Making Systems Fail-Aware: A Semi-Supervised Machine Learning Approach for Identifying Failures by Learning the Correct Behavior of a System

Herbert Muehlburger [*] **Franz Wotawa** [**]

[*] *Institute of Software Technology, Graz University of Technology, Inffeldgasse 16b/II, 8010 Graz, Austria (e-mail: muehlburger@ist.tugraz.at).*
[**] *Institute of Software Technology, Graz University of Technology, Inffeldgasse 16b/II, 8010 Graz, Austria (e-mail: wotawa@ist.tugraz.at)*

Abstract: Observing the interaction between a system, its environment, and its internal state is vital to detect failures during operation. Monitoring systems often use predefined system properties to detect such failures, and violations indicate potential failures. However, obtaining these properties is work-intensive and error-prone. Therefore, we describe an approach to obtain a system model by learning only the correct behavior using machine learning. Monitoring systems can use such models to predict correct future behavior. A potential failure is raised if real-world data deviate significantly from this prediction. We use a semi-supervised LSTM-based forecasting approach with a simple architecture, apply our approach to simulation data from a battery control system, and discuss our experimental results.

Keywords: monitoring, fault detection, application of machine learning

1. INTRODUCTION

Systems have well-defined boundaries and interact with their surrounding environments during operation. Every system has a behavior, i.e., actions performed on the environment the system obtains from perceived information from the environment and its internal state. Over time, system performance may degrade, leading to undesired behavioral changes and finally preventing the system from performing its intended tasks. Degradation is not the only source of misbehavior. Attacks from outside or misinterpreting perceived information from its environment can also cause the system to fail. In any case, we must identify failures as early as possible and react with countermeasures to compensate for or correct detected faults.

In practice, we use monitoring systems to observe the behavior of a system over time. Applications of monitoring range from keeping track of performance (Lucas (1971); Zeng and Wang (2009)), identifying security issues (Casola et al. (2019)), to checking for safety (Wotawa and Lewitschnig (2022)). Monitoring systems often use properties to detect failures, that is, deviations from the expected behavior. If the system's observations violate a property, the monitoring system raises an alarm message. Such messages can either trigger human operators to take countermeasures or pass control. In any case, we have to specify the properties of a system, which is a time-consuming and work-intensive task. If we do not declare all the properties of the system, the monitoring system may fail to identify a critical situation. The monitoring system may deliver too many alarms or error messages if we specify weak properties. Considering the increasing complexity of today's systems comprising hardware and software components. It seems unlikely that we will obtain all of the properties of the system that we use for monitoring during development. Hence, there is a need to automate this task.

This paper suggests using machine learning to obtain models to be checked during operation. The idea is to observe the system's interactions with the environment and its internal state. Take this information and use a machine learning algorithm to get the system model representing the system's behavior. Later, this model is used to verify the behavior. If we learn the model when the system starts to operate, we may safely assume that the learned model only captures the correct behavior of the system. After learning, we use the model to predict the system's behavior over time. We consider deviations between the predictions and the observations to be failures. From this idea, several important research questions arise: *RQ-1*: Can our approach identify faults in a real-world setup? *RQ-2*: When should we stop learning, when should we go operational, and when should we restart learning? *RQ-3*: What effect do our machine learning model's parameter and setup choices have on the failure detection's overall performance? Finally, we want a general process that allows deploying such a monitoring approach to any system. In this paper, we focus on answering the first three questions raised. For this purpose, we introduce the architecture and implementation of the proposed machine learning monitoring approach that utilizes recurrent neural networks. In previous work, we described using an LSTM autoencoder model for intrusion detection (Mühlburger and Wotawa (2022)). In this work, we use a bidirectional LSTM – no autoencoder model – to detect anomalies in battery controller signals. The model uses a forecast-based time-series anomaly detection approach and sensor data from battery controllers deployed within electrified vehicles. Recurrent Neural Networks and Long Short-Term Memory (LSTM) networks specifically are capable of learning long-range temporal relationships in data (Hochreiter and Schmidhuber (1997)). When training LSTMs only on nominal data, it is possible to learn the nominal behavior of a system as shown in Bontemps et al. (2016). We use this approach and apply a bidirectional LSTM Graves et al. (2013) to learn the nominal behavior of our monitored cyber-physical system. It is possible to detect anomalies in the system behavior using this trained model. From the experimental analysis, we gain insights regarding the potential practical use of machine learning-based monitoring, which includes answering the questions about when to stop learning and the influence of parameters on the overall performance.

