach has not been introduced in the literature before, but we compare our method to this approach to demonstrate that DNN provides a better way to build $\hat{\boldsymbol{g}}(\cdot)$ than the random forest, which is also widely used as a general purpose function approximator. We call this method RF-Inv henceforth. (See Section S5 for details.)

The third method to be compared is the standard forward model-based calibration method explained in Section 2.1. Two different versions of the forward model-based framework are included in our comparison: GP-Fwd, which employs a Gaussian process emulator [28] to approximate the forward model $Y(\theta)$ and Bayesian inference to infer the best parameters $\boldsymbol{\theta}^*$ [22], and True-Fwd, which directly uses $Y(\theta)$ for calibration as in Equation (2.1) without emulation. The comparison with GP-Fwd and True-Fwd will tell us how much the difficulties in emulation affect the calibration performance. (See Section S6 for details.)

5.1. Synthetic Model Outputs and Observational Data. By following the usual way of conducting simulation studies in the calibration method literature [6, 19], we generate synthetic model runs and observational data and try to learn the true input parameter settings for the synthetic observations using the synthetic model runs. To this end, we first train the statistical emulators (either for the forward or inverse relationship) using the synthetic model runs and apply it to recover the parameter values for the synthetic observations. We compare the performance of all five methods in recovering the true input parameter settings for synthetic observations.

We generate synthetic model outputs that have similar characteristics as the model outputs in Section 6. For $p = 480$, time points $t = 1, \ldots, 480$, the model output $Y(\boldsymbol{\theta}, t)$ is defined as follows:

$$Y(\boldsymbol{\theta}, t) = 0.3 + \frac{\theta_1 + 0.3}{\sqrt{2\pi(\theta_3 + 0.1)}} \exp\left[-\frac{(u_t - \theta_2 + 0.5)^2}{\theta_3 + 0.1}\right] \tag{5.1}$$

where $u_1, \ldots, u_{480}$ are equally spaced points starting from -2 to 2, $\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3]^T$ is a vector of the input parameters that governs how the synthetic model output behaves. The whole model output at a given input parameter setting $\boldsymbol{\theta}$ can be denoted as $\mathbf{Y}(\boldsymbol{\theta}) = [Y(\boldsymbol{\theta}, 1), \ldots, Y(\boldsymbol{\theta}, 480)]^T$. As shown in Figure 2, the synthetic model outputs are smooth curves on the interval $[0, 480]$ with a single peak. The first parameter θ_1 controls the overall scale of the output, the second parameter θ_2 controls the location of the peak, and the third parameter θ_3 controls the overall dispersion of the curve. Based on this model, we generate training and test data sets as follows:

- *Synthetic Model Runs for Training Data*: We assume that the synthetic model runs are obtained at $n = 200$ design points, $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{200}$, sampled using the improve Latin Hypercube design method [3] from $[0, 1]$ intervals for all parameters. These model runs are assumed to be given to the user for creating the inverse emulator $\hat{\boldsymbol{g}}(\cdot)$ (in DNN-Q, DNN-MC, and RF-Inv) or the forward emulator $\boldsymbol{\eta}(\cdot)$ (in GP-Fwd), depending on the method. The generated model runs are illustrated in Figure 2. We use these synthetic model runs to build the inverse emulator $\hat{\boldsymbol{g}}(\cdot)$ in the DNN-Q, DNN-MC, and RF-Inv methods, or the forward emulator $\boldsymbol{\eta}(\cdot)$ in GP-Fwd method.
- *Synthetic Observations for Test Data*: We also generate 50 model runs for randomly sampled parameter settings from $[0.05, 0.95]$ intervals for all three parameters and build synthetic observations based on them. This choice is to mimic the situation that the true values $\boldsymbol{\theta}^*$ are not too close to or outside of the boundaries of the design points for the synthetic model runs. For each of these test cases we create 30 different synthetic observations by superimposing different synthetic data-model discrepancies ($\boldsymbol{\delta}$) generated from GPs with the squared exponential covariance function, whose covariance parameters are obtained by simple random sampling. For each of the $50 \times 30 = 1,500$ generated discrepancies, the range is sampled from $[50, 250]$ to create discrepancy patterns that operate in various time scales. The sill is sampled from $[0.005^2, 0.045^2]$ to create synthetic errors with various overall discrepancy scales, and the nugget parameter is sampled from $[0.0005^2, 0.0010^2]$ to

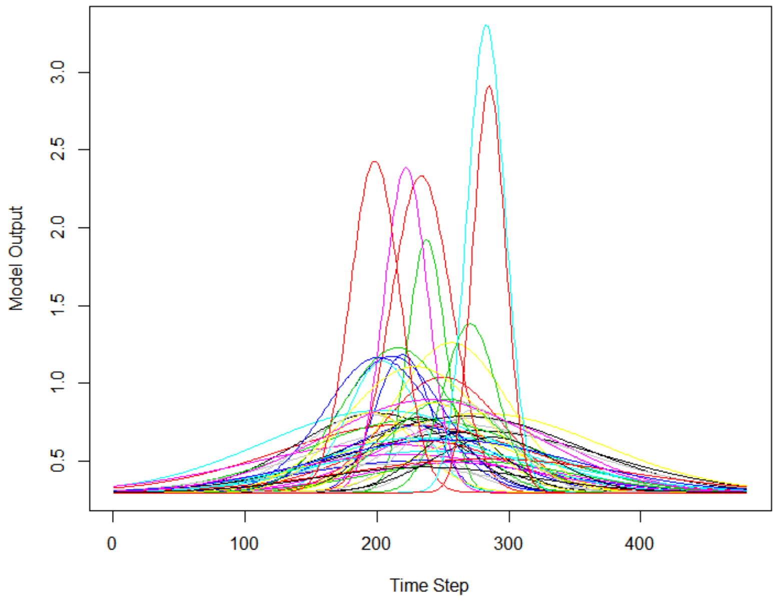

FIG. 2. *Example 50 simulation model runs randomly sampled out 200 model runs described in Section 5.1. Random sampling is done for better readability of the figure. Each model output is a time series with 480 time steps, which is of the same length as the model output and observational data in the WRF-Hydro example described in Section 6.*

