Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 171–181
DOI: 10.15439/2023F5263
ISSN 2300-5963 ACSIS, Vol. 35

Time-series Anomaly Detection and Classification with Long Short-Term Memory Network on Industrial Manufacturing Systems

Tijana Markovic §
School of Innovation, Design and Engineering
Malardalen University
72123, Vasteras, Sweden
tijana.markovic@mdu.se

Alireza Dehlaghi-Ghadim §
Research Institute of Sweden (RISE)
School of Innovation, Design and Engineering, Malardalen University
72164, Vasteras, Sweden
alireza.dehlaghi.ghadim@ri.se,mdu.se

Miguel Leon §
School of Innovation, Design and Engineering
Malardalen University
72123, Vasteras, Sweden
miguel.leonortiz@mdu.se

Ali Balador
School of Innovation, Design and Engineering
Malardalen University
72123, Vasteras, Sweden
ali.balador@mdu.se

Sasikumar Punnekkat
School of Innovation, Design and Engineering
Malardalen University
72123, Vasteras, Sweden
sasikumar.punnekkat@mdu.se

***Abstract*—Modern manufacturing systems collect a huge amount of data which gives an opportunity to apply various Machine Learning (ML) techniques. The focus of this paper is on the detection of anomalous behavior in industrial manufacturing systems by considering the temporal nature of the manufacturing process. Long Short-Term Memory (LSTM) networks are applied on a publicly available dataset called Modular Ice-cream factory Dataset on Anomalies in Sensors (MIDAS), which is created using a simulation of a modular manufacturing system for ice cream production. Two different problems are addressed: anomaly detection and anomaly classification. LSTM performance is analysed in terms of accuracy, execution time, and memory consumption and compared with non-time-series ML algorithms including Logistic Regression, Decision Tree, Random Forest, and Multi-Layer Perceptron. The experiments demonstrate the importance of considering the temporal nature of the manufacturing process in detecting anomalous behavior and the superiority in accuracy of LSTM over non-time-series ML algorithms. Additionally, runtime adaptation of the predictions produced by LSTM is proposed to enhance its applicability in a real system.**

***Index Terms*—anomaly detection, anomaly classification, machine learning, deep learning, LSTM, sensor data, manufacturing systems**

I. Introduction

MODERN manufacturing systems have hundreds of sensors that record a huge amount of data, which provides an opportunity to use data science to improve the performance of manufacturing processes [1]. Valuable information and knowledge can be extracted from these data using different techniques, such as machine learning algorithms, statistical analysis methods, data visualization techniques, etc. [2]. One very important aspect that can be addressed is the detection of anomalous behavior in the manufacturing system. Abnormal process behavior is a major problem that can cause a decrease in the quality of a product or a complete process failure that results in the direct loss of a huge amount of money and raw material. The process may fail due to malfunctioning equipment, poor maintenance, external hacker attack, etc. All components and sensors in the system must be continuously monitored, and prompt actions should be provided if any deviations from normal behavior are identified.

This paper aims to detect anomalous behavior in industrial manufacturing systems by considering the temporal nature of the manufacturing process. This temporal nature can be presented by time-series data, which can be easily obtained from any manufacturing system. The time-series is a collection of observations made chronologically, characterized by large data size and high dimensionality [3]. In this paper, the time-series Machine Learning (ML) algorithm, more specifically, Long Short-Term Memory (LSTM) network, is applied on a synthetically generated dataset called Modular Ice-cream factory Dataset on Anomalies in Sensors (MIDAS) [4], which was created using a simulation of a modular manufacturing system for ice cream production. Two different problems are addressed: Anomaly Detection (AD) and Anomaly Classification (AC).

The main contributions of the paper can be summarized as follows:

- an analysis of the LSTM performance (in terms of accuracy, execution time, and memory consumption) in a new data set on manufacturing systems for AD and AC,

§Equal contribution

- a comparison in performance between LSTM performance and non-time-series ML algorithms,
- runtime adaptation of the predictions produced by LSTM to enhance its applicability in a real system.

The rest of the paper has been divided into the following sections. Section II provides a description of related research. A background on deep learning and more specifically on LSTM is provided in Section III. The experimental setup, and the results and discussion for the conducted experiments are given in Sections IV and V, respectively. The conclusion and the future work wrap up the paper in Section VI.

II. Related Work

Detecting anomalies in time-series data using ML techniques has recently piqued the interest of many researchers and has been the subject of several studies. Time-series data can be found in various application domains such as smart manufacturing, health monitoring, cyber security, and smart energy management [5]. The data source and its application can have a significant impact on the efficacy of different AD approaches. Moreover, to select an appropriate approach, data characteristics should be also considered, such as label availability and data volume. Several surveys exist that study techniques for AD with respect to different application domains [6], [7], [8]. The focus of the work presented in this paper is to detect anomalies in industrial systems through the analysis of data from multiple sensors.

Identifying patterns or deviations in industrial sensors can be done using various techniques such as statistical analysis, simple ML algorithms, and deep learning. Simple ML methods often utilize classical techniques to model the distribution of time-series data. The authors in [9] used One-Class Support Vector Machine (OC-SVM) and extended Kalman filter for real-time sensor AD in connected automated vehicle sensors. Their proposed method combines model-based signal filtering and ML technique to detect anomalous sensor readings and/or malicious cyber attacks. In [10], the authors used multivariate linear regression and Gaussian mixture models to detect anomalies in the reading sensor values from engine-based machines. They create a model from the correct behavior of the system based on sensor values such as fuel usage, engine load, and oil pressure to gauge and identify specific failures in the machine by comparing the received data with the normal model.

In recent years, deep learning models have shown promising results in time-series modeling [11]. The deep learning ability to automatically learn complex patterns from huge amounts of data makes it suitable for time-series AD. In the application of deep learning on time-series data, there are three clear approaches that can be seen in the literature.

The first approach is to use LSTM. In [12], an automated approach to detecting anomalies using a supervised LSTM and statistical analysis is introduced. This study uses an LSTM neural network to predict non-robust statistical properties and robust ones. This model identifies anomalies in time-series data by combining statistical analysis with a supervised LSTM neural network. In [13], a supervised LSTM-based model is introduced to detect abnormal data generated by mechanical equipment. The model extracts spatial features from the visual representation of the signal's frequency spectrum over time using a convolutional model and detects anomalies using a residual LSTM. In [14], authors introduced an LSTM-based real-time AD algorithm for time-series that can tolerate minor pattern changes. This algorithm automatically calculates the detection threshold based on changes in the data pattern. They employed two LSTM models, each with a distinct threshold. The first threshold is derived by taking into account all data points, whereas the second threshold is determined by taking into account those data points that are considered normal. Two LSTM models work in parallel, where one LSTM model finds single-point anomalies, while the other detects anomalous based on the long-term threshold.