Anomaly detection has been a widely researched area for system monitoring applications. Chandola et al. (2009) provided a comprehensive review of various anomaly detection techniques. Hyndman and Athanasopoulos (2013) discussed various time series forecasting techniques that can be applied for anomaly detection. More recently, Schlegl et al. (2021) pointed out that models need to represent normal behavior well in the case of semi-supervised learning to detect meaningful anomalies. Other recent papers on the detection of anomalies using machine learning approaches include Tuli et al. (2022); Pang et al. (2021); Chalapathy and Chawla (2019); Lindemann et al. (2021). Our work builds on these foundational techniques by adapting and extending them to detect system failures. We integrate forecast-based time series anomaly detection to enhance the identification of failures, allowing us to better understand the system's normal behavior and predict potential failures.

We organize this paper as follows. In Sec. 2, we formulate our problem and describe our method with its architecture and components. Sec. 4 presents the results of our experiments, including data analysis, experimental setup, ablation study, discussion, and threats to validity. Finally, we summarize the paper.

2. METHODOLOGY

We propose a semi-supervised, forecast-based approach for univariate time series. Therefore, we designed a Bidirectional Long Short-Term Recurrent Neural Network (Bi-LSTM) model that learns to forecast the nominal behavior of the monitored cyber-physical system (Fig. 1).

2.1 Problem Formulation

Imagine a monitoring device that captures sensor values from a monitored system over time. We may assume that the system behaves normally and contains no anomalies for a certain time. This time forms the basis for training a Bi-LSTM machine learning model to capture the nominal state of a system. To use captured sensor values as input sequences for such a machine learning model, we denote x as the input samples of length t. We use $x^{<1>}, x^{<2>}, ..., x^{<t>}$ to denote the t-th element of the input sequence x. t is an index into the position of the sequence. y is our output sequence and we use $y^{<1>}, y^{<2>}, ..., y^{<t>}$ to denote the t-th element of the output sequence y. The length of the input and output sequences is denoted by T_x and T_y, respectively. $X^{(i)}$ denotes the i-th training sample, and we use $X^{(i)<t>}$ to refer to the t-th element of the sequence in the training sample i. $T_x^{(i)}$ is the input sequence length of the training sample i. $T_y^{(i)}$ is the output sequence length of training example i. $y^{(i)<t>}$ refers to the t-th element of the output sequence in the training sample i. In our approach, we set $T_y^{(i)} = 1$ and $T_y = 1$, so for each training sample we generate an output value $T_y^{(i)}$. This represents a forecast-based method with univariate time series data. Our goal is to learn a good representation of the nominal state of a system to detect states that do not conform with this learned behavior (potential faults).

2.2 Overall Architecture

We use a monitoring device and record internal and external system properties, such as temperature, power consumption, and cooling. Our goal is to train a machine learning model on the nominal behavior of the monitored system. This model can identify potential faults by comparing its predicted future values with real-life data. An anomaly is detected if this comparison exceeds a threshold (Fig. 1). The architecture contains the following modules:

(1) *Preprocessing*: This step involves cleaning, normalizing, and transforming the raw data to ensure its quality and suitability for further analysis.
(2) *Windowing mechanism*: A technique that divides time-series data into smaller overlapping segments, allowing for more efficient analysis and improved model performance.
(3) *Forecasting model*: A machine learning algorithm responsible for learning the system's normal behavior and making predictions based on the pre-processed data and windowing mechanism.
(4) *Anomaly scoring*: A method that sets boundaries for detecting anomalies between nominal behavior and deviations by comparing the forecasting model's predictions with predefined thresholds.