add independent noise that is much smaller than the time dependent component to make it similar to the application problem in Section 6. As a result, we have 1,500 different scenarios for $\mathbf{Z}$ with various true $\boldsymbol{\theta}^*$ values and synthetic discrepancies from various types of $\boldsymbol{\delta}$. These test data sets with various properties of $\mathbf{Z}$s will be used to compare the model performance in Section 5.3.

Out of these 1,500 scenarios for synthetic data, we use only 50 scenarios (randomly selected one for each input parameter setting) to measure the performance of the GP-Fwd approach because the computational cost for this method is too expensive to apply it to all 1,500 scenarios, due to the need to run a long MCMC chain for each of these 1,500 scenarios which requires about 15,000 hours of computing time in total. We can, however, easily apply DNN-Q, DNN-MC, and RF-Inv methods to all these 1,500 test scenarios because estimating the input parameters for any given observational data is computationally cheap once the inverse model $\hat{\boldsymbol{g}}(\cdot)$ is constructed in these approaches.

5.2. Implementation and Computation Details. To implement the inverse model-based approaches, DNN-Q, DNN-MC, and RF-Inv, we first create contaminated model outputs $\tilde{\mathbf{Y}}_1, \ldots, \tilde{\mathbf{Y}}_N$ by following the procedure described in Section 3.3. These contaminated model runs serve as training data set to build the inverse emulator $\hat{\boldsymbol{g}}(\cdot)$. We generate $6,000$ realizations of simulated data-model discrepancy $\boldsymbol{\delta}_{ij}$ $(i = 1, \ldots, 200, j = 1, \ldots, 30)$ from GP models so that we have $n_d = 30$ contaminated model outputs for each of the 200 input parameter settings for the synthetic model runs in the training data. The sill (κ), range (ϕ), and nugget (ζ) parameters are sampled from random Latin hypercube design. The range for ϕ is set to be $[10, 300]$ (roughly from 2% to 60% of the total time interval) so that the DNN model is exposed to various types of discrepancy patterns as discussed in Section 3.3. The sill is sampled from $[0.001^2, 0.05^2]$ and the nugget parameter is sampled from $[0.0001^2, 0.0015^2]$, assuming those intervals reflect the user's vague guess on the variances of the time-dependent component and the independent component. This choice merely assumes that the assumed ranges for the GP parameters cover the true GP parameter values used to create synthetic observation $\mathbf{Z}$ for each test case – the sampled values used to generate the contaminated model outputs $\tilde{\mathbf{Y}}_1, \ldots, \tilde{\mathbf{Y}}_{6000}$ do not exactly match any of the GP parameter values used to generate the synthetic observations (i.e. the 1,500 test cases) in Section 5.1.

For both DNN-Q and DNN-MC methods, we use $L = 4$ hidden layers for the fully-connected layer part and input dimension of $d_x = 5$ (i.e. the present and four previous time steps are used as the input vector $\mathbf{x}_t$). Increasing the number of hidden layers beyond four for our simulation data does not improve the results and unnecessarily increases the overall computing time. The results are also quite robust against the choice of d_x and increasing d_x does not improve the estimation performance. We fit the model by minimizing the penalized loss function in (4.3) based on $N = 4,800$ contaminated model runs using the SGD described in Section S1. The remaining $1,200$ contaminated model outputs are used as a validation data set to determine the two tuning parameters p_{keep} and ξ. As mentioned in Section S1, we use an early stopping rule (Chapter 7 in [14]) by terminating the SGD algorithm if the prediction accuracy in the validation data set does not improve for 50 iterations. Implementation of RF-Inv also follows a similar procedure with the same 4,800 contaminated outputs for training and 1,200 for validation. In all these three methods, the tuning parameters are selected to minimize the RMSE averaged over all three parameters in the validation data. For DNN-Q and DNN-MC methods the considered values for ξ are $\{2000, 2500, 3000, 3500, 4000\}$ and the considered values for p_{keep} are $\{0.96, 0.97, 0.98\}$ in the tuning. After trying all possible combinations of these candidate values for ξ and p_{keep}, we have chosen $\xi = 2000$ and $p_{keep} = 0.96$ for DNN-Q and $\xi = 2500$ and $p_{keep} = 0.97$. Values outside of these ranges for either of the tuning parameters result in less accurate point estimates or undercoverage of the interval estimates. For RF-Inv we use 10,000 trees and 100 splits because increasing the number of trees or the number of tried splits does not improve the estimation accuracy. When computing the point estimate, we use the median prediction for DNN-Q and the ensemble mean for DNN-MC and RF-Inv.

For GP-Fwd method, we build an emulator for the synthetic model output based on 200 original model runs (without contamination) using a GP with the standard separable covariance structure for the input parameter effect and time dependence effect, each of which is modeled by an exponential covariance function following the general recommendation for choosing the covariance function in the literature when the smoothness parameter is not known [34]. (See Section S6.1 for further details on the emulation step of GP-Fwd.) Then, calibration is done via Bayesian inference based on the forward model in (2.1), with the constructed emulator $\boldsymbol{\eta}(\boldsymbol{\theta})$ in place of $\mathbf{Y}(\boldsymbol{\theta})$. The discrepancy $\boldsymbol{\delta}$ is also modeled through a GP with an exponential covariance function. We use weakly informative inverse gamma priors for the covariance parameters for $\boldsymbol{\delta}$ whose modes are at the true assumed values for the covariance parameters. This unrealistic assumption gives some advantages to the GP-Fwd model. When computing the point estimates, we have used the median of the posterior sample. True-Fwd uses the same implementation strategy as GP-Fwd except that $\mathbf{Y}(\boldsymbol{\theta})$

is directly used in calibration. (See Section S6.2 for further details on the calibration step of GP-Fwd and True-Fwd).

The overall computing time for building the inverse emulator $\hat{\boldsymbol{g}}$ takes about one hour for DNN-Q and 30 minutes for DNN-MC using **TensorFlow** and **Keras** libraries in Python. The computation is accelerated by parallel computing using 8 CPU cores (Xeon E5-2680 v4 2.40GHz). This computing time can be further reduced by utilizing GPU computing, but we did not pursue such speed-up because the current computing scheme is fast enough for our purpose. The longer computing time for DNN-Q is due to the need to refit the model three more times to find the DNN for the 0.025th, 0.975th, and 0.5th (median) quantiles. For the GP-Fwd approach we run MCMC for 200,000 iterations for each test case to obtain a well-mixed chain. The overall computing time for the entire chain is about 10 hours, which is much slower than the DNN-based approaches, as expected.

5.3. Results. We implement the five compared methods DNN-Q, DNN-MC, RF-Inv, GP-Fwd, and True-Fwd based on the generated synthetic model runs and observational data. We use five different metrics to compare the performance of different methods: the bias, the standard error (SE) the root mean square error (RMSE), the average length and the empirical coverage of the 95% interval estimates for $\boldsymbol{\theta}^*$. The formal definition of these metrics are given in S7. The comparison results for the five methods are summarized in Tables 1 and 2.

In Table 1, we compare the performance of the three inverse model-based approaches, DNN-Q, DNN-MC, and RF-Inv, for the 1,500 test cases. The results show that all three methods provide decent point predictions with small biases and RMSEs for the test data set. For all three methods, most of the RMSE can be attributed to the standard error (SE). Both DNN-based approaches have comparable RMSEs, lower than that of RF-Inv for all three parameters. In terms of uncertainty quantification through interval estimates, DNN-Q yields empirical coverages that are close to the nominal confidence level for all three parameters for the test cases. DNN-MC yields much shorter prediction intervals than the other two methods but has notable undercoverage issues for the first (0.803) and second parameters (0.893), which is often expected for a variational Bayes approximation. RF-Inv method shows notable undercoverage (0.823) for the first parameter. Moreover, for all three parameters, RF-Inv method leads to much wider average interval lengths for all three parameters compared to DNN-Q. Overall DNN-Q shows the most stable performance without showing any notable undercoverages and with notably shorter prediction intervals than RF-Inv.

In Table 2, we compare the performance of DNN-Q, GP-Fwd, and True-Fwd based on 50 selected cases out of the 1,500 test cases, since applying GP-Fwd and True-Fwd to all 1,500 test cases is computationally too expensive due to the need of running a long MCMC chain for each case. The details on how these 50 cases are selected is described in Section 5.1. As we have seen in Table 1, DNN-Q method provides accurate point estimates and sound uncertainty quantification for all three input parameters. On the contrary, GP-Fwd results in overly dispersed prediction intervals for the first parameter (that cover almost the entire parameter range $[0, 1]$) and notable biases and undercoverage of the prediction intervals for the second and third parameters. True-Fwd leads to a comparable performance to DNN-Q in point estimation, yielding similar Bias, SE, and RMSE values, but the length of the prediction intervals are overall too narrow to result in empirical coverage values that are notably lower than the nominal coverage. This result highlights the advantage of our

TABLE 1
Simulation Study Results for all Test Cases

Parameter	Method	Bias	SE	RMSE	PI Length†	PI Coverage‡
θ_1	DNN-Q	-0.002	0.056	0.056	0.204	0.923
	DNN-MC	0.000	0.053	0.053	0.116	0.803
	RF-Inv	0.018	0.092	0.094	0.260	0.823
θ_2	DNN-Q	-0.005	0.043	0.043	0.131	0.933
	DNN-MC	-0.002	0.042	0.042	0.106	0.893
	RF-Inv	-0.007	0.051	0.051	0.218	0.948
θ_3	DNN-Q	-0.001	0.033	0.034	0.106	0.931
	DNN-MC	0.002	0.030	0.030	0.108	0.954
	RF-Inv	-0.001	0.048	0.048	0.194	0.957

†: Average Length of 95% Prediction Interval.
‡: Empirical Coverage of 95% Prediction Interval.

TABLE 2
Simulation Study Results for Selected Cases for GP-Fwd

Data Set	Method	Bias	SE	RMSE	PI Length†	PI Coverage‡
θ_1	DNN-Q	0.006	0.046	0.046	0.208	0.980
	GP-Fwd	-0.013	0.248	0.248	0.936	1.000
	True-Fwd	-0.007	0.053	0.053	0.091	0.860
θ_2	DNN-Q	0.004	0.044	0.044	0.128	0.940
	GP-Fwd	-0.045	0.187	0.192	0.071	0.280
	True-Fwd	-0.002	0.041	0.041	0.054	0.820
θ_3	DNN-Q	0.002	0.031	0.031	0.109	0.980
	GP-Fwd	0.076	0.193	0.207	0.098	0.300
	True-Fwd	0.000	0.036	0.036	0.049	0.520

†: Average Length of 95% Prediction Interval.
‡: Empirical Coverage of 95% Prediction Interval.

approach over the forward model-based framework, as our approach yields a similar point estimation performance as True-Fwd and a better uncertainty quantification performance than True-Fwd even without direct access to the true simulation model itself. Moreover, the results show us that, while the difficulty in emulation is a significant factor in poor performance of the forward model-based method, the forward model-based method can still notably suffer from inferential issues without emulation due to the identifiability between the input parameter effect and the data-model discrepancy. Our method outperforms both approaches as it does not suffer from either the emulation errors or the identifiability issue. If we know the discrepancy parameters with a high confidence and impose strong priors accordingly, GP-Fwd and True-Fwd may suffer less from inferential issues, but assuming that the form of discrepancy is exactly known is highly unrealistic in practice.

Another important limitation of GP-Fwd is the difficulty of building an accurate emulator. The emulation performance evaluation described in Section S6.1 shows that the Gaussian process emulator does not provide a satisfactory prediction accuracy for the test cases in this emulation problem. Although the overall RMSE on the test data seems to be reasonably low (0.094), Figure S1 comparing the original and the predicted values shows that the emulator cannot accurately capture the trends in model runs with higher output values. The 95% prediction intervals from the emulator achieves an empirical coverage of 0.990, well above the nominal confidence level, but at the expense of overly wide intervals (with average length of 0.787) compared to the variation of output values show in Figure 2. One might be able to improve the emulation performance by incorporating a more complicated (and potentially non-stationary) dependence structure in the Gaussian process emulator model, but such added complexity may cause computational and inferential challenges.

As an alternative emulation model we have tried the generalized additive model (GAM) with the first and second order terms given by thin-plate splines using **mgcv** in R as well, while treating the three input parameters (θ_1, θ_2, and θ_3) and the time step (t) as the predictors and the individual model output $Y(\boldsymbol{\theta}, t)$ as the response. However, their prediction accuracy and uncertainty quantification performance were not better than the GP emulator, with RMSE of 0.159 on the test data and the empirical coverage of 10% for the 95% prediction intervals. The GAM model also has trouble in accurately predicting the model runs with higher output values.

Table 3
Simulation Study Results for Different Discrepancy Magnitudes

Parameter	Max Sill*	Bias	SE	RMSE	PI Length†	PI Coverage‡
θ_1	0.045	-0.002	0.056	0.056	0.204	0.923
	0.090	-0.010	0.091	0.092	0.339	0.953
	0.180	-0.018	0.141	0.142	0.470	0.919
θ_2	0.045	-0.005	0.043	0.043	0.131	0.933
	0.090	-0.007	0.070	0.070	0.227	0.971
	0.180	-0.003	0.108	0.109	0.330	0.943
θ_3	0.045	-0.001	0.033	0.034	0.106	0.931
	0.090	-0.005	0.051	0.051	0.181	0.953
	0.180	-0.009	0.076	0.076	0.263	0.961

$*$: Maximum value of the sill parameter for the simulated discrepancies
†: Average Length of 95% Prediction Interval.
‡: Empirical Coverage of 95% Prediction Interval.

The last aspect that we discuss is how the magnitude of the discrepancy (represented as the sill parameter for the assumed discrepancy in this study) is related to the length of the prediction intervals from our DNN-Q approach. It is also of interest whether the uncertainty bounds computed by DNN-Q is well calibrated (i.e., yielding an empirical coverage close to the nominal coverage) for a larger discrepancy magnitude. We have tried two additional cases with larger assumed discrepancies: one with an assumed sill range of $[0.001^2, 0.100^2]$ for the training data and $[0.005^2, 0.090^2]$ for the test data, and the other with an assumed sill range of $[0.001^2, 0.200^2]$ for the training data and $[0.005^2, 0.190^2]$ for the test data. The ranges for the range and nugget parameters were not changed from the original setting for the simulation study. The results show that larger the magnitude of the discrepancy, wider the width of the prediction intervals for all three parameters, but the empirical coverage are all reasonably close to the nominal coverage. The RMSE and the standard errors become larger as the magnitude of the discrepancy gets larger, but most of the observed RMSEs can be attributed to the standard errors in all cases and hence we can conclude that there are no notable biases caused by the larger discrepancies.

This manuscript is for review purposes only.

6. Application to WRF-Hydro Model.

In this section, we apply our proposed DNN-Q method to the problem of calibrating WRF-Hydro [12], the hydrologic extension of WRF model to demonstrate that our method can be used to calibrate a highly complicated computer model and provide useful information about the input parameter uncertainty and the data-model discrepancy. For performance comparison we also apply GP-Fwd calibration method discussed in Section 5. The WRF-Hydro model provides an innovative way to simulate the entire water cycle (surface and sub-surface runoff, and channel routing) by coupling a land surface component and high-resolution hydrologic components. It contains a large number of uncertain parameters that need to be properly tuned for realistic simulations [38].

The observational data used for calibration are collected by the United States Geological Survey (USGS). The objective is to find the best input parameter setting for simulating the streamflow at Iowa River at Wapello, IA (USGS ID#05465500), and the relevant model output and observational data are hourly time series for the same time period.

We use a perturbed physics ensemble with 400 WRF-Hydro model runs for 14 varied input parameters because running a simulation run for the WRF-Hydro model is computationally expensive. Running one simulation run requires a few hours of computing time using hundreds of CPUs. These 14 parameters are found to be the most important ones in terms of affecting the hydrograph in the Midwest region by a previous study [38]. The input parameter settings for the ensemble were determined by Latin hypercube sampling with maximin criteria [33]. All of the 400 model runs are used for both DNN-Q and GP-Fwd; for DNN-Q, we generate the contaminated outputs based on these 400 runs and use them for training the DNN model, and for GP-Fwd, we build a GP emulator based on these 400 runs.

The model ensemble has 14 varied parameters but we calibrate only five of them, as the other parameters are not relevant to the terrain types of the target site. The five parameters include: Manning's roughness coefficient for channel type #5 (MANN5) and overland roughness control factor (OVN#), which control the hydrograph shape and the timing of the peaks; deep drainage (SLOPE), infiltration-scaling parameter (REFKDT), and saturated soil lateral conductivity (REFDK), which control the total water volume. We therefore calibrate only these five relevant parameters using DNN-Q method. The actual ranges for these parameters are defined as $[0.02, 1]$ for MANN5; $[10^{-4},1]$ for SLOPE; $[10^{-8}, 10^{-5}]$ for REFDK; $[0.01, 5]$ for REFKDT, $[0.1,10]$ for OVN#, based on their physical indications following a previous study [32]. The tuning parameters for DNN-Q are chosen in the same way as in Section 5 and $\xi = 2000$ and $p_{keep} = 0.96$ are selected.

The simulated and observed time series are the instantaneous water volume (feet^3/sec) of streamflow for hourly intervals recorded from April 9th to 28th in 2013, having 480 time steps in total. Both the model output and the observational data are instantaneous streamflows recorded at hourly resolution and hence directly comparable. The overall model runs and observational data are shown in Figure 3a. This period had a major precipitation event in the area (highlighted with a red box in Figure 3a) and hence provides useful information on input parameters relevant to modeling streamflows. Figure 3b shows the best 20 runs in terms of the timing of the streamflow and the best 20 runs in terms of the magnitude of the streamflow along with the observational data during this period.

As shown in Figure 3b, the observational data do not resemble any of the model runs due to the notable mismatch in terms of the timing or the magnitude of the peak, suggesting that there are some notable data-model discrepancies. This suggests that

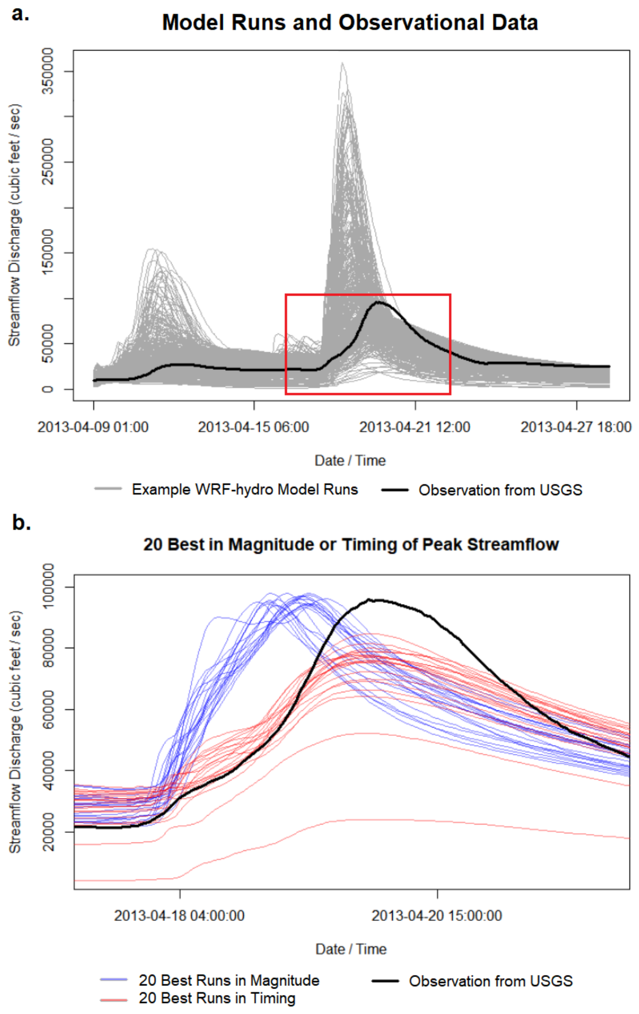

FIG. 3. *a. WRF-Hydro model outputs and observational data for Iowa River at Wapello, IA (USGS ID#05465500). The grey lines represent model runs at the 400 parameter settings in the perturbed physics ensemble. The red box highlights the major precipitation event occurred during the study period. b. The observational data and example model runs during the major precipitation event (the red-boxed period in panel a). The model runs shown as blue lines are the best 20 runs whose magnitudes of the peak streamflow are closest to the observational data. Those shown as red lines are the best 20 runs selected based on the closeness of the timing of the peak to the observational data. The plot shows that none of the existing model runs in the original perturbed physics ensemble can accurately capture the observed peak of the streamflow.*

our inverse model-based approach is useful to properly estimate the parameter values while accounting for the data-model discrepancy in this problem. The range, partial sill, and nugget parameters for the discrepancy term $\boldsymbol{\delta}$ are sampled from an improved Latin hypercube design. The sampling range for the discrepancy range parameter (ϕ) is set to be [10,300], as in the simulation study in Section 5 to train our DNN model based on discrepancy patterns with various time scales. The range for the nugget parameter (ζ) set to be [1,10] to reflect the fact that both the model output

TABLE 4
Calibration Results for WRF-Hydro by DNN-Q

	Method	Quantile	mann5	xslop1	refdk	refdkt	ovn#
Scaled	DNN-Q	0.025th	0.552	0.704	0.240	0.579	0.461
		Median	0.589	0.762	0.270	0.632	0.576
		0.975th	0.623	0.812	0.312	0.704	0.657
	GP-Fwd	0.025th	0.621	0.783	0.495	0.561	0.740
		Median	0.902	0.803	0.502	0.565	0.758
		0.975th	0.996	0.848	0.507	0.568	0.782
Non-Scaled	DNN-Q	0.025th	0.0640	0.704	2.42×10^{-6}	2.92	2.56
		Median	0.0670	0.761	2.72×10^{-6}	3.18	3.19
		0.975th	0.0697	0.811	3.13×10^{-6}	3.53	3.63
	GP-Fwd	0.025th	0.0695	0.783	4.96×10^{-6}	2.83	4.08
		Median	0.0919	0.803	5.03×10^{-6}	2.85	4.18
		0.975th	0.0942	0.847	5.08×10^{-6}	2.86	4.31

and the observational data show very smooth trends. To determine the range for the partial sill (κ) we have conducted some exploratory data analysis and found that the interquartile range for the mean squared error between the model output and the observational data range from 10,292 (ft^3/sec) to 21,670 (ft^3/sec). Loosely based on this observation, we set the range for the partial sill parameter to be $[5000^2, 15000^2]$ so that the lower bound is well less than 10,292, and 2×(the upper bound)=30,000 well exceeds 21,670.

Table 4 summarizes the calibration results with and without rescaling the values into $[0, 1]$ intervals, i.e. the 'non-scaled' values in the table are the parameter values estimated in the original scale used in the WRF-Hydro model. The 0.025th and 0.975th quantiles are computed to find the 95% interval estimates and the median values are computed to find the point estimates for the best input parameter values $\boldsymbol{\theta}^*$. The results show that the 95% credible interval from GP-Fwd for mann5 is much wider than the prediction interval from DNN-Q while they have a small overlap. For refdk, refdkt, and ovn#, the credible intervals from GP-Fwd are much narrower than the prediction intervals from DNN-Q and there are no agreements between the intervals from the two methods. The results here bear some resemblance to the simulation study results where the interval estimate from GP-Fwd is too wide for one parameter and too narrow for the others, and the interval estimates from the two methods often show some notable disagreement.

To understand how the difference in parameter estimates affects the outcome of simulation, we have run the WRF-Hydro model based on the calibration results from both approaches. For each method, we have generated 100 parameter samples within the ranges given by the interval estimates shown in Table 4 using a Latin Hypercube design. The resulting calibrated model runs obtained by running the WRF-Hydro model on those design points are shown in Figure 4. For both methods, compared to all of the model runs in the ensemble shown in Figure 3a, the parametric uncertainty in simulation is significantly reduced. The calibrated runs have better captured two important hydrologic quantities, the timing and the magnitude of the peak streamflow discharge (with a slight overestimation for the magnitude, though). The runs calibrated by DNN-Q show better performance in terms of capturing the time and magnitude of the peak, as well as how the streamflow increases while reaching to the peak. Note, however, there are still notable discrepancies before and after the peak surges in both results, which may be due to WRF-Hydro model deficiencies

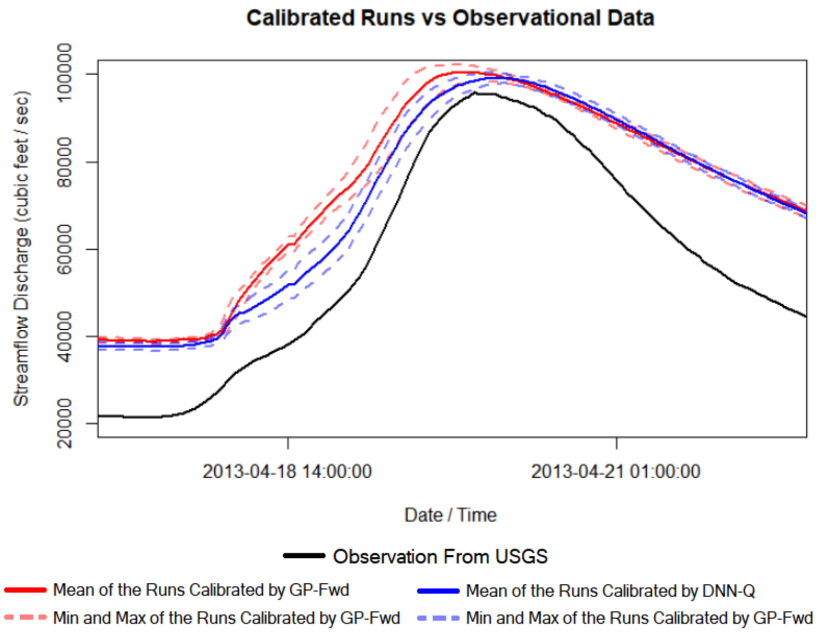

FIG. 4. *Calibrated WRF model runs at the estimated parameter settings by GP-Fwd (red) and DDN-Q (blue) compared to the observations from USGS (black) during the major precipitation event (red-boxed period in Figure 3a). The solid line represents the point-wise ensemble mean and the dashed lines represent the point-wise ensemble minimum and maximum. The runs calibrated by DNN-Q are closer to the observational data than those by GP-Fwd when the streamflow rapidly increases and reaches the maximum.*

in capturing certain hydrological processes that need to be improved or taken into account. The model runs shown here do not include estimated discrepancies. When simulating the streamflow for an unobserved period, adding a statistically estimated discrepancy trend can improve the simulation accuracy of both approaches. As a related note, the calibration result might be further refined by reflecting the fact that a large portion of the data-model discrepancy here can be attributed to the positive offset between the simulated and observed values. Such a sequential design approach can be an interesting future direction.

7. Summary and Future Directions. In this paper, we have proposed a new computer model calibration method using deep learning. The framework focuses on the case where the model output and observational data are in the form of time series, but the basic framework can be easily modified for other types of data, such as spatial data or spatio-temporal data, by substituting the LSTM feature extraction layers with convolutional layers ([14], Chapter 9) or convolutional LSTM layers [43]. Utilizing the feature extraction capacity of LSTM layers and the flexibility of fully-connected layers, our DNN-based method provides an accurate way to capture the inverse relationship between the model output and the input parameters. Using the learning with noise idea, we train a DNN emulator for inverse relationship that can efficiently filter out the effects from data-model discrepancy on input parameter estimation. This provides a viable solution to one of the long-standing issues in computer model calibration literature: non-identifiablity between the effects of input parameters and data-model discrepancy. Our framework also provides a way to quantify the uncertainty in parameter estimation in the form of interval estimates using quantile regression. This approach can be used to quantify the uncertainty in any DNN-based modeling problem, and, hence, has an implication beyond the problem of computer model calibration.

As per possible future extensions, perhaps the most important direction is to relax the stationarity assumption. Instead of assuming a vague stationary discrepancy distribution, it may be possible to define a mixture of non-stationary discrepancy models to better reflect possible model deficiencies in reality. Specifying such a discrepancy model will need to be guided by scientific knowledge on the specific problem at hand and hence is more suitable for application-oriented research. However, this direction also poses some interesting methodological questions as well–can a DNN model handle a highly uncertain non-stationary discrepancy? If the standard DNN framework has a trouble in handling highly uncertain discrepancy assumptions, can a more advanced model structure or training strategy such as a generative adversarial network [15] provide a solution?

Another possible direction is to modify our framework so that it can handle non-continuous data such as binary or count data. This requires generating non-continuous contaminated model outputs and hence some generalized linear model-type approach is needed. Similarly, it may be possible to formulate a DNN-based calibration method for temporally or spatially varying input parameters, which will require handling of high-dimensional response variables with complicated dependence structures in DNN modeling. All of these possible future developments have to be accompanied with development of a proper uncertainty quantification method through a statistical inference procedure that is specifically designed for particular distributional assumptions and variable types at hand.

Acknowledgments. This research is partially supported by the University of Cincinnati Charles Phelps Taft Research Center and the Ohio Super Computing Center (OSC). The authors are grateful to the anonymous reviewers for their valuable comments and suggestions.

References.

[1] G. An, *The effects of adding noise during backpropagation training on a generalization performance*, Neural Computation, 8 (1996), pp. 643–674.

[2] M. Bayarri, J. Berger, J. Cafeo, G. Garcia-Donato, F. Liu, J. Palomo, R. Parthasarathy, R. Paulo, J. Sacks, and D. Walsh, *Computer model validation with functional output*, Ann. Statist., 35 (2007), pp. 1874–1906.

[3] B. Beachkofski and R. Grandhi, *Improved distributed hypercube sampling*, in 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2002, p. 1274.

[4] C. M. Bishop, *Training with noise is equivalent to tikhonov regularization*, Neural Computation, 7 (1995), pp. 108–116.

[5] J. Brynjarsdóttir and A. O'Hagan, *Learning about physical parameters: The importance of model discrepancy*, Inverse Problems, 30 (2014), p. 114007.

[6] W. Chang, M. Haran, P. Applegate, and D. Pollard, *Calibrating an ice sheet model using high-dimensional binary spatial data*, J. Am. Statist. Assoc., 111 (2016), pp. 57–72.

[7] W. Chang, M. Haran, R. Olson, and K. Keller, *Fast dimension-reduced climate model calibration and the effect of data aggregation*, Ann. Appl. Stat., 8

(2014), pp. 649–673.

[8] W. Chang, M. Haran, R. Olson, and K. Keller, *A composite likelihood approach to computer model calibration using high-dimensional spatial data*, Statistica Sinica, 25 (2015), pp. 243–259.

[9] M. Chen, H. Jiang, W. Liao, and T. Zhao, *Efficient approximation of deep relu networks for functions on low dimensional manifolds*, in Advances in Neural Information Processing Systems, 2019, pp. 8172–8182.

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

[11] F. A. Gers and E. Schmidhuber, *LSTM recurrent networks learn simple context-free and context-sensitive languages*, IEEE Trans. Neural Netw., 12 (2001), pp. 1333–1340.

[12] D. Gochis, M. Barlage, A. Dugger, K. FitzGerald, L. Karsten, M. McAllister, J. McCreight, J. Mills, A. RafieeiNasab, L. Read, K. Sampson, D. Yates, and W. Yu, *The WRF-Hydro modeling system technical description, Version 5.0*, NCAR Technical Note, (2018).

[13] D. Gochis, W. Yu, and D. Yates, *The WRF-Hydro model technical description and user's guide, version 3.0. NCAR Technical Document*, NCAR Technical Document, (2015).

[14] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1, MIT Press, Cambridge, 2016.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, in Advances in Neural Information Processing Systems, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds., vol. 27, Curran Associates, Inc., 2014, https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[16] A. Graves and J. Schmidhuber, *Framewise phoneme classification with bidirectional LSTM and other neural network architectures*, Neural Networks, 18 (2005), pp. 602–610.

[17] M. Gu, J. O. Berger, et al., *Parallel partial Gaussian process emulation for computer models with massive output*, The Annals of Applied Statistics, 10 (2016), pp. 1317–1347.

[18] Y. Guan, C. Sampson, J. D. Tucker, W. Chang, A. Mondal, M. Haran, and D. Sulsky, *Computer model calibration based on image warping metrics: an application for sea ice deformation*, J. Agr. Biol. Envir. St., 24 (2019), pp. 444—-463.

[19] D. Higdon, J. Gattiker, B. Williams, and M. Rightley, *Computer model calibration using high-dimensional output*, J. Am. Statist. Assoc., 103 (2008), pp. 570–583.

[20] L. Holmstrom and P. Koistinen, *Using additive noise in back-propagation training*, IEEE Transactions on Neural Networks, 3 (1992), pp. 24–38.

[21] Z. Huang, W. Xu, and K. Yu, *Bidirectional lstm-crf models for sequence tagging*, arXiv preprint arXiv:1508.01991, (2015).

[22] M. Kennedy and A. O'Hagan, *Bayesian calibration of computer models*, J. R. Stat. Soc. Ser. B Stat. Methodol., 63 (2001), pp. 425–464.

[23] P. Koistinen and L. Holmström, *Kernel regression and backpropagation training with noise*, in Advances in Neural Information Processing Systems, 1992, pp. 1033–1039.

[24] P. L. McDermott and C. K. Wikle, *Deep echo state networks with uncertainty quantification for spatio-temporal forecasting*, Environmetrics, 30 (2019), p. e2553.

[25] N. Meinshausen, *Quantile regression forests*, Journal of Machine Learning Research, 7 (2006), pp. 983–999.

[26] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, *Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review*, International Journal of Automation and Computing, 14 (2017), pp. 503–519.

[27] N. G. Polson, V. Sokolov, et al., *Deep learning: A bayesian perspective*, Bayesian Anal., 12 (2017), pp. 1275–1304.

[28] J. Sacks, W. Welch, T. Mitchell, and H. Wynn, *Design and analysis of computer experiments*, Stat. Sci., 4 (1989), pp. 409–423.

[29] H. Sak, A. Senior, and F. Beaufays, *Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition*, arXiv preprint arXiv:1402.1128, (2014).

[30] J. M. Salter, D. B. Williamson, J. Scinocca, and V. Kharin, *Uncertainty quantification for computer models with spatial output using calibration-optimal bases*, J. Am. Statist. Assoc., 114 (2019), pp. 1800–1824.

[31] J. Schmidt-Hieber, *Nonparametric regression using deep neural networks with relu activation function*, arXiv preprint arXiv:1708.06633, (2017).

[32] D. T. Soong, C. D. Prater, T. M. Halfar, and L. A. Wobig, *Manning's roughness coefficient for Illinois streams: U.S. Geological Survey Data Series 668*, USGS, 2012.

[33] M. Stein, *Large sample properties of simulations using Latin hypercube sampling*, Technometrics, 29 (1987), pp. 143–151.

[34] M. Stein, *Interpolation of Spatial Data: Some Theory for Kriging*, Springer-Verlag, Berlin, 1999.

[35] C.-L. Sung, Y. Hung, W. Rittase, C. Zhu, and C. J. Wu, *A generalized gaussian process model for computer experiments with binary time series*, J. Am. Statist. Assoc., 115 (2020), pp. 945–956.

[36] R. Tuo and C. J. Wu, *Efficient calibration for imperfect computer models*, Ann. Statist., 43 (2015), pp. 2331–2352.

[37] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, *Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion*, Journal of Machine Learning Research, 11 (2010), pp. 3371–3408.

[38] J. Wang, C. Wang, V. Rao, A. Orr, E. Yan, and R. Kotamarthi, *A parallel workflow implementation for pest version 13.6 in high-performance computing for wrf-hydro version 5.0: a case study over the midwestern united states*, Geosci. Model Dev., 12 (2019), pp. 3523—-3539.

[39] Y. Wang, M. Huang, X. Zhu, and L. Zhao, *Attention-based lstm for aspect-level sentiment classification*, in Proceedings of the 2016 conference on empirical methods in natural language processing, 2016, pp. 606–615.

[40] H. White, *Some asymptotic results for learning in single hidden-layer feedforward network models*, J. Am. Statist. Assoc., 84 (1989), pp. 1003–1013.

[41] C. K. Wikle, *Comparison of deep neural networks and deep hierarchical models for spatio-temporal data*, J. Agr. Biol. Envir. St., 24 (2019), pp. 175–203.

[42] R. Wong, C. Storlie, and T. Lee, *A frequentist approach to computer model calibration*, J. R. Stat. Soc. Ser. B Stat. Methodol., 79 (2017), pp. 635–648.

[43] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, *Convolutional LSTM network: A machine learning approach for precipitation nowcasting*, in Advances in Neural Information Processing Systems, 2015, pp. 802–810.

[44] H. Zhang, J. Zimmerman, D. Nettleton, and D. J. Nordman, *Random forest prediction intervals*, Am. Stat., 74 (2020), pp. 392–406.