The second approach combines an autoencoder with LSTM for unsupervised/semi-supervised problem-solving. In [15], an LSTM-based scheme is proposed for multi-sensor AD. It uses normal data to train an encoder-decoder model, which reconstructs normal data with minimal error. Anomalies are detected by measuring reconstruction error and likelihood. In [16], a model using autoencoders and residual error detects anomalies in sound sensor data for complex machines, aiding early maintenance planning. [14] introduces an LSTM-based real-time AD algorithm for time-series, adjusting the detection threshold based on pattern changes. Two LSTM models work together for single-point and long-term AD. In [17], a stacked autoencoder connects gated neural networks and LSTMs for short-term and long-term AD in discrete manufacturing. Authors in [18] characterized multi-sensor time series with a deep convolutional autoencoder model. They used a non-linear bidirectional LSTM and linear auto-regressive model for prediction. Multi-Scale Convolutional Recurrent Encoder-Decoder (MSCRED) framework proposed in [19], where they used convolutional recurrent encoder-decoder for detection and diagnosis in multivariate time series data. In this framework, the spatial information is encoded into signature matrices using a convolutional encoder, while the temporal information is modeled using an attention-based ConvLSTM. In [20], the authors proposed the Multivariate Time-series Anomaly Detection via Graph Attention Network (MTAD-GAT) framework, which treats each univariate time series as an individual feature and includes a feature-oriented graph attention layer and a time-oriented graph attention layer to detect the complex dependencies of multivariate time series. Furthermore, to find anomalies, it computes the inference score of forecasting-based model prediction reconstruction-based model predictions. Variational Autoencoder Generation Adversarial Networks (LSTM-based VAE-GAN) method proposed in [21] which uses Generative Adversarial Networks (GAN) for time series AD to monitor the equipment sates. This method trains the encoder, the generator and the discriminator with LSTM models and detects anomalies based on reconstruction error and discrimination results. Corizzo et al. [22] presented an AD model that leverages spatial awareness through a stacked

autoencoder architecture. The proposed method utilizes spatial autocorrelation patterns generated by the autoencoder during the distinctive encoding phase to detect anomalies based on distance measures. The authors evaluated the effectiveness of their algorithm using geo-distributed data collected from renewable energy plants.

The final approach is to use a Convolutional Neural Network (CNN). In [23] a fault detection and diagnosis model using a CNN in semiconductor manufacturing is proposed. The authors applied windowing techniques to deal with variable-length multiple time-series data and trained CNNs to detect anomalies in the quality of electric wafers. This model is equipped with a mechanism to identify the contribution of each sensor on anomalies to provide traceable information for fault diagnosis. The work presented in [24] proposes a new deep learning architecture for supervised AD on multi-sensor data using a combination of CNN and Recurrent Neural Networks (RNN). This architecture is proposed to detect anomalies in elevators with different sensors in terms of type and count. They used individual CNN networks for each sensor as a first layer to extract features. This layer eliminates the need for preprocessing and provides architecture design flexibility for the next layers. The next layer of this architecture aggregates these features using LSTM or RNN networks using the windowing technique, where each window can model part of the time-series. The work presented in [25] proposed an algorithm based on CNN and Spectral Residual (SR) to improve the performance of AD on public datasets and Microsoft production data. They converted time series data with the SR model and then used visual saliency detection for AD in time series.

The main difference between this paper and all the above mentioned papers is that this paper focuses on the application of basic LSTM approach to detect and classify anomalies in analog sensors data from a manufacturing environment and compares that approach with different non-time-series ML algorithms.

III. Long Short-term Memory Recurrent Neural Network

A. Artificial Neural Network

Artificial Neural Network (ANN) [26], [11] is an ML algorithm inspired by the human brain. Artificial neurons are organized into layers (input layer, one or more hidden layers, and output layer), where the output of one layer is used as input to the next layer. An example of ANN architecture can be seen in Fig. 1. The layers are interconnected with connections that have parameters called weights. The output of each layer is calculated as follows

$$\overline{O}_{layer_i} = activation((W_{layer_i})^T \times \overline{O}_{layer_{i-1}} + \overline{B}_{layer_i}) \quad (1)$$

where $O_{layer_{i-1}}$ is the output of the previous layer (if i is the first layer, then $O_{layer_{i-1}}$ is equal to the input values X), $(W_{layer_i})^T$ is the transpose of the weights that connect layer i and layer $(i-1)$ and B_{layer_i} are random weights that are not multiplied by any input. Finally, an activation

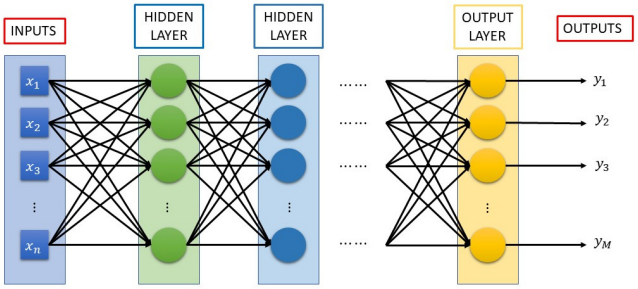


Fig. 1: Artificial Neural Network

function transforms the output into a different range. There are different activation functions as sigmoid, tanh, softmax, ReLU, LeakyReLU, etc. [26]. Three activation functions that are used in the paper are: sigmoid (Eq. (2)), tanh (Eq. (3)) and softmax (Eq. (4)). The sigmoid activation function transforms the data into $(0,1)$ interval, while tanh transforms it to $(-1,1)$. The softmax activation function is only used for the output layer and it does not normalize the output where the values are in the $(0,1)$ range and the summation of all values is equal to 1.

$$sigmoid(x) = \sigma(x) = \frac{1}{1+e^{-x}} \quad (2)$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3)$$

$$softmax(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \quad (4)$$

The goal of the ANN training process is to adjust its weights (black connections in Fig. 1) to fit the objectives. This is done using the backpropagation algorithm [27], [11]. Once the weights are trained, the ANN is ready to be used.