2.3 Model Architecture

Our Bi-LSTM-based model has the following model architecture:

(1) The model starts with a Lambda layer that reshapes the input data into the shape (Samples, Timesteps, Features). This layer does not introduce any trainable parameters.
(2) The first bidirectional LSTM layer consists of 120 hidden units with 29,760 trainable parameters. This layer is followed by a dropout layer with a dropout rate of 0.5, to prevent overfitting during training.
(3) The second bidirectional LSTM layer also contains 120 hidden units and has 86,880 trainable parameters.

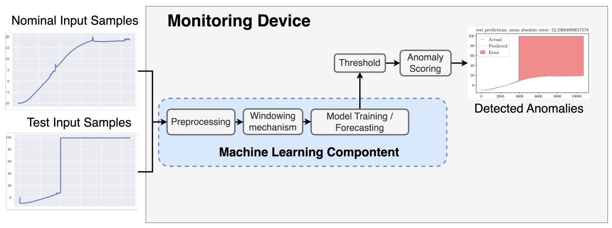


Fig. 1. Overview of Semi-Supervised Machine Learning Approach for Monitoring Devices. Input data are time series values. First, nominal input samples train a forecast-based machine learning model. Second, a threshold based on this anomaly score is calculated for each value, enabling the detection of potential anomalies.

We also use a dropout rate of 0.5 after this bidirectional LSTM layer.

(4) Next, a unidirectional LSTM layer with 60 hidden units is included, adding 43,440 trainable parameters to the model.
(5) Finally, a Dense layer with a single output unit and 61 trainable parameters is used to produce the model output.

In total, the model has 160,141 trainable parameters and no non-trainable parameters. The input to the network is a sequence $x^{(i)}$ with a length of n_steps (Sec. 2.4.2). The output layer generates output sequences $y^{(i)}$ with one element. We trained in 50 epochs with a batch size of 120. We use the gradient descent optimization algorithm with a momentum of 0.9, a learning rate of 1×10^{-3} (Fig. 2), and the Huber loss criterion for training.

2.4 *Implementation*

We implemented[1] our model using Tensorflow[2] and Kedro[3] machine learning frameworks.

Data Preprocessing In the preprocessing steps, we load the raw data consisting of multivariate time series. We select a univariate time series from the multivariate time series for further processing. We normalize the features by scaling the values to a feature range of [0,1].

$$x_{\text{std}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$
$$z = x_{\text{std}} \times (\max - \min) + \min$$

x_{std} represents the standardized and z the scaled versions of x. $\min(x)$ denotes the minimum values and $\max(x)$ the maximum values in x. max and min are used to represent the specified scaling range.

For training and validation, we only used data that contain normal behavior. For testing, we used data containing normal and possibly faulty behavior.

Windowing Mechanism We aim to transform the time series prediction into a supervised learning problem. Therefore, we span a window of n_steps past values to predict a *horizon* of future values at time T. We set $horizon = 1$, which means that we only forecast one future value. We tested different values for n_steps and found that $n\_steps = 60$ works well.

Threshold Calculation Previous work suggests calculating anomaly scores using reconstruction, forecast, or prediction errors (Luo et al. (2020); Koutroulis et al. (2023); Hundman et al. (2018)). We calculate the forecast error and measure the forecasts' mean absolute error (MAE) on the training set. We take the maximum error on the training set. Our model follows a semi-supervised learning approach, as our training set contains no anomalies. The forecast error on the training set is lower than on the test set, which contains potential anomalies. During anomaly scoring, we use the test set, generate forecasts, and compare the forecast error to the calculated threshold of the training set. If the error exceeds this threshold, an anomaly is detected.

3. EXPERIMENTAL SETUP

To answer the research questions from Sec.1, we evaluated our approach experimentally using the following dataset.

3.1 *Dataset Description*

Our dataset consists of multiple time series readings produced by a Cyber-Physical System (CPS), concretely a battery management system in an electrified vehicle. It contains sensor values of temperature, power consumption, velocity, and airflow of the cooling system and has 90 minutes (5,400.5 seconds) of continuous readings, resulting in 10,801 total samples. We divided it into training and validation in a proportion of 95% (10,260 training samples) and 5% (541 validation samples). For evaluation, we used a separate test dataset.

We collected the dataset by simulating a battery management system in an electrified vehicle (CSP) with an array of sensors installed at different locations. These sensors continuously monitored and recorded various system parameters under different operating conditions. Some signals in the dataset are identical; therefore, we selected 4 types of signals from the 13 available. The first two signals represent temperature sensors (temp5 and temp6). Two other sensors measure the electric current (amperage) and airflow in liters per second from a cooling device (volflow). Before training our model, we preprocessed the dataset by transforming each time series into a sequence x with input length $T_x = 60$ and output length $T_y = 1$. The sequence x contains one signal from the dataset. The resulting dataset represents a supervised learning problem, forecasting one value based on historical values.

We aim to train a Bi-LSTM machine learning model using this dataset to detect anomalies in the cyber-physical system's operation. The performance of the model is evaluated on a separate test dataset.

3.2 *Model Training*

We train our model using input sequences to predict future values accurately. After training, the model can forecast

[1] Source code and data is available under an open-source license: https://github.com/muehlburger/safeprocess2024-making-systems-fail-aware
[2] https://www.tensorflow.org, Accessed October 20, 2023.
[3] https://kedro.org, Accessed October 2nd, 2023.

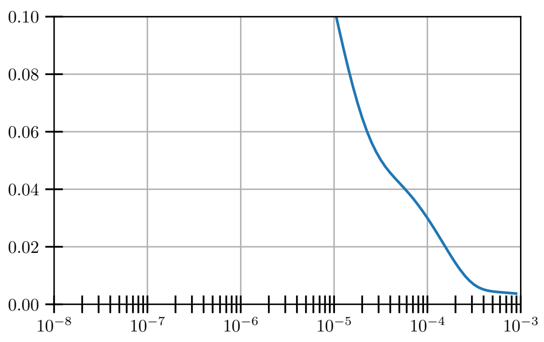


Fig. 2. Optimization of the learning rate for the 'temp5' model. A learning rate of 1×10^{-3} was identified as optimal, balancing model accuracy and training efficiency.

the next values based on historical sensor readings. When forecasting nominal input sequences, our model has a low forecast error. We use the low forecast error to define a threshold and compare it with the forecast errors of the test sequences to detect potential anomalies. Fig. 3 shows the loss and mean absolute error (MAE) for the training and validation sets. The errors decrease steadily, indicating that the model is able to learn the data well.

3.3 Evaluation Metrics

We evaluate the performance of our model using the mean absolute error and the mean squared error metrics following Hyndman and Athanasopoulos (2013) and Willmott and Matsuura (2005). The lower the values, the better the forecast performance of the model.

Mean Absolute Error (MAE) measures the mean error in the absolute values between the predictions $\hat{y}$ and the actual values y.

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| \,.$$

Mean Squared Error (MSE) measures the mean error in squared values between predictions $\hat{y}$ and actual values y and is therefore more sensitive to larger errors.

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

Mean Absolute Percentage Error (MAPE) measures the mean of the percentage error of the difference between predictions $\hat{y}$ and actual values y.

$$\text{MAPE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \frac{|y_i - \hat{y}_i|}{\max(\epsilon, |y_i|)}$$

4. RESULTS

Our results highlight the model's precision in forecasting and anomaly identification, with detected anomalies closely correlating with the simulated faults. These findings, illustrated for models "temp5" and "amperage" in Figs. 4 and 3 underscore the potential of our approach in enhancing monitoring systems' sensitivity to early signs of system failures. Fig. 4 shows the predictions in the train set, test set, the errors, and detected anomalies.

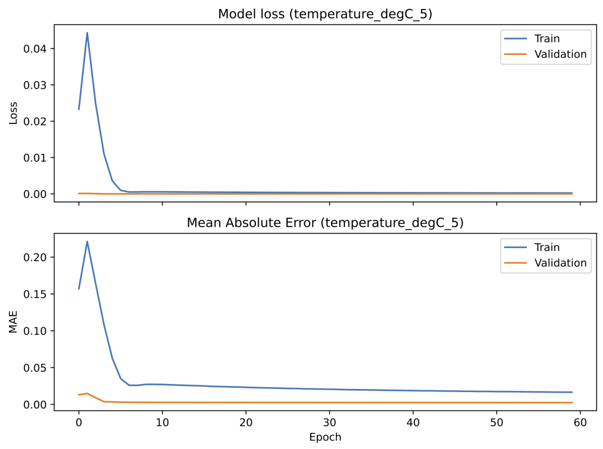


Fig. 3. Training history for the 'temp5' model, demonstrating the model's convergence over 60 epochs. This efficiency supports the feasibility of frequent re-training to adapt to new system behaviors.

4.1 Model Performance

Table 1 presents the evaluation of our models temp5, temp6, volflow, and amperage, on various metrics such as Mean Absolute Error (MAE), and the count of detected faults for both training and testing. The ability of temp5 and volflow to detect 6800 anomalies in the test set, as opposed to 1156 in 'temp6', underscores the variability in model performance and highlights the challenge of learning the intricate dynamics of CPS features like amperage, where we observed a high number of false positives (8767 anomalies detected). We selected 60 epochs, a learning rate of 0.001, a batch size of 120, a window size of 60, and a train / test split of 95% train and 5% test data.

4.2 Analyzing Anomalies and Fault Detection

Our method's ability to distinguish between nominal and anomalous states is further evidenced by the detailed analysis of temp5 with different window sizes shown in Table 2. The variation in detected anomalies with changing window sizes demonstrates the model's responsiveness to temporal features of the data, which is crucial for real-time anomaly detection.

The computational efficiency of our approach, validated on an Apple MacBook Pro 14-inch with an M1 chip and 16GB of RAM, underscores the feasibility of deploying such models in real-world CPS with limited computational resources. Each experiment was carried out several times, and average values are reported. Our experiments aimed to find a small window size for model training to produce good forecasts.

5. DISCUSSION

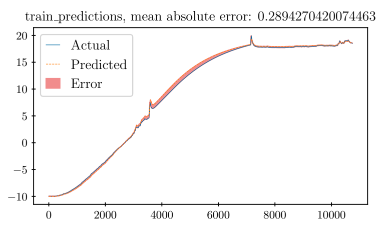
Our approach can identify faults in a real-world setup (see Table 1). Therefore, we can confirm RQ-1. Our models for temp5, temp6, and volflow detected faults on

Table 1. Results for all models. For each model, we report the calculated threshold, the mean absolute error (MAE), and the detected faults in the training and test datasets.

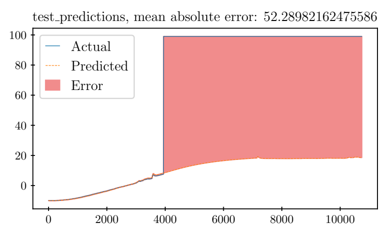
Model	Data	MAE	MSE	MAPE	Window Size	Epochs	Threshold	Anomalies detected
amperage	test	18.33	672.86	209.10	60	60	6.65	8766
amperage	train	18.34	673.26	208.77	60	60	6.65	8767
temp5	test	52.29	4309.47	55.68	60	60	1.05	6800
temp5	train	0.29	0.14	4.60	60	60	1.05	0
temp6	test	0.44	0.66	1.82	60	60	1.37	1156
temp6	train	0.15	0.04	0.62	60	60	1.37	0
volflow	test	6.53	67.14	52.38	60	60	1.70	6800
volflow	train	0.14	0.04	1.76	60	60	1.70	0

Table 2. Results for temp5 with different window sizes. The model was trained with a window size of 60.

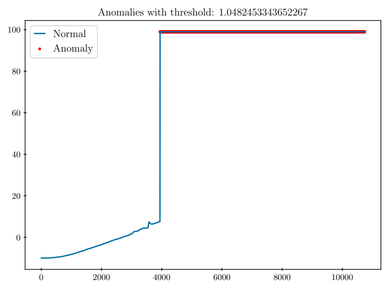
Model	Data	MAE	MSE	MAPE	Window Size	Epochs	Threshold	Anomalies detected
temp5	test	68.41	7035.16	625063.56	1	60	1.05	10366
temp5	test	64.95	6392.36	208.10	5	60	1.05	10287
temp5	test	60.22	5566.50	131.75	10	60	1.05	10077
temp5	test	52.63	4382.89	54.96	30	60	1.05	6800
temp5	test	52.29	4309.47	55.68	60	60	1.05	6800
temp5	train	16.30	349.05	346.60	1	60	1.05	10366
temp5	train	12.81	214.76	225.08	5	60	1.05	10287
temp5	train	8.07	84.28	120.79	10	60	1.05	10077
temp5	train	0.40	0.25	3.94	30	60	1.05	43
temp5	train	0.29	0.14	4.60	60	60	1.05	0



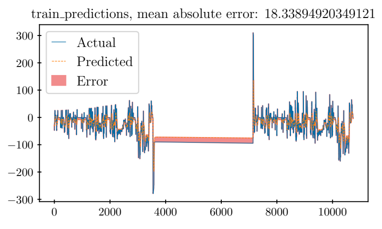
(a) Temp5 - Training Predictions



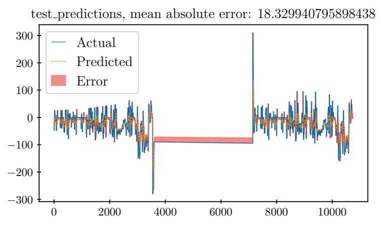
(b) Temp5 - Test Predictions



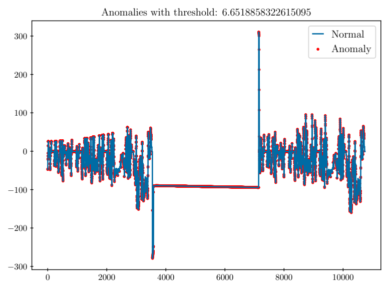
(c) Temp5 - Test Anomalies



(d) Amperage - Training Predictions



(e) Amperage - Test Predictions



(f) Amperage - Test Anomalies

Fig. 4. Model Predictions and Anomaly Detection for "temp5" and "amperage" models. The figure presents the predictions of the model during training, the test performance with the mean absolute error, and the detected anomalies using a predefined threshold for the training set. It illustrates the ability of the bidirectional LSTM model to learn normal behavior and detect deviations, demonstrating its potential for real-world system monitoring.

the test set, and a manual analysis confirmed that the detected anomalies are true positives. We experimented with different parameters, such as window size, learning rate, number of epochs, and different train/test splits. We were unable to find reliable information to answer RQ-2 unquestionably. One possibility is to perform an error analysis of the training history. Fig. 3 shows the training history of temp5, and as we can see, it was sufficient to train for 60 epochs. So, we can stop learning after 60 epochs and go operationally fast, as the training time is less than two minutes. Further, it enables the model to be re-trained after a certain time to learn new nominal behavior of the system.

To answer RQ-3 we calculated thresholds for each model that predicted no false positives on the training sets and found that a window size of 60 has a suitable performance on our dataset. We trained for not more than 60 epochs, which is rather low. To find an appropriate learning rate to train the temp5 model, we used a learning rate scheduler, testing different learning rates and selecting 1×10^{-3} as optimal (Fig. 2).

The mean absolute error is highly dependent on the possibility that the model fits well and the ability to learn the nominal behavior. Due to our setup, it is not possible to evaluate if all anomalies are found because we only rely on the fact that no anomalies are present in the training set (semi-supervised learning). Window size is an important parameter in time series, as it defines the historical values that are used to predict the future. A bigger window size enables the model to can more information, but it also requires more resources (compute, memory) and has the potential to overfit (see Tables 1 and 2).

6. CONCLUSION

In conclusion, this paper introduces a semi-supervised, Bi-LSTM-based forecasting model for anomaly detection in CPS. Distinguished by its focus on learning nominal behavior and evaluated through a simulation dataset, our approach signifies a step forward in developing fail-aware systems capable of adapting to real-world complexities. Future directions include expanding our dataset to encompass wider operational spectra and exploring the integration of additional forecasting techniques for enhanced anomaly detection capabilities.

ACKNOWLEDGEMENTS

ArchitectECA2030 receives funding within the Electronic Components and Systems For European Leadership Joint Undertaking (ESCEL JU) in collaboration with the European Union's Horizon2020 Framework Programme and National Authorities, under grant agreement number 877539. All ArchitectECA2030 related communication reflects only the author's view and the Agency and the Commission are not responsible for any use that may be made of the information it contains. The work was partially funded by the Austrian Federal Ministry of Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK) under the program "ICT of the Future" project 877587.

REFERENCES

Bontemps, L., Cao, V.L., McDermott, J., and Le-Khac, N.A. (2016). Collective Anomaly Detection Based on Long Short-Term Memory Recurrent Neural Networks. In T.K. Dang, R. Wagner, J. Küng, N. Thoai, M. Takizawa, and E. Neuhold (eds.), *Future Data and Security Engineering*, volume 10018, 141–152. Springer International Publishing, Cham.

Casola, V., De Benedictis, A., Riccio, A., Rivera, D., Mallouli, W., and de Oca, E.M. (2019). A security monitoring system for internet of things. *Internet of Things*, 7, 100080.

Chalapathy, R. and Chawla, S. (2019). Deep Learning for Anomaly Detection: A Survey. *arXiv:1901.03407 [cs, stat]*.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3), 15:1–15:58.

Graves, A., Mohamed, A.r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 6645–6649.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.

Hundman, K., Constantinou, V., Laporte, C., Colwell, I., and Soderstrom, T. (2018). Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, 387–395. Association for Computing Machinery, New York, NY, USA.

Hyndman, R.J. and Athanasopoulos, G. (2013). Forecasting: Principles and practice.

Koutroulis, G., Mutlu, B., and Kern, R. (2023). A Causality-Inspired Approach for Anomaly Detection in a Water Treatment Testbed. *Sensors*, 23(1), 257.

Lindemann, B., Maschler, B., Sahlab, N., and Weyrich, M. (2021). A Survey on Anomaly Detection for Technical Systems using LSTM Networks. *arXiv:2105.13810 [cs, stat]*.

Lucas, H. (1971). Performance evaluation and monitoring. *ACM Comput. Surv.*, 3(3), 79?91.

Luo, Y., Xiao, Y., Cheng, L., Peng, G., and Yao, D.D. (2020). Deep Learning-Based Anomaly Detection in Cyber-Physical Systems: Progress and Opportunities. *arXiv:2003.13213 [cs, eess]*.

Mühlburger, H. and Wotawa, F. (2022). A passive testing approach using a semi-supervised intrusion detection model for scada network traffic. In *Proceedings of the 4th IEEE International Conference on Artificial Intelligence Testing (AITest)*. IEEE, San Francisco Bay Area, CA, USA.

Pang, G., Shen, C., Cao, L., and van den Hengel, A. (2021). Deep Learning for Anomaly Detection: A Review. *ACM Computing Surveys*, 54(2), 1–38.

Schlegl, T., Schlegl, S., West, N., and Deuse, J. (2021). Scalable anomaly detection in manufacturing systems using an interpretable deep learning approach. *Procedia CIRP*, 104, 1547–1552.

Tuli, S., Casale, G., and Jennings, N.R. (2022). TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data. *10.48550/arXiv.2201.07284*

Willmott, