

Machine Learning Based Prediction and Analysis for Virtual Metrology Tasks in Long Batch Processes

Ritam Guha^{a,e}, Anirudh Suresh^{b,c}, Jared DeFrain^c and Kalyanmoy Deb^{d,e}

^aComputer Science and Engineering, Michigan State University, East Lansing, USA;

^bMechanical Engineering, Michigan State University, East Lansing, USA; ^cInformation Technology, Hemlock Semiconductor Operations LLC, Hemlock, MI, USA; ^dElectrical and Computer Engineering, Michigan State University, East Lansing, USA;

^eComputational Optimization and Innovation (COIN) Laboratory, Michigan State University, East Lansing, USA

COIN Report Number 2023009

Abstract

A long batch process typically runs for several hours to produce different process outcomes. During the entire duration of the process, several sensor data are recorded involving complicated non-linear dynamics among process constituents, which are difficult to model. The users are often interested in predicting the eventual process outcomes well before the completion of the process so that the process can be terminated in case the predicted outcome is not as desired or get a reasonable time estimate to stop the process for achieving the desired outcome. Virtual Metrology (VM), a virtual property estimation procedure, has gained importance over the years as a supporting tool to address this problem. In this paper, we have proposed a generalized VM pipeline including a deep-learning model that can be scaled to support high-dimensional input sensors and outputs. We have illustrated the applicability of the pipeline over data collected for a long process from the industry. The developed model is able to predict end-results with less than 10% error after about one-fifth of the total process-time. Critical sensors affecting prediction error are also identified. Moreover, we also modify the model to provide uncertainty of predictions, thereby making the overall prediction process pragmatic and trustworthy.

Keywords: metrology, virtual, simulation, prediction, DNN.

Introduction:

Modern batch processes are complex, involving a large number of inputs processed through a time-varying manner and finally producing resultant process outcomes as outputs. Such processes are resource consumptive and are usually monitored by a system of sensors measuring process indicators as a time series of sensor data. The processes are usually computationally expensive to be simulated during process operation due to a variety of reasons. First, the underlying dynamics of constituents are

nonlinear and complex to analyze. Second, there are internal uncertainties in the system which are difficult to include in exact computational procedures. Third, the multitude of inputs, sensors, and outputs makes the task non-scalable in practice. However, in these processes, usually, a number of recipe data (input as meta-parameters, time-varying sensor data for the entire duration of the process, and the outputs as target product at the end of the batch process) are available from the past execution of the process. It then becomes important to analyze the available data and develop a model of the outputs¹ as a function of inputs and partial or full sensor data. If such a model is developed, it can be used to predict outputs with the knowledge of inputs and a few initial time steps of sensor data. Such a task will be helpful to know the quality of outputs well before the batch process is completed, thereby saving time and effort and hence increasing productivity and revenue from the process.

In terms of the data analysis problem, the above problem becomes a modeling task:

$$\mathbf{y} = f(\mathbf{x}, \mathbf{s}^{1:t_b}), \quad (1)$$

in which \mathbf{x} is the input vector (defining the meta-variable of the batch process, such as raw materials, material type, machine number, etc.), \mathbf{s}^t is the sensor data vector at time-step t , and \mathbf{y} is the output vector at the end of the process (that is, \mathbf{y} is determined at the final time $t = T$). The model $f(\cdot)$ is to be determined by a data analysis procedure. The sensor data is utilized from $t = 1$ to an intermediate time $t = t_b (\leq T)$. It is interesting to note that the above model does not explicitly utilize the available sensor data from $t = t_b + 1$ to $t = T$, simply because at the time of deployment the above model must be applied at time $t = t_b$ and only sensor data from the beginning to $t = t_b$ are available. However, during the model development process, the entire sensor data is available, and can be used to enhance the modeling task better by predicting the sensor data from $t = t_b + 1$ to $t = T$ along with the outputs:

$$[\mathbf{y}, \mathbf{s}^{(t_b+1):T}] = f(\mathbf{x}, \mathbf{s}^{1:t_b}). \quad (2)$$

The above virtual metrology (VM) task is gaining popularity in industries due to extensive efforts in data recording and the availability of efficient machine learning procedures. This is a huge problem faced by practitioners in the industry and there is no generalized framework to solve this problem at a large scale. Moreover, every process will have its own associated sensor readings and recipes. It is hard to capture a single VM framework that can be generalized for multiple differing systems. It is also important to collect a significant amount of data from the system to learn the pre-existing relationships in the recipe and make efficient predictions.

In this paper, we propose a generalized machine learning-based framework, named VMNet, which can be easily extended to a large number of dimensions and process outputs. To illustrate the applicability of the proposed framework, we use proprietary process data from the industry. The proposed VMNet procedure has been developed and deployed to the industry with success. Besides developing the prediction

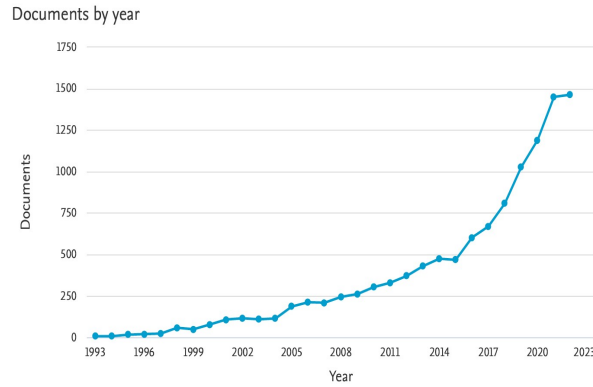
¹ In this paper, we have referred to the final product of the process as outputs, response, process outputs interchangeably.

model, we also use other machine learning methods to identify critical sensors affecting the prediction quality so that industries can improve the data collection from these specific critical sensors. Moreover, to make the deployment process more practical and reliable, we use a quantile-based machine learning approach to provide an estimate for the level of uncertainty in the prediction so that besides getting a prediction value, the user is also aware of the uncertainty of the prediction.

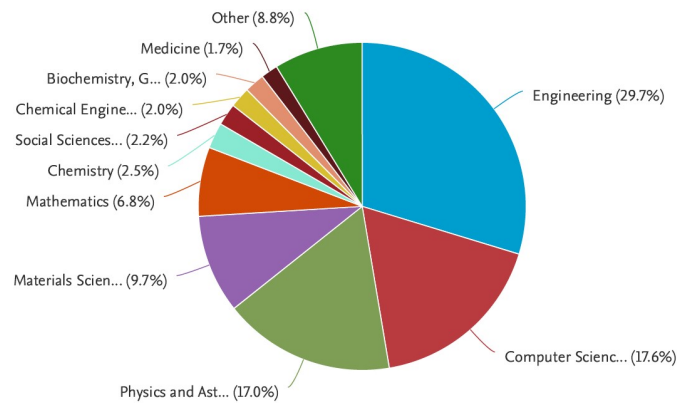
The remainder of the paper is organized as follows. Section 2 discusses some of the recent works in the domain of VM. A formal description of the problem is defined in Section 3. Some preliminary concepts required to understand the proposed process are described in Section 4. Section 5 illustrates the proposed ML framework for performing the VM. The results obtained for the proposed framework are mentioned in Section 6. For a more confident deployment process, uncertainty in the prediction is considered in Section 7. Finally, conclusions are drawn in Section 8.

Related Work:

VM has gained enormous interest in recent times. As a result, multiple researchers have focused on developing VM systems as supporting tools for batch processes. Figure 1a shows the number of publications on issues related to VM, whereas Figure 1b plots the division of publications coming from different subject areas using VM technology. From Figure 1a, we can clearly see that the number of publications on VM has risen sharply over the last decade. In this section, we are providing a brief overview of the existing VM tools and their limitations.



(a) Number of publications on VM over the years.



(b) VM Publication pie chart with respect to different subject areas.

Figure 1. Analysis of publications on VM according to Scopus [3].

Chang et. al [1] proposed a hybrid system consisting of a piece-wise linear NN and fuzzy NN, where the piece-wise linear NN is used to capture the drift in the process recipe over time and then the drift information is used by a fuzzy NN to predict silicon wafer outcome. The paper proposes a way to consider the drifts in the process parameters, but it becomes complicated to map the drifts coming from a huge number of process parameters to the final yield. The authors have divided each timeline into multiple clusters assuming that a deviation from a linear structure denotes a change in the process recipe which is not always true in complicated high-dimensional system dynamics. Moreover, the structure requires the process parameter readings for the entire timeline which is not always available, leading to limited applicability in deployment.

In 2009, Kang et. Al [2] proposed a framework for performing virtual metrology in semiconductor manufacturing and provided a direction for systematic analysis. The authors obtained the data from a Korean manufacturing company. The entire process was very specific to the problem under consideration and lacked generalization. The process had only 8 steps and the data had fixed dimensions in the processing, but it is not true in every situation. For example, some runs may take 50 hours to get completed, but there can be other runs that take 60 hours to get completed. The paper did not provide any procedure for dealing with these varying dimensionality of the data. Moreover, the prediction models that the authors have used are not scalable enough along the timeline. The addition of a recurrent structure in the prediction model would permit it to be easily scaled to the required time limit.

Hung et. al [4] used a radial basis function NN (RBFN) as a VM model to predict the Chemical Vapor Deposition (CVD) thickness. The input sensors are mapped to the hidden layers of the NN using fixed radial basis functions as the transfer functions. In this way, the authors have removed the iterative training process of the standard backpropagation-based NNs. Such a modification was done to the model to deal with the low quantity of data available for the training. Without backpropagation, it gets difficult to capture the proper dynamic relationships of the sensors using radial basis functions. Moreover, as the training data increases, the RBFN training complexity becomes very high. So, it is not completely scalable. The paper also does not deal with

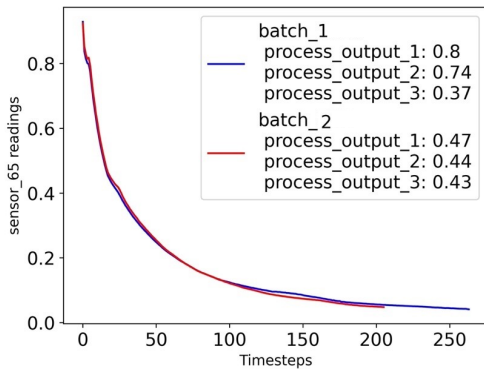
varying dimensions of input data along the timeline.

The proposed methodology is the most related to the work of Cheng et. al [5]. The authors of the paper have proposed an automatic VM framework for fabwide VM deployment and illustrated the automated VM refreshment concept on CVD tools. The proposed framework was impressive, but very complicated and lacked interpretability. In this paper, we have tried to simplify the pipeline which can be scaled as and when needed while helping the users interpret some of the basic functionalities of the model.

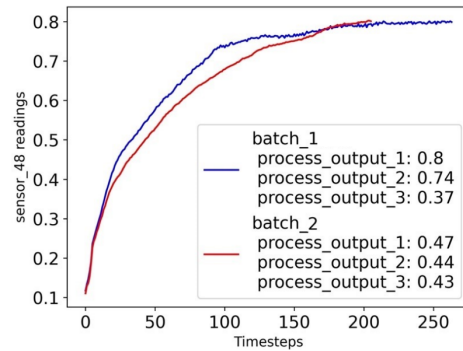
There are some other papers related to VM that might be interesting to the readers like the adaptive VM in [6], VM for run-to-run control [7], feedback control in VM [8], the reliance of VM system [9], and others [10–13], etc. Please note that most of the important concepts of VM have been incorporated into VMNet while keeping it as simplistic and generalized as possible. But every VM problem is different in nature and the framework should be adjusted to get the optimal performance.

Problem Definition:

A VM process consists of mainly two types of spaces: machine parameter space and process output or response space. Let the machine parameters space be defined as \mathcal{S} which mostly represents the sensor readings of the process and response space be represented as \mathcal{Y} . The goal of the VM procedure is to find a mapping: $f : \mathcal{S} \rightarrow \mathcal{Y}$. But in deployment conditions, we have access to only limited sensor readings because we do not know the exact number of time steps at which the process ends. So, at any point (say after t_b time steps), we are trying to observe how the response property will change after t_a additional time steps. Based on this analysis, the process will be stopped after a certain number of time steps. For this reason, in practice, the problem can be defined as finding the following mapping: $f : \mathcal{S}_{obs} \rightarrow \mathcal{S}_{sim} \rightarrow \mathcal{Y}$, where \mathcal{S}_{obs} and \mathcal{S}_{sim} are defined as the observed sensor readings and the simulated sensor readings, respectively.



(a) Readings of sensor 65 of two random batches in the dataset.



(b) Readings of sensor 48 of two random batches in the dataset.

Figure 2. Sensor readings of two different batches in the dataset.

In Figure 2, we illustrate an example of the sensor readings present in the dataset². Two different batches are stopped at two different timesteps. But we do not know if stopping the processes at these timesteps was optimal. Manually analyzing the processes and stopping them is tedious and not optimal. VM can help us decide on predicting an optimal stopping time. In deployment, the idea is to collect the process parameters for t_b timesteps and simulate the process for additional t_a ($\gg t_b$) timesteps and predict the process outcome at the final timestep ($t_a + t_b = T$). By varying t_a we can analyze the response for stopping at different timesteps and then we can select a suitable t_a that leads to a reasonable output. This is how the stopping time can be decided by taking help from VM. Moreover, we can notice that the two batches in Figure 3 have different sensor readings along the timeline. This indicates that there is a data drift observed in the dataset which can happen due to multiple reasons like recipe change, sensor modifications, data collection process updates, etc. Due to the data drift, the mapping problem becomes non-trivial and a generalized VM framework should be able to capture this drift to make efficient predictions.

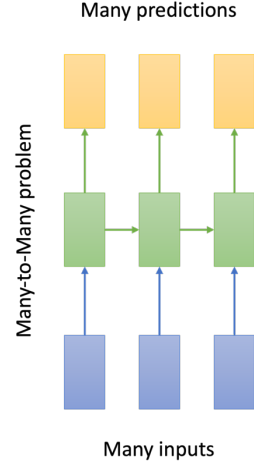


Figure 3. Example of a many-to-many sequential problem.

In ML literature, this kind of sequential problem is termed as a many-to-many problem [14,15], as the problem takes multiple inputs and predicts multiple outputs in a sequential fashion as shown in Figure 3. In the proposed framework, these mappings are performed using a deep neural network (DNN) based model architecture.

Preliminaries:

In this section, we have described a few preliminary concepts that might be useful to understand the working principle of the proposed framework.

Recurrent Neural Networks (RNNs):

After DNNs started gaining popularity as an important estimation tool, the researchers realized there is a need for introducing a recurrent structure to the NN for handling recurring estimations with dynamic dimensions of inputs. In 1982, Hopfield [16] proposed a preliminary version of recurrent neural network (RNN) known as Hopfield Network in which recurrent structure was introduced in the nodes. The modern unfolding structure in RNNs was introduced by Jurgen [17]. But even though RNNs can be recurrently unfolded to any number of time steps, it was unable to capture long-term dependencies. This problem of long-term dependency retrieval was solved by Long Short-Term Memory (LSTM).

² For data secrecy issue, we have normalized all sensor and response data within [0,1].

LSTM Structure:

Proposed by Hochritter et. Al ^[18] in 1997, Long Short-Term Memory (LSTM) was one of the breakthrough models in the domain of deep learning. A structure of a typical LSTM cell is shown in Figure 4. The most interesting component of an LSTM cell are the three gates that control the flow of information along the timeline namely: forget gate, input gate and output gate. It has two data lines passing through the cell like conveyor belts: cell state and hidden state. Suppose we are focusing on the prediction of t^{th} time step. We get the input x_t and the two states from the last time step: hidden state (h_{t-1}) and cell state (c_{t-1}). The first gate is the forget gate which determines how much of the previous data should be ignored. The forget state of the current prediction is denoted by f_t . It denotes a probability value that gets multiplied by the previous cell state. Then the input gate determines the amount of new information that should be introduced to the memory based on the current input and the input state is denoted by i_t . After performing some calculations, the new information is added to the cell state. Finally, we get the current cell state and the output gate decides the value which is suitable to be used as the output of the current state and that value becomes the current hidden state (h_t). This hidden state becomes the output for the current cell and both h_t and c_t are passed to the next time step. The goal of this introduction to LSTM was to give a brief intuition on how LSTM structure works, so we have not mentioned all the computations that are happening inside the cell.

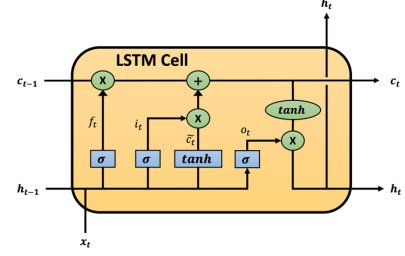


Figure 4. Pictorial representation of an LSTM cell.

Uncertainty Estimation:

When we are dealing with a real-life dataset, there can be two types of sources of uncertainty: data-based and model-based.

Data-based uncertainty ^[19] originates from irregularities in the dataset. The dataset could contain one-to-many mapping that might hinder the training of the model. For example, two runs in the training set could have identical sensor data in the first few timesteps but lead to two different outcomes. This ambiguity can severely hinder the training and testing of the deep learning model. The sensor readings could also contain noise that could not be eliminated while cleaning the data. The second type of uncertainty ^[20] could originate from the model itself. The model can learn some regions of the dataset better than others leading to good performance in some regions and poor performance in other regions. This could happen due to a variety of reasons such as bias in the dataset, initialization, and training mechanism.

In this study, we focus on the first type of uncertainty and aim to train a model that can learn the uncertainties along with the actual dataset. The method we use to learn the uncertainties is called Quantile Regression. One of the ways to train a model to learn quantile regression is by using pinball loss.

Pinball Loss:

Pinball loss ^[21,22] is a loss function used to train a model to learn quantile predictions. Let \hat{y} be the model prediction for q^{th} quantile of a particular quantity, whereas the true value for the quantity is y , then the pinball loss (also known as quantile loss) of the prediction can be calculated according to Equation (3).

$$\mathcal{L}_q(y, \hat{y}) = \begin{cases} q(y - \hat{y}), & \text{if } y \geq \hat{y}, \\ (1 - q)(\hat{y} - y), & \text{otherwise.} \end{cases} \quad (3)$$

Intuitively, the equation assigns more loss to low-quantile predictions if the predictions are higher than the original value and less loss to low-quantile predictions if the predictions are lower than the original value. This formulation helps low-quantile predictions to be low in value. A similar situation occurs with high-quantile predictions but the loss levels get altered for the kind of predictions.

Proposed Prediction Method:

In this section, we provide the entire description of the proposed VM framework. VMNet starts with data preparation where the input data is processed to follow a particular order. In this process, some of the outliers are identified and removed from the dataset so that our DNN model does not get overfitted to the outlier observations. Then the dataset is divided into training, validation, and test data where training data is used to train the prediction model, validation data is used to identify the instant the model starts getting overfitted. The test data is completely hidden from the training process. The final model is applied to the test data to evaluate the model quality. The entire framework is described in Figure 5.

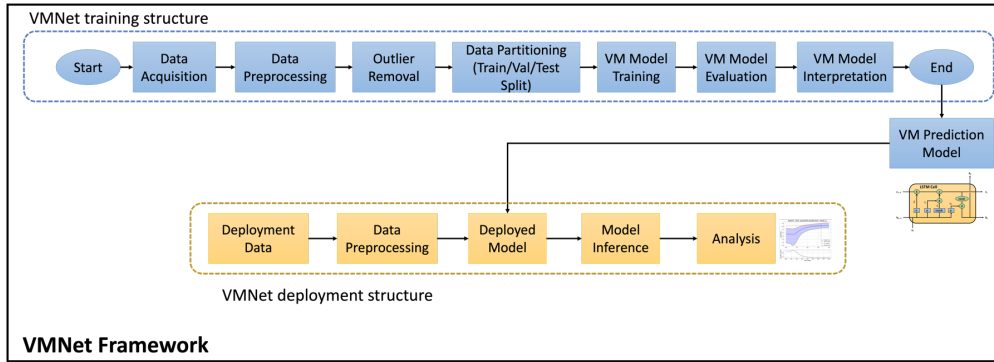


Figure 5. Proposed VMNet framework for generating prediction model and deploying it in practice.

Data Preparation:

To perform virtual metrology on a long batch process, we need to collect the sensor readings after a fixed interval (say after every τ minutes) for every run until it ends. Each such run produces a batch of final product which can be identified by a unique batch ID. The individual batches are finally measured to form the process response that we are interested in predicting. Overall, we are supposed to have the output response values and associated fixed-interval sensor readings for every batch, but the number of time steps in the runs can vary as some batches might take more time than others. The dataset obtained for the present virtual metrology process comes in two parts: sensor information and metadata information. The sensor information contains the sensor readings in the order of one hundred sensors after every fixed interval for all the batches and the metadata information contains higher-level information about the batches and the process response measured for the runs.

In order to keep the proprietary information secret, we have normalized all the sensor readings and the process output values between 0 and 1. We have also renamed all the sensors in terms of numbers (e.g. sensor_1, sensor_2, ...), instead of their actual names. Finally, the data has been partitioned using a 75%-15%-15% scheme for training, validation, and testing of our model, respectively.

Model Architecture:

We have used a Long-Short Term Memory (LSTM) based architecture for the time series simulation and two branches of linear layers for sensor reading prediction and response prediction, respectively. The entire model configuration is shown in Figure 6. The model receives the first t_b time steps of ground truth data (x_1, x_2, \dots, x_{t_b}) for the sensor readings. The LSTM module then encodes the information from the t_b initial time steps and starts simulating from $(t_b + 1)^{th}$ time step till the end ($x'_{t_b+1}, x'_{t_b+2}, \dots, x'_T$). For each time step, a linear layer (MLP_s) maps the hidden state of the LSTM to the sensor readings which is recursively fed back to the LSTM for prediction of the next timestep and at the end of the simulation, another linear layer (MLP_y) maps the hidden states to the response values. The problem of mapping is extremely sparse as the dataset does not contain the intermediate response values.

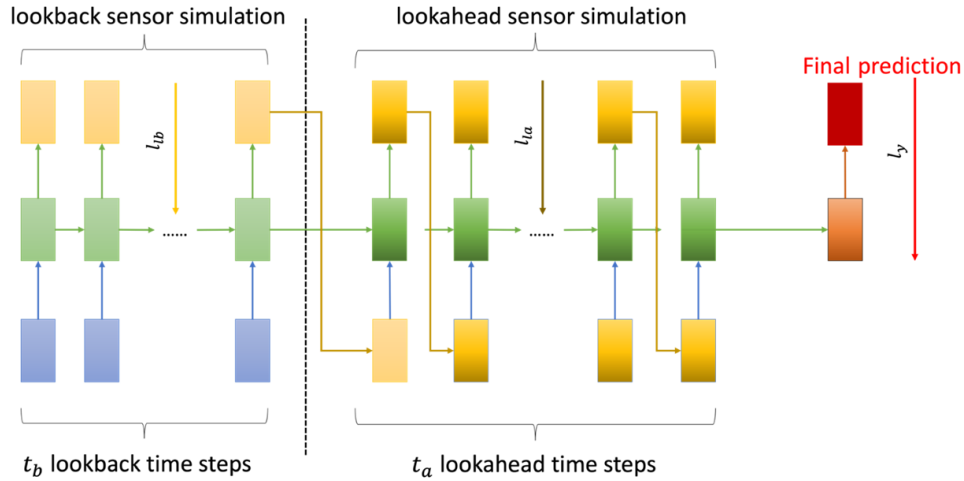


Figure 6. VMNet model architecture. There are three different components in the architecture: lookback sensor simulation, lookahead sensor simulation, and final response prediction. The loss propagation during backpropagation from the architecture is also explained in the diagram.

Model Training:

For training the proposed model, we need to use a loss function that can combine the losses from both simulations of the sensors and the final predictions. While training the model, we have created two separate phases for the LSTM component. One phase is called the lookback phase and the other phase is named as the lookahead phase. In lookback phase, the LSTM gets the ground truth sensor data for the first t_b time steps. After the lookback phase is over, the LSTM uses the final captured hidden state and cell state from the lookback phase and enters the lookahead phase. In the lookahead phase, we do not use any ground truth information. In this phase, we collect the hidden state of

the LSTM at each time step, map the hidden state to the sensor readings using an MLP (MLP_s), and the mapped sensor readings are recursively fed back to the LSTM as the input for the next time step. The difference between the lookback and lookahead phases is that lookback phase gets access to the ground truth sensor data for the initial t_b timesteps, whereas in lookahead phase, the LSTM output is used as an approximation for the ground truth data. This lookback-lookahead system closely imitates the original deployment situation where we have access to the sensor readings of t time steps. Then we have to use the information captured from the observed sensor readings to predict the rest of the future readings. So, the lookahead part imitates how it will be predicted in deployment. At the final timestep (T) of the lookahead, we also map the hidden representation of the LSTM to the process response values using a separate MLP (MLP_y).

For both lookback and lookahead, we compute the error with respect to the sensor predictions by comparing it with the original sensor readings of the dataset. At the final timestep (T), we also compute the error with respect to the predicted response. We categorize the losses that we get from the data and the prediction into three components: response loss (l_y), lookback loss (l_{lb}), lookahead loss (l_{la}). The final loss is the sum of all these losses as shown in Equation (4).

$$Loss = l_y + l_{la} + l_{lb}. \quad (4)$$

Results:

In this section, we present a brief description of the data used for training and testing the model, followed by the results obtained by our proposed method.

Data Description:

The raw data is first processed by the method mentioned in Section 5.1. Then the outliers are removed from the dataset based on runtime and some knowledge about different sensors. The final dataset consists of more than one thousand batches. Each run contains information from on the order of one hundred sensors along the runtime and three process response values that we are trying to predict. The examples of some of the sensor trends till 200 time steps are shown in Figure 7.

In some data batches, the batch process continues until about 300 timesteps. The final problem can be treated as finding a mapping to the time series data of the sensors to the final response values. The frequency distributions of all these values are shown in Figure 8.

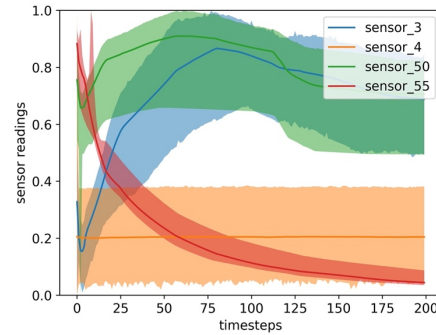


Figure 7. Example of four sensor trends from the dataset. The wide band for the sensor readings represent the amount of variation seen in the dataset for the individual sensors.

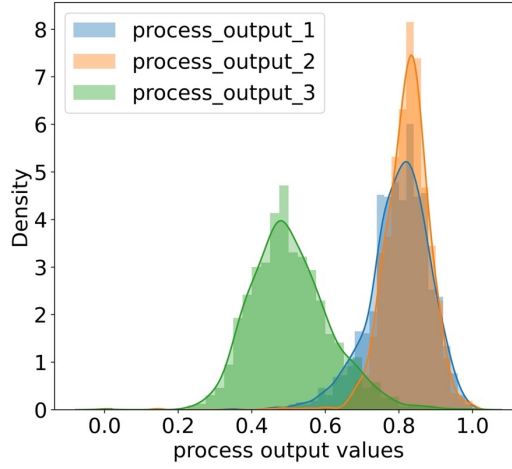


Figure 8. Frequency distribution of the response values in the available dataset, clearly stating that they vary from data to data.

Neighborhood Exploration:

Before starting with the model training, we need to have an idea about the difficulty of the problem in terms of the dataset. This kind of analysis can help us decide on a model to apply to the problem. One way to think about the difficulty of the given problem is that if the data agrees with the problem formulation, the batches which have similar sensor readings should have similar response values for prediction. To verify that we have used a neighborhood identification process to find out the nearest neighbor of every batch in the dataset.

Let \mathbf{S}_i and \mathbf{Y}_i denote the sensor readings and response values of batch i , respectively. Then the batch nearest to batch i (batch j_i) is computed using Equation (5), where N is the number of sensors in the dataset.

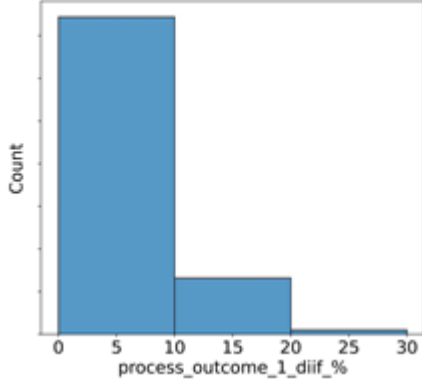
$$j_i = \underset{x}{\operatorname{argmin}} \frac{|\mathbf{S}_i^k - \mathbf{S}_x^k|}{N}. \quad (5)$$

Here, the superscript k denotes the timestep. After retrieving the nearest neighbors (index j_i) for all the batches, we check the differences in the response for the corresponding neighboring batches. The difference is represented in percentage and calculated from Equation (6), where d_i^y denotes the percentage difference in response for batch i from its nearest neighbor (batch j_i).

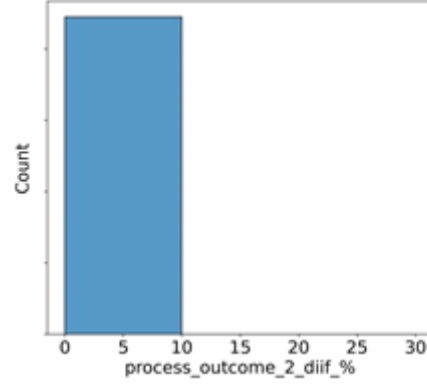
$$d_i^y = 2 \times \frac{|Y_{j_i}^y - Y_i^y|}{Y_{j_i}^y + Y_i^y} \times 100. \quad (6)$$

We have computed the neighborhood response difference for every batch using this process and the histogram of the differences corresponding to all the values is presented in Figure 9. From the Figure, we can see that the maximum amount of batches have less than 10% response difference compared to their nearest neighbors. As the difference is not very high, this problem is solvable by deep learning techniques. Deep learning relies on the data for learning the inter-dependencies of the input variables and

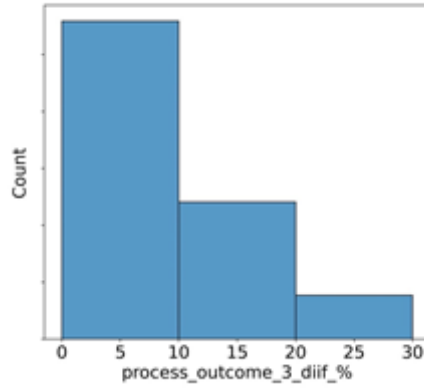
if the data is not having proper correlations among different samples, it can never work. This analysis provides us with the confidence that deep learning techniques might work in the current prediction process.



(a) Neighborhood difference histogram for response 1.



(b) Neighborhood difference histogram for response 2.



(c) Neighborhood difference histogram for response 3.

Figure 9. Neighborhood exploration results for the three responses under consideration.

Model Performance:

The main goal of the VM process is to predict the final process outputs as accurately as possible. For this reason, the performance of the model is tested using Mean Absolute Percentage Error (MAPE). A lower value of MAPE indicates a better prediction of the responses. It can be calculated using Equation (7) where A represents the actual value and F is the predicted or forecasted value:

$$MAPE = \frac{|A-F|}{A} \times 100. \quad (7)$$

Table 1. Performance of the proposed model for predicting the three process outputs considered for the batch process.

Performance Criterion	Response 1	Response 2	Response 3
MAPE (in %)	4.19	2.16	7.10

The performance of the proposed model over the three process responses is displayed in Table 1. We are able to achieve a prediction error of $< 5\%$ for the first output and $< 3\%$ for the second output, while the error for the third output is $< 8\%$. When we look at the distribution of error for different batches in Figure 10, we can see that there are certain data points outside the 100-th percentile marks of the boxplots.

We obtain less than 10% MAPE on all the different process outputs under consideration. Our model performance also agrees with the difficulty of prediction as explained in Section 6.1.1. The third value is the most difficult to predict and we are able to get an average MAPE of 7.10% on that, followed by 4.19% for the first output and 2.16% for the second one. These prediction errors are acceptable in practice considering the complexity of the prediction problem involving time-varying sensors with around 250-260 timesteps and the prediction is achieved only after about one-fifth of the total process time has passed.

Model Interpretation:

Deep learning models are hard to interpret in terms of what functions they are computing to get the mapping. For example, the proposed model configuration

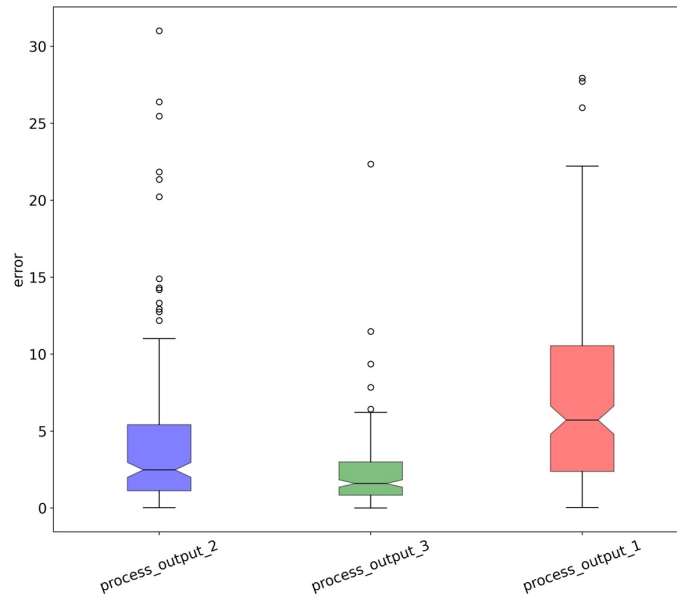


Figure 10. Error distribution of different process responses.

uses approximately one million trainable parameters which are learned during the training process. It is complicated to learn how the parameters get interconnected to provide the final predictions because of the layered nature of the model. So, in this section, we have mainly focused on suggesting some of the ways we can explore the working principles of the model through its inference on the data.

Embedded Data Analysis with Meta-parameters:

If we have some meta-parameters of interest, we can visualize the model inference on the basis of those meta-parameter values to check if the visualization agrees with the subject experts. This visualization can be used as a sanity check of the model performance. In the current process, we were interested to check if the prediction model could distinguish among different factories where the batches were processed. But this is a high-dimensional space and in order to visualize the inference, we need to map the data in two or three-dimensional space. For that reason, we have used t-SNE^[23] plots – a commonly used machine learning-based dimensionality reduction technique – to visualize the differences. To perform the analysis, we have collected the hidden representations of the different batches at the final state in the LSTM and embedded the hidden representation in two dimensions using t-SNE. Then we labeled the batches based on the factory numbers in a two-dimensional plot as shown in Figure 11. Please note that inter-cluster distance in t-SNE should not be assumed to be actual distance. But intra-cluster distances are a measure of how close they are. As we can see in Figure 11, the model can find significant clustering in terms of factory which indicates that the trained model is able to differentiate among these batches. This distinction was in agreement with the subject experts of the process. So, it increases the reliability of the model under consideration. Moreover, if it was not showing expected results, there can be multiple routes to get the expected results. The users might need to re-train the prediction model on a larger dataset/new data (if there is a data drift due to sensor modifications) or identify and fix some noisy sensors. In other scenarios, the different subject experts might be interested in other meta-parameters. The same thing can be achieved using the above-mentioned procedure.

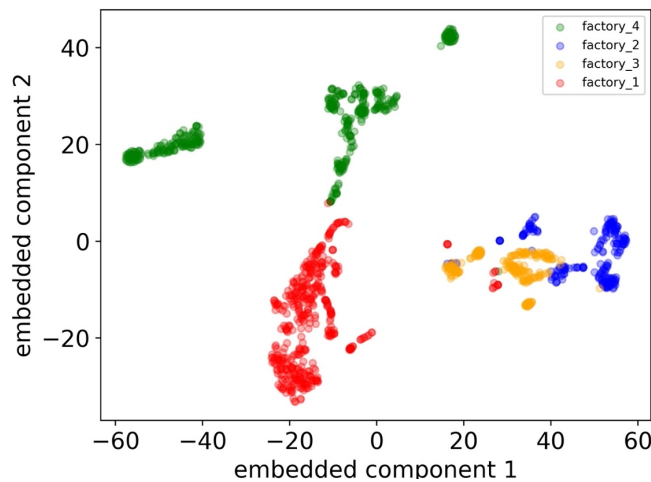


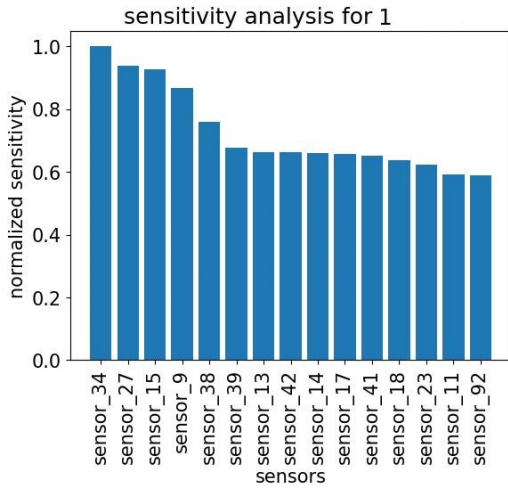
Figure 11. Embedded representations of the batches produced in different factories in a two-dimensional space, indicating that data has a bias according to the factory.

Sensitivity Analysis for Identifying Critical Sensors:

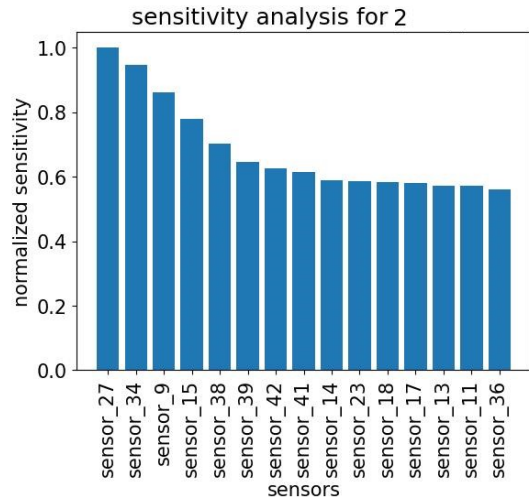
For a batch process, it is interesting to observe how sensitive the model is with respect to the different sensors used in the process. We have performed some experimentation to see how our model performance changes with respect to the variations in the different

sensor readings. To perform the sensitivity analysis, we have used the integrated gradients method ^[24] on the sensor data. Integrated gradients start by considering a baseline for the input sensor data and change the input linearly to get to the original value of the sensor data. In the process, it observes the changes happening in the process response predictions and computes the gradients for the sensor data for corresponding changes in the output data. The gradients are integrated to find an attribution score for each sensor which approximates the sensitivity of the sensor in model predictions. In the current situation, we have three different responses. We want to know which sensors are the most sensitive in predicting these three process outcomes.

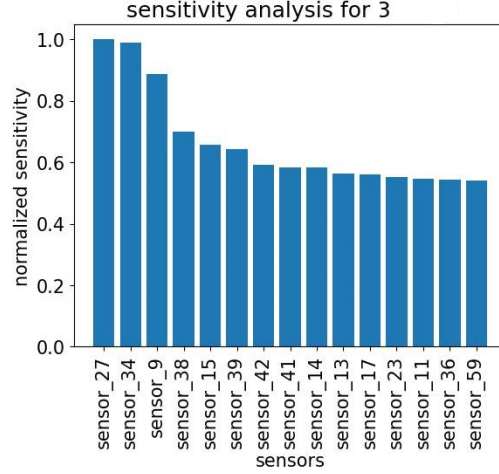
In Figure 12, we have shown the top 15 sensitive sensors affecting the prediction of the three process responses under consideration. We can see that most of the sensitive sensors (13 to be exact) are common to all three predictions. It suggests that these sensors should get more attention than the rest of them to achieve a better prediction of process responses. One immediate conclusion that we can draw from this analysis is that if we need to do quality control on the sensors so that the correct data is getting recorded, we can start with the most sensitive sensors first as the model is more sensitive to the changes in those sensors. So, if we can reduce the noise on these sensors, the model performance can be improved. In addition to making predictions, our approach is also capable of identifying key sensors affecting the prediction.



(a) Top 15 sensitive sensors for predicting Response 1.



(b) Top 15 sensitive sensors for predicting Response 2.



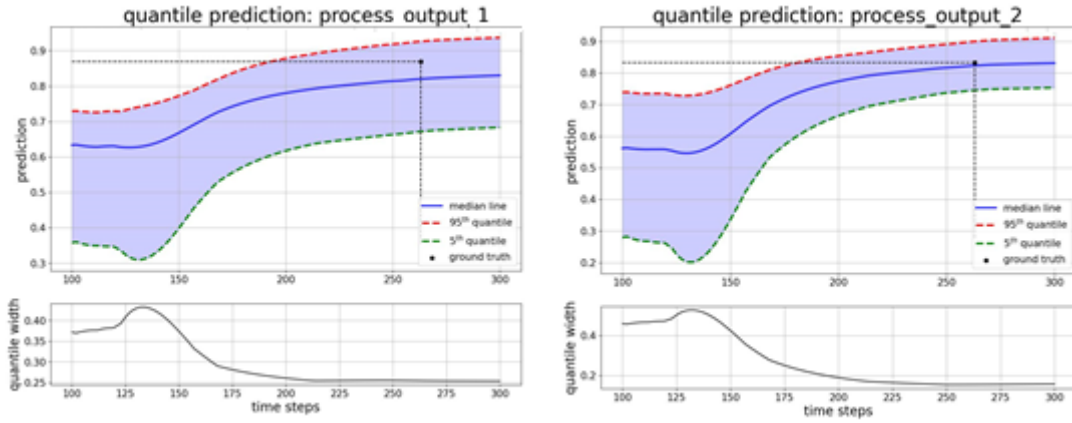
(c) Top 15 sensitive sensors for predicting Response 3.

Figure 12. Sensor sensitivity exploration for different predictions.

Uncertainty-based Model for Better Deployment:

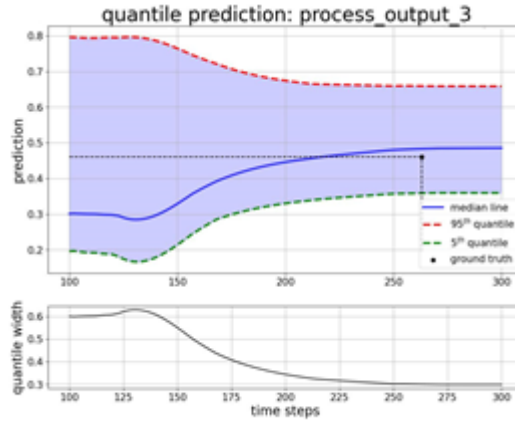
In most of the batch processes involving stochastic simulations, we need to move towards more stochastic predictions over deterministic predictions for the models. It might be helpful to provide a range of values as predictions, rather than providing a single value. The main idea is to provide a range of predictions with a certain confidence to the user. To achieve that, we need to use a modified version of the model that is capable of providing ranges, instead of particular values. For the proposed framework, we have attempted to predict the 5-th, 50-th (median), and 95-th percentile predictions using a recent quantile-based machine learning approach [25]. The model has been trained using pinball loss [22]. We named the model as quantile model. As the model provides a range of uncertainty-based estimations with a certain confidence bound, it is extremely useful for deployment.

To show the application of such a model, we have randomly selected a batch and analyzed the model predictions in Figure 13. The data for the first 100 timesteps of the process are recorded and used to predict three process responses thereafter until the end of the operation. It becomes interesting to note how the uncertainty bound (difference between 95% and 5% predictions) changes with timesteps. For all the responses, the original ground truth value for the batch is shown as a black circle in the plots and the blue line represents the median prediction over the timeline. The subplot at the bottom represents the uncertainty width for predictions at different time steps. For this batch, the quantile model is able to make accurate predictions and the uncertainty width is significantly low when it reaches the time step of the original ground truth. In deployment, the process can be stopped when the VM model provides a significantly small prediction uncertainty bound. While the difference in predictions between the 95-th and 5-th quantile is relatively wide, the median prediction is quite close to the ground truth for all three responses.



(a) Quantile model prediction for Response 1 for a random batch.

(b) Quantile model prediction for Response 2 for a random batch.



(c) Quantile model prediction for Response 3 for a random batch.

Figure 13. Quantile model performance for a random batch in the dataset over three process responses. The predictions are made after 100 timesteps are executed for the remaining duration of the batch process.

Conclusion:

In this paper, we have proposed a generalized and scalable machine learning-based virtual metrology system named VMNet for a general time-consuming batch process. The system consists of data processing, outlier removal, model training, prediction, and interpretation steps. The prediction model is capable of handling scaled machine parameter dimensions and generating predictions of process outputs. Furthermore, we have proposed systematic processes to analyze data difficulty and model interpretations. A modified version of the prediction system (predicting quantile ranges, instead of a single value) is also proposed which is more suitable for deployment. The final prediction error for the three response values under consideration is found to be less than 10% which is acceptable after utilizing only one-fifth data of the entire batch timeline.

Every long batch process is different in nature and the proposed framework might need some modifications to suit the problem under consideration. But this

framework was developed keeping scalability, generalization, interpretability, and deployment in mind. So, the framework is easily extendable to cover a wide range of time-varying problems. In the future, it might be interesting to add a robustness analysis step before deploying the model. In deployment, it is important to properly analyze the robustness of the model to take care of data drift in the collected data over the years due to process changes and sensor updates. Moreover, an outlier detection module might help to improve the performance of the model significantly in the deployment where the incoming batch data should go through a binary classifier to be identified as an outlier or a non-outlier batch. The model should only process the non-outlier batches in the deployment.

References

- [1] Chang, Y.-J.; Kang, Y.; Hsu, C.-L.; Chang, C.-T.; Chan, T. Y. Virtual Metrology Technique for Semiconductor Manufacturing. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*; IEEE, 2006; pp 5289–5293.
- [2] Kang, P.; Lee, H.; Cho, S.; Kim, D.; Park, J.; Park, C.-K.; Doh, S. A Virtual Metrology System for Semiconductor Manufacturing. *Expert Syst. Appl.* **2009**, *36* (10), 12554–12561.
- [3] Scopus. <https://www.scopus.com> (accessed 2022-11-12).
- [4] Hung, M.-H.; Lin, T.-H.; Cheng, F.-T.; Lin, R.-C. A Novel Virtual Metrology Scheme for Predicting CVD Thickness in Semiconductor Manufacturing. *IEEE/ASME Trans. mechatronics* **2007**, *12* (3), 308–316.
- [5] Cheng, F.-T.; Huang, H.-C.; Kao, C.-A. Developing an Automatic Virtual Metrology System. *IEEE Trans. Autom. Sci. Eng.* **2011**, *9* (1), 181–188.
- [6] Hirai, T.; Kano, M. Adaptive Virtual Metrology Design for Semiconductor Dry Etching Process through Locally Weighted Partial Least Squares. *IEEE Trans. Semicond. Manuf.* **2015**, *28* (2), 137–144.
- [7] Kang, P.; Kim, D.; Lee, H.; Doh, S.; Cho, S. Virtual Metrology for Run-to-Run Control in Semiconductor Manufacturing. *Expert Syst. Appl.* **2011**, *38* (3), 2508–2522.
- [8] Khan, A. A.; Moyne, J. R.; Tilbury, D. M. Virtual Metrology and Feedback Control for Semiconductor Manufacturing Processes Using Recursive Partial Least Squares. *J. Process Control* **2008**, *18* (10), 961–974.
- [9] Cheng, F.-T.; Chen, Y.-T.; Su, Y.-C.; Zeng, D.-L. Evaluating Reliance Level of a Virtual Metrology System. *IEEE Trans. Semicond. Manuf.* **2008**, *21* (1), 92–103.
- [10] Lee, S.; Kang, P.; Cho, S. Probabilistic Local Reconstruction for K-NN Regression and Its Application to Virtual Metrology in Semiconductor Manufacturing. *Neurocomputing* **2014**, *131*, 427–439.
- [11] Su, Y.-C.; Lin, T.-H.; Cheng, F.-T.; Wu, W.-M. Accuracy and Real-Time Considerations for Implementing Various Virtual Metrology Algorithms. *IEEE Trans. Semicond. Manuf.* **2008**, *21* (3), 426–434.
- [12] Susto, G. A.; Pampuri, S.; Schirru, A.; Beghi, A.; De Nicolao, G. Multi-Step Virtual Metrology for Semiconductor Manufacturing: A Multilevel and Regularization Methods-Based Approach. *Comput. Oper. Res.* **2015**, *53*, 328–337.
- [13] Park, C.; Kim, Y.; Park, Y.; Kim, S. B. Multitask Learning for Virtual Metrology in Semiconductor Manufacturing Systems. *Comput. Ind. Eng.* **2018**, *123*, 209–219.
- [14] Eger, S. Multiple Many-to-Many Sequence Alignment for Combining String-

- Valued Variables: A G2P Experiment. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*; 2015; pp 909–919.
- [15] Iwata, T.; Lloyd, J. R.; Ghahramani, Z. Unsupervised Many-to-Many Object Matching for Relational Data. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *38* (3), 607–617.
 - [16] Hopfield, J. J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. Natl. Acad. Sci.* **1982**, *79* (8), 2554–2558.
 - [17] Schmidhuber, J. Habilitation Thesis: System Modeling and Optimization. *Page 150 ff Demonstr. Credit Assign. across Equiv. 1,200 layers an unfolded RNN* **1993**.
 - [18] Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9* (8), 1735–1780.
 - [19] Chang, J.; Lan, Z.; Cheng, C.; Wei, Y. Data Uncertainty Learning in Face Recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; 2020; pp 5710–5719.
 - [20] Busch, M.; Schnoes, F.; Elsharkawy, A.; Zaeh, M. F. Methodology for Model-Based Uncertainty Quantification of the Vibrational Properties of Machining Robots. *Robot. Comput. Integr. Manuf.* **2022**, *73*, 102243.
 - [21] Fox, M.; Rubin, H. Admissibility of Quantile Estimates of a Single Location Parameter. *Ann. Math. Stat.* **1964**, 1019–1030.
 - [22] Koenker, R.; Bassett Jr, G. Regression Quantiles. *Econom. J. Econom. Soc.* **1978**, 33–50.
 - [23] Van der Maaten, L.; Hinton, G. Visualizing Data Using T-SNE. *J. Mach. Learn. Res.* **2008**, *9* (11).
 - [24] Sundararajan, M.; Taly, A.; Yan, Q. Axiomatic Attribution for Deep Networks. In *International conference on machine learning*; PMLR, 2017; pp 3319–3328.
 - [25] Wang, Y.; Gan, D.; Sun, M.; Zhang, N.; Lu, Z.; Kang, C. Probabilistic Individual Load Forecasting Using Pinball Loss Guided LSTM. *Appl. Energy* **2019**, *235*, 10–20.