B. Recurrent Neural Network

Recurrent Neural Networks (RNN) is a type of Artificial Neural Network that takes into consideration time-series data. In order to compute the output at time t, the calculations made in $t-1$ are used. Fig. 2 shows an example of RNN, and how it uses the different time steps. To be more specific, the equations that compute the output for time step t ($\overline{Y}(t)$) are as follows

$$\overline{Y}(t) = activation((W_y)^T \times \overline{S}(t)) \quad (5)$$

$$\overline{S}(t) = activation((W_x)^T \times \overline{X}(t) + (W_s)^T \times \overline{S}(t-1)) \quad (6)$$

where W_y, W_s, W_x are weight matrices for the connection between the hidden layer and the output layer, the hidden layer connected to itself, and the connection between the input layer and the hidden layer, respectively. Additionally, $\overline{X}$ and $\overline{S}$ are vectors that represent the input and the output of the hidden layer, respectively.

Unfortunately, RNN has a problem, called grading vanishing problem, when training with backpropagation algorithm [26]. For this reason, a more advanced type of network is needed.

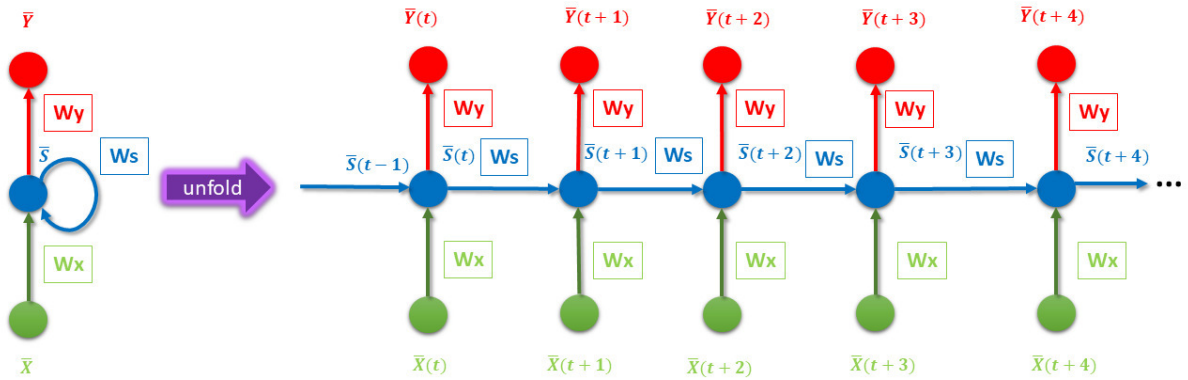


Fig. 2: Recurrent Neural Network in its fold and unfolded version.

C. Long-Short Term Memory Neural Network

Long-short Term Memory (LSTM) neurons are presented to solve the grading vanishing problem. Each of this units is composed of:

- Two memories:
  - **Long-term memory** (*LTM*) which passes the information that has been used for a long period of time to the current time step. Additionally, thanks to this memory, the gradient vanishing problem is solved.
  - **Short-term memory** (*STM*) which saves the information that has been used in the previous time step.
- Four different areas (gates):
  - **Forget gate** which finds information that can be discarded. It is calculated as follows:

$$\overline{F}(t) = \sigma((W_f)^T \times [\overline{X}(t), \overline{STM}(t-1)] + \overline{B}_f) \quad (7)$$

where W_f represents the weights of the forget gate, $\overline{X}$ the input information, $\overline{STM}$ the short-term memory, t the time step and $[\overline{a}, \overline{b}]$ the concatenation of the vector a with the vector b.

  - **Input gate** which finds information that is relevant ($\overline{R}(t)$) and needs to be modified into the LTM, and weights the values based on their importance ($\overline{I}(t)$) into the [-1,1] range. The calculations are as follows:

$$\overline{R}(t) = \sigma((W_r)^T \times [\overline{X}(t), \overline{STM}(t-1)] + \overline{B}_r) \quad (8)$$

$$\overline{I}(t) = tanh((W_i)^T \times [\overline{X}(t), \overline{STM}(t-1)] + \overline{B}_i) \quad (9)$$

where W_r and W_i are the weights used to calculate the relevance and the importance, respectively. In the same way, $\overline{B_r}$ and B_i represent the biases used to calculate the relevance and the importance, respectively.

  - **Update gate** which modifies LTM by adding the information found in the input gate. Additionally, LTM needs to be modified by $\overline{F}$ calculated in Eq. (7). The LTM vector ($\overline{LTM}$) for the current time step t is calculated as follows:

$$\overline{LTM}(t) = \overline{LTM}(t-1) \times \overline{F}(t) + \overline{R}(t) \times \overline{I}(t) \quad (10)$$

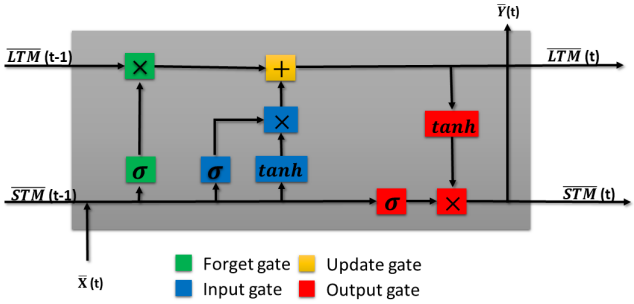


Fig. 3: Long-Short term memory unit. STM and LTM stand for Short-Term Memory and Long-Term Memory, respectively, where t indicates the time step.

  - **Output gate** which calculates the final output that depends on the input information, the output used in the previous time step and the LTM that is previously modified. It is calculated as follows:

$$\overline{Y}(t) = \overline{STM}(t) = \sigma((W_y)^T \times [\overline{X}(t), \overline{STM}(t-1)] + \overline{B}_y) \times tanh(\overline{LSTM}(t)) \quad (11)$$

where W_y is the weight matrix used in the use gate, while $\overline{B}_y$ is the bias used on the same gate.

The complex units explained above are used to replace the simple units in the hidden layer of RNN (blue circles in Fig. 2). An overview of the entire LSTM unit is given in Fig. 3.

IV. Experiments

A. Dataset Description

The experiments presented in this paper are conducted on a publicly available dataset called MIDAS[1], which is created using a modular manufacturing simulation environment, where an ice cream making process is simulated [4]. The simulated system is composed of 6 interconnected modules, and each module has various analog and digital sensors [28], [29]. To

[1]https://github.com/vujicictijana/MIDAS/

create MIDAS dataset, three different types of anomalies in analog sensors were injected during the simulation process, and each of them represents modifying the value in a different way: freeze value, step change and ramp change. From the moment of anomaly injection, the actual sensor value was replaced by a modified value until the end of the current run, which means that the simulation behavior was changed from that point since the controllers have access to the wrong values. In this way, different scenarios were simulated, from malfunctioning sensors to external intrusions. The anomalies were injected into the 8 different analog sensors, one sensor at a time.

One run of the system represents a full ice cream making process according to the given recipe, from mixing all the ingredients to producing the ice cream cones as the final product. All runs in the dataset are generated using the same recipe and they are completely independent of each other. The dataset contains a separate csv file for each of 1000 runs, where 258 runs represent normal behavior and 742 runs contains anomalies. There are three different types of anomalies: Freeze value (24.9% of total runs), Step change (24.% of total runs) and Ramp change (24.6% of total runs). The dataset is divided into 500 run for training, 100 for validation and 400 runs for testing data.

Each run has around 36000 instances and one instance represents system state at a certain time point, which results in 36,124,859 instances in the entire dataset. Each instance has 60 columns, including ordinal number of instance within one run, all sensor readings for all modules, and information about anomaly injection. If the instance contains an anomaly the anomaly type, sensor where the anomaly was injected, and actual sensor value are provided. The dataset is well balanced since the percentage of anomalous instances is 50.33%, and approximately one third of anomalous instances belongs to each type of anomaly.

B. Dataset preprocessing

During the dataset preprocessing phase, the following steps were conducted:

- Output variable was defined - The “Anomaly” column is used as the output variable. This column contains 4 different values (- for no anomaly, Step, Ramp, and Freeze) that were encoded to numbers from 0 to 3, in the given order.
- Input variables were defined - All sensor readings for all modules were used as input variables, except the ones that have the same value in the entire dataset, which resulted in 38 input variables.
- Input variables were normalized - All input variables are binary values or real numbers. Binary values remained unchanged and real numbers were normalized in the range [0, 1] using the Min-Max normalization technique.
- Numeric values were transformed to float32 - By default, the columns in the dataset have float64 as a data type. To prevent memory overflow, the data type of the normalized numbers was changed from float64 to float32, with a negligible effect on the results of the LSTM.
- Sliding window approach was applied - In order to generate time-series data for LSTM a sliding window approach was applied. The experiments were conducted with different window sizes and the windows were generated only within a run. Additionally, only the output in the last time step was consider as an output for AD/AC.

C. Experimental settings

Two different problems were addressed: Anomaly Detection (AD) and Anomaly Classification (AC) of abnormal time points on multivariate temporal data using supervised ML algorithms.

All experiments were performed in a MacBook Pro 2019 2.6GHz, Intel Core i7, 16 GB RAM. Python programming language and Tensorflow[2] [30] ML library were used to implement the experiments. The experiments were performed with 500-100-400 runs for training, validation and testing, respectively.

All experiments were conducted with all input variables that were selected during the preprocessing phase (full data), but also with the reduced number of input variables (reduced data). For the experiments with the reduced number of input variables only variables that contain sensor readings for sensors in which the anomalies were injected are considered (8 variables). During all experiments the time and memory consumption was recorded with a goal to analyse how demanding the resulting LSTM is. Each experiment was repeated 5 times and the average results are presented.

Four different experiments have been carried out:

1) *Study of hyper-parameters in LSTM for AD and AC* - The validation set was used to find the best combination of hyper-parameters in LSTM: number of hidden neurons and number of epochs. Values 1, 2, 5, 10 and 20 were tested for both parameters, for both AD and AC. Additionally, two different activation functions were tested for AD (Sigmoid and Softmax), while Softmax activation function was used for AC.
2) *Study of window sizes in LSTM for AD and AC* - The validation set was used to find the best window size with the best combination of hyper-parameters from the first experiment. The window sizes that were tested include 1, 2, 5, 10, and 20 for both AD and AC. The maximal number was selected to cover all time steps within half of a second.
3) *LSTM performance for AD and AC* - LSTM was trained with the best combination of the hyper-parameters from the first experiment and the best window size from the second experiment and tested with the testing data.
4) *Runtime adaptation of LSTM prediction* - Since the resulting LSTM will be applied in a simulator of a modular manufacturing system, it is needed to prevent false negative and false positive alarms as much as

[2]https://www.tensorflow.org/

possible. After it is certain that an anomaly has occurred, the alarm should be on until the run ends or until someone checks the systems. This means that all the remaining time points until the end of that run should be considered as anomalous. To achieve this, runtime adaption of current prediction (y_i) was implemented by checking the previous predictions and changing it according to Eq. 12. Different amount of past time points that are considered ($t \in \{10, 20, 30 \dots 3000\}$) and different rates of points that have to be predicted as anomalies ($r \in \{1, 2, \dots 9\}$) were tested on validation set. Before the moment when certainty is achieved, predictions will be considered false positive if less than 10% of predictions in the previous period are predicted as anomalies, so the final prediction will be normal behaviour. All adaptions are made considering only data from the specific run.

$$y_i = \begin{cases} 1 & \text{if } \exists_{k \leq i} \ \sum_{j=k-t-1}^{k} y_j = \frac{t}{r} \\ 0 & \text{otherwise, if } \sum_{j=i-t-1}^{i} y_j < \frac{t}{10} \\ y_i & \text{otherwise} \end{cases} \tag{12}$$

V. Results and Discussion

A. Study of hyper-parameters in LSTM for AD and AC

The performance of LSTM with different combinations of hyper-parameters on the validation set is presented in Fig. 4 for AD and in Fig. 5 for AC. It can be noticed that the accuracy of LSTM is higher as the number of epochs increases, for both AD and AC, which is expected because the longer training process enables LSTM to fit better to the data. The same happens if the number of hidden neurons is increased, which means that the LSTM can better generalize the model. However, increasing from 10 to 20, for both epochs and hidden neurons, does not bring a big improvement.

With respect to the activation function in the output layer for AD, it can seen how the use of softmax gives improvements in all cases except when there is one neuron in the hidden layer. This is normal since having two neurons in the output layer helps generalize better than having only a single unit.

Finally, if considering using all the input variables or only the variables where anomalies were injected, it can be noticed that using all available information gives benefits to the proposed model, which means that other variables in the process are also affected by the anomalies.

Taking into consideration all of the above statements it was decided to use softmax as an activation function for the output layer for AD, and 10 as the number of hidden neurons and the number of epochs for all problems. The selected combination did not have the absolute highest accuracy, however, the results are very similar with the best combination. This option is selected because it makes the model more efficient in terms of time and memory. Although the results using the full data are better, the experiments in the following sections are conducted using both options.

TABLE I: Accuracy of LSTM on validation set for different window sizes for AD and AC with all and reduced input variables

Problem / Window	AD		AC	
	full	**reduced**	**full**	**reduced**
1	84.01	82.72	68.81	68.29
2	86.75	85.21	69.58	69.51
5	87.45	86.73	69.74	69.89
10	88.06	86.07	71.27	69.01
20	**88.51**	**86.92**	**71.52**	**70.55**

B. Study of window sizes in LSTM for AD and AC

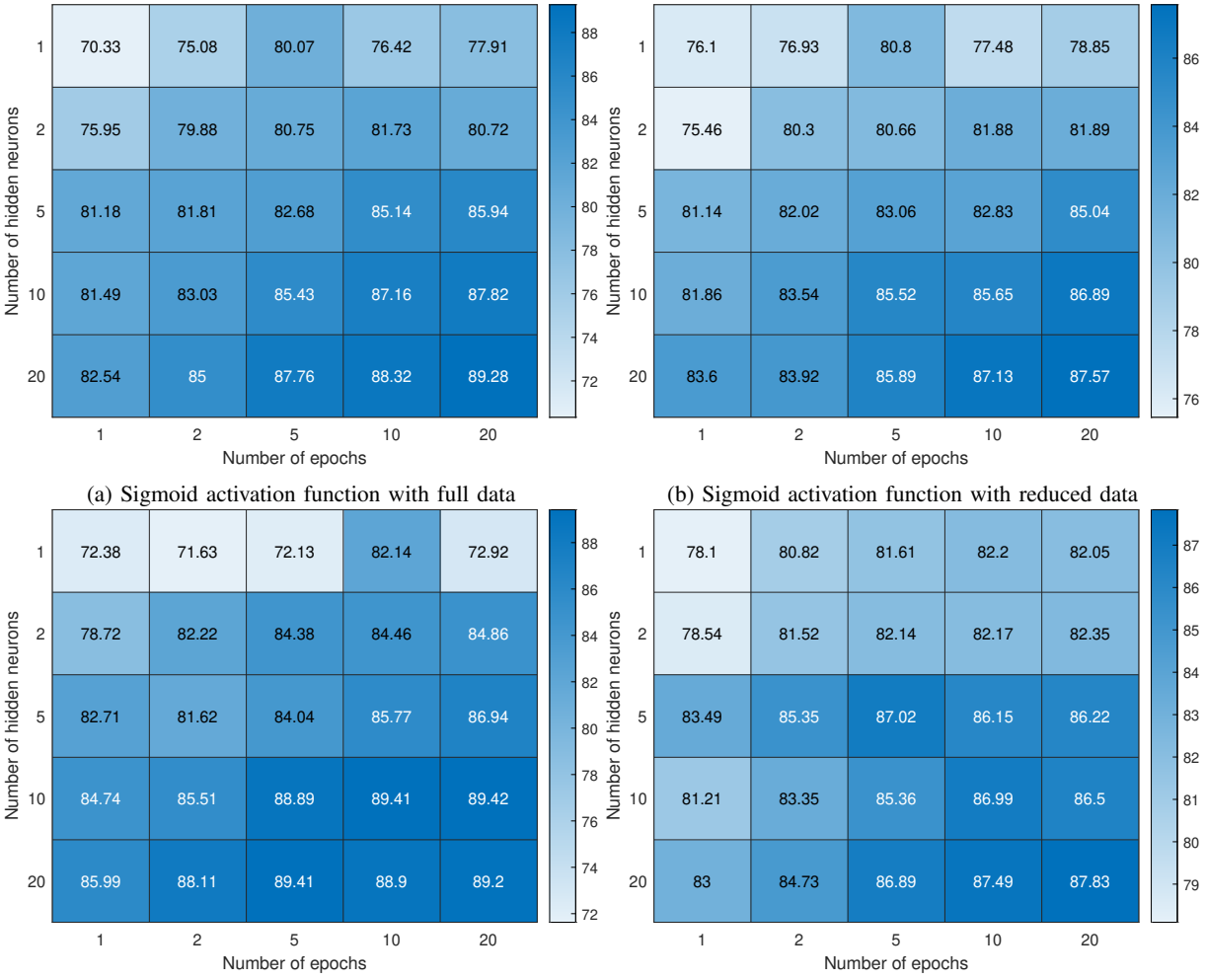
The best combination from the previous section is used to train LSTM with different window sizes and the results are presented in Table I. As expected, as the window size increases the accuracy of LSTM is higher, since the model considers more time steps back in time. It can be observed that the improvement is around 1 percentage point in almost all cases for full data in AD for window sizes 1, 2 and 5. The differences from window size 5 to 10 and 10 to 20 are around 0.5 percentage points. On the other hand, on full data for AC, the improvement is only big when comparing window size 1 to window size 2 (around 3 percentage points). Then, for the rest of the window sizes the accuracy remains stable. If the reduced data are considered, a similar behavior can be noticed for both AD and AC.

If the results between window size 1 (which considers only a single time step) and window size 20 (which considers half a second of the manufacturing process) are considered, the improvement can be noticed in all cases varying from 2.2 to 4.5 percentage points. This clearly shows the benefit of considering time-series when making a predictions.

C. LSTM performance for AD and AC

The window size that achieved the best performance on the validation set (Table I) was used to measure the LSTM performance on the test set. The accuracy of LSTM and the comparison with non-time-series ML algorithms is presented in Table II, for both AD and AC. The non-time-series ML algorithms that were considered include: Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), and Multi-Layer Perceptron (MLP). The parameters used for these algorithms can be found in paper [4].

It can be observed that LSTM has the highest accuracy, obtaining a better performance than the best non-time-series algorithm (MLP) by 2.7 and 0.12 percentage points for AD and AC, respectively. This again proves that considering the previous data can help detect anomalous behavior. Additionally, it can be noticed that LSTM that uses reduced data also has higher accuracy than all non-time-series algorithms for AD, while it is better than all of them and almost equal to MLP for AC. This means that historical information can bring more benefits than using data from all sensors without considering the temporal nature of the process.



(a) Sigmoid activation function with full data

(b) Sigmoid activation function with reduced data

(c) Softmax activation function with full data

(d) Softmax activation function with reduced data

Fig. 4: Hyper-parameters selection for LSTM network for AD - Heatmaps with accuracies of LSTM on the validation set

TABLE II: LSTM performance for AD and AC on testing set and comparison with non-time-series ML algorithms

Problem / Algorithm	AD	AC
LR	64.60	51.42
DT	69.97	56.53
RF	75.07	62.49
MLP	82.36	70.73
LSTM (full data)	**85.06**	**70.85**
LSTM (reduced data)	84.49	69.89

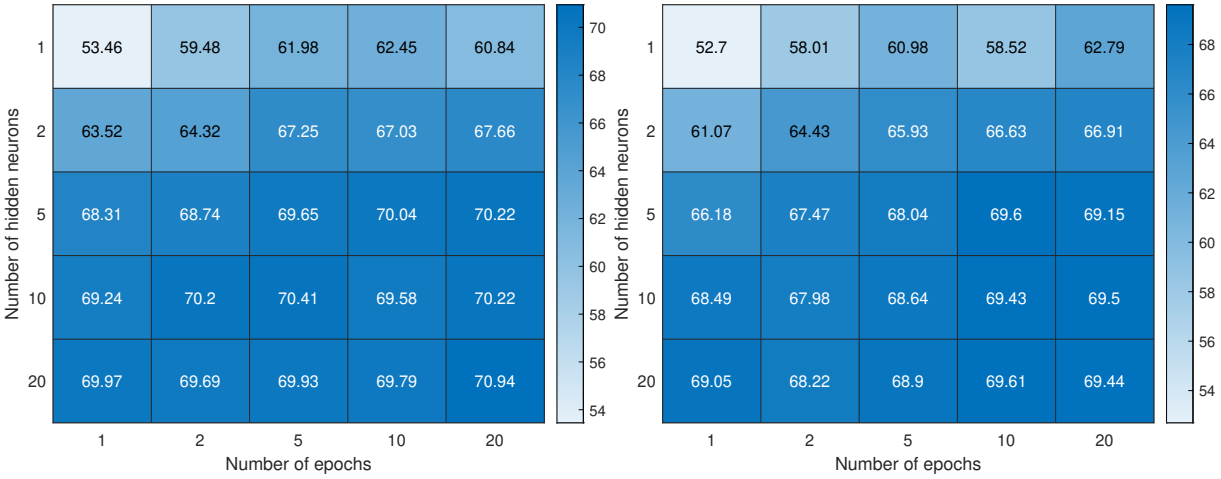
D. Runtime adaptation of LSTM prediction

The performance of LSTM on validation set after making a runtime adaption of the predictions is shown in Figure 6. It can be observed that the best results were achieved if all the previous time points ($r = 1$) in the selected period are predicted as anomalous. The best results were achieved when the amount of previous time points was 390. This means that the system needs around 8 seconds of continues positive predictions to be certain that an anomaly happened. Additionally, it can be noticed that after the selected number of time points (t) the accuracy is stabilized. Comparing all considered combinations, they have the same trend. However, with smaller number of required ones, the system needs more time to be certain about the anomaly, which results in lower overall accuracy.

After applying real-time adaption on the testing set, the overall accuracy of the model improved in average 1.8 percentage points, resulting in overall average accuracy of 86.9%. The actual performance of the runtime adaptation is showed in Fig. 7. It can be noticed that by adding the runtime adaption some of the wrong predictions were successfully fixed (some of the false positive and negative alarms were prevented).

E. Time and memory consumption

An important aspect to consider is the time that LSTM requires to create the model, as well as the time that is needed to apply the model to detect/classify the anomalies in a real-time environment. Firstly, a comparison between model training and testing time depending on the window size and



(a) Softmax activation function with full data

(b) Softmax activation function with reduced data

Fig. 5: Hyper-parameters selection for LSTM network for AC - Heatmaps with accuracies of LSTM on the validation set

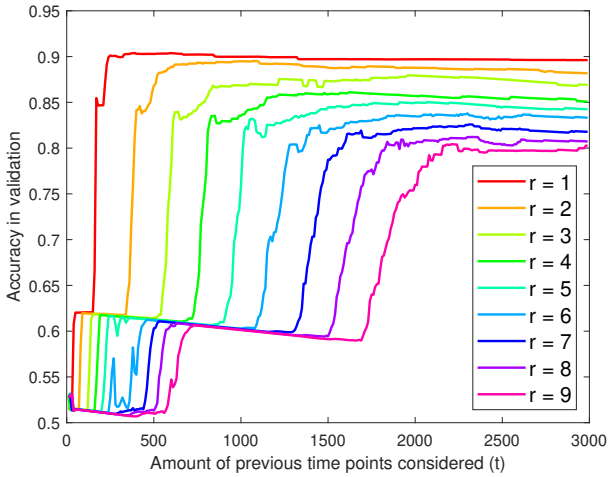


Fig. 6: Accuracy of adapted LSTM on validation set for different combinations of amount of previous time points considered (t) and the rate of anomalous predictions (r)

the amount of data (full data and reduced data) is given in Fig. 8. It can be observed how the time increases linearly for training (Fig. 8a), except when the full data with window size 20 is used. The main reason for this increase in time is that the RAM memory is not large enough to allocate the whole dataset, so the computer is forced to use the hard-drive to load the data. Secondly, if the testing time is considered (Fig. 8b), it can be noticed that all models with all window sizes would be able to run on a real-time system. The time between data readings within the system is equal to 25 ms, which is much higher than the slowest LSTM model that was evaluated (which requires less than 0.05 ms).

Additionally, the time used by LSTM and the non-time-series ML algorithm are compared with respect to training and testing time (Table III). Using LSTM requires more time

TABLE III: Average time of LSTM for training on entire training set and testing of one single case and comparison with non-time-series ML algorithms

Algorithm \ Problem	AD		AC	
	Train (s)	**Test (ms)**	**Train (s)**	**Test (ms)**
LR	889.06	0.00032	4520.92	0.00027
DT	415.44	0.00048	447.42	0.00063
RF	1493.27	0.00432	2197.63	0.00576
MLP	11362.16	0.00052	17925.70	0.00110
LSTM (full data)	45367.63	0.04386	61860.21	0.04328
LSTM (reduced data)	3028.62	0.04475	4702.19	0.04509

than the non-time-series ML algorithms for both training and testing, which is expected since the LSTM uses more data points within every example. When focusing only on the testing time, it can seen that the LSTM is 84 times slower than non-time-series version of ANN (MLP) for AD, while it is 39 times slower for AC.

Finally, the memory used by the training data, for both full and reduced data, is given in Fig. 9. It can be seen that the required memory increases linearly, however, with the full data from the window size 10, the dataset requires more memory than the amount of memory available in the RAM. This also proves that the limit of RAM is the reason why time increases exponentially after window size 10 (as shown in Fig. 8a).

VI. Conclusions and Future Work

In this paper, LSTM is applied to detect and classify anomalies in the manufacturing system. Experiments were conducted on the synthetically generated dataset called MIDAS that contains various anomalies in analog sensors data during ice cream making process and realistically represents a manufacturing process. The results showed that considering time-series nature of the data is beneficial for the accuracy of both detection and classification of anomalies. The LSTM had better performance than all non-time-series ML algorithms

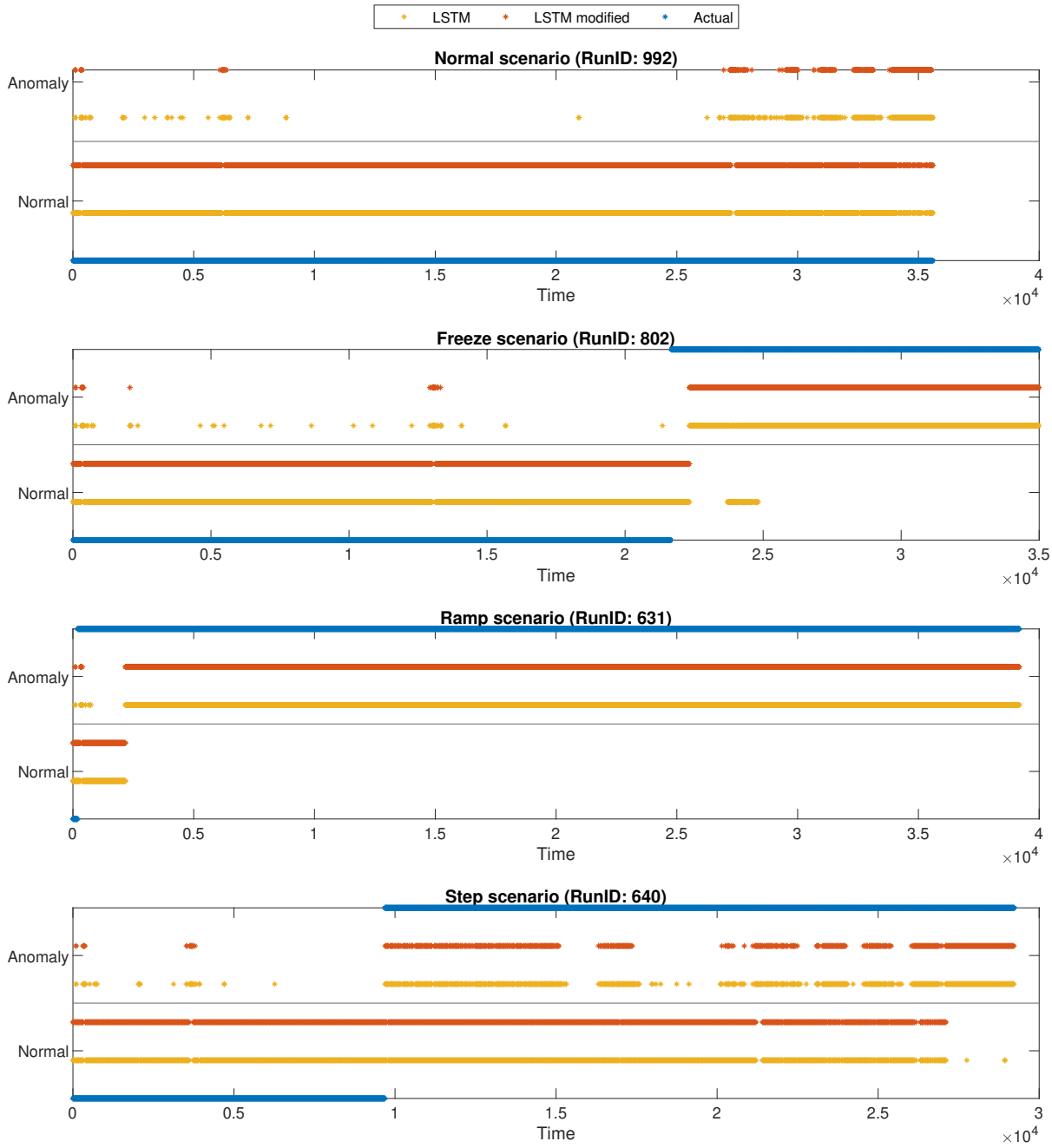


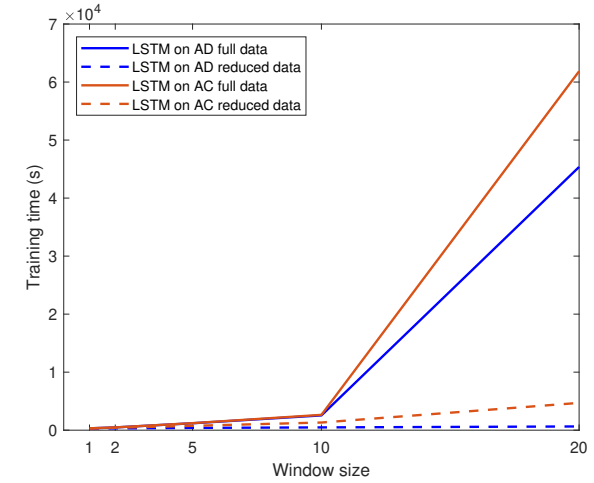
Fig. 7: Predictions of LSTM and LSTM after runtime adaptation (LSTM modified) for 4 full scenarios from the testing set for anomaly detection. Every time-point prediction is plotted.

it was compared with. On the other hand, it is more time consuming, but it can still run on a real-time system. Furthermore, the predictions generated by LSTM were adapted during runtime, with a goal to improve its performance and applicability in real systems.

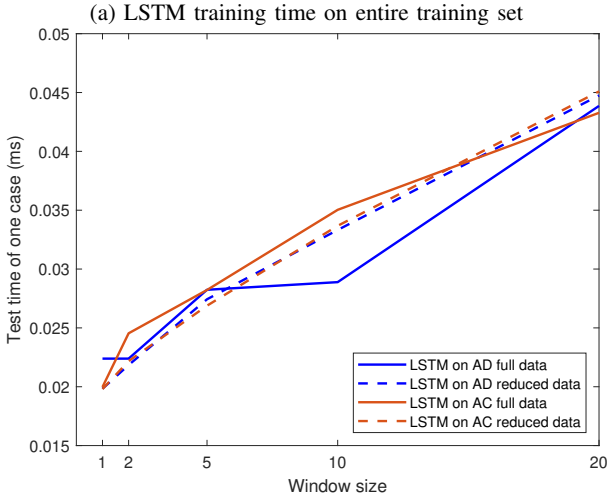
As a future work, we plan to integrate the proposed model in the simulation environment that is used to generate MIDAS dataset. Additionally, we plan to develop a federated learning approach for LSTM that will be able to distribute the AI on different system levels.

ACKNOWLEDGMENT

This work has been partially supported by the H2020 ECSEL EU projects Intelligent Secure Trustable Things (InSecTT) and Distributed Artificial Intelligent System (DAIS). InSecTT (www.insectt.eu) has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876038 and DAIS (https://dais-project.eu/) has received funding from



(a) LSTM training time on entire training set



(b) LSTM testing time for one case

Fig. 8: Time comparison of LSTM during training and testing with different window sizes.

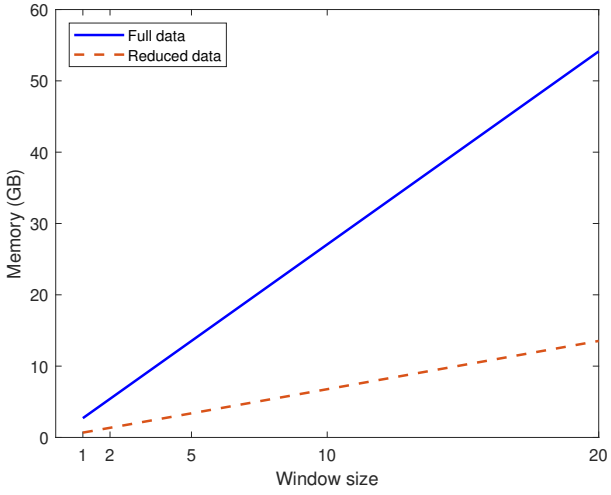


Fig. 9: Memory used by the training data

the ECSEL JU under grant agreement No 101007273. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey.

REFERENCES

[1] B. Esmaeilian, S. Behdad, and B. Wang, "The evolution and future of manufacturing: A review," *Journal of manufacturing systems*, vol. 39, pp. 79–100, 2016.

[2] A. K. Choudhary, J. A. Harding, and M. K. Tiwari, "Data mining in manufacturing: a review based on the kind of knowledge," *Journal of Intelligent Manufacturing*, vol. 20, pp. 501–521, 2009.

[3] T.-c. Fu, "A review on time series data mining," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

[4] T. Markovic, M. Leon, B. Leander, and S. Punnekkat, "A modular ice cream factory dataset on anomalies in sensors to support machine learning research in manufacturing systems," *IEEE Access*, vol. 11, pp. 29 744–29 758, 2023.

[5] K. Choi, J. Yi, C. Park, and S. Yoon, "Deep learning for anomaly detection in time-series data: review, analysis, and guidelines," *IEEE Access*, 2021.

[6] B. Lindemann, B. Maschler, N. Sahlab, and M. Weyrich, "A survey on anomaly detection for technical systems using lstm networks," *Computers in Industry*, vol. 131, p. 103498, 2021.

[7] A. A. Cook, G. Mısırlı, and Z. Fan, "Anomaly detection for iot time-series data: A survey," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2019.

[8] S. Schmidl, P. Wenig, and T. Papenbrock, "Anomaly detection in time series: a comprehensive evaluation," *Proceedings of the VLDB Endowment*, vol. 15, no. 9, pp. 1779–1797, 2022.

[9] Y. Wang, N. Masoud, and A. Khojandi, "Real-time sensor anomaly detection and recovery in connected automated vehicle sensors," *IEEE transactions on intelligent transportation systems*, vol. 22, no. 3, pp. 1411–1421, 2020.

[10] G. Shah and A. Tiwari, "Anomaly detection in iiot: A case study using machine learning," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 2018, pp. 295–300.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] W. Jia, R. M. Shukla, and S. Sengupta, "Anomaly detection using supervised learning and multiple statistical methods," in *2019 18th IEEE International Conference On Machine Learning and Applications (ICMLA)*. IEEE, 2019, pp. 1291–1297.

[13] G. Hong and D. Suh, "Supervised-learning-based intelligent fault diagnosis for mechanical equipment," *IEEE Access*, vol. 9, pp. 116 147–116 162, 2021.

[14] M.-C. Lee, J.-C. Lin, and E. G. Gan, "Rere: A lightweight real-time ready-to-go anomaly detection approach for time series," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 322–327.

[15] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[16] D. Y. Oh and I. D. Yun, "Residual error based anomaly detection using auto-encoder in smd machine sound," *Sensors*, vol. 18, no. 5: 1308, 2018.

[17] B. Lindemann, N. Jazdi, and M. Weyrich, "Anomaly detection and prediction in discrete manufacturing based on cooperative lstm networks," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020, pp. 1003–1010.

[18] Y. Zhang, Y. Chen, J. Wang, and Z. Pan, "Unsupervised deep anomaly detection for multi-sensor time-series signals," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[19] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 1409–1416.

[20] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Multivariate time-series anomaly detection via graph attention network," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 841–850.

[21] Z. Niu, K. Yu, and X. Wu, "Lstm-based vae-gan for time-series anomaly detection," *Sensors*, vol. 20, no. 13, pp. 3738–3750, 2020.

[22] R. Corizzo, M. Ceci, G. Pio, P. Mignone, and N. Japkowicz, "Spatially-aware autoencoders for detecting contextual anomalies in geo-distributed data," in *International conference on discovery science*. Springer, 2021, pp. 461–471.

[23] C.-Y. Hsu and W.-C. Liu, "Multiple time-series convolutional neural network for fault detection and diagnosis and empirical study in semiconductor manufacturing," *Journal of Intelligent Manufacturing*, vol. 32, pp. 823–836, 2021.

[24] M. Canizo, I. Triguero, A. Conde, and E. Onieva, "Multi-head cnn–rnn for multi-time series anomaly detection: An industrial case study," *Neurocomputing*, vol. 363, pp. 246–260, 2019.

[25] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at microsoft," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 3009–3017.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[27] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[28] B. Leander, T. Marković, A. Čaušević, T. Lindström, H. Hansson, and S. Punnekkat, "Simulation environment for modular automation systems," in *IECON 2022–48th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2022, pp. 1–6.

[29] B. Leander, T. Markovic, and M. Leon, "Enhanced simulation environment to support research in modular manufacturing systems," in *IECON 2023–49th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2023, pp. 1–6.

[30] M. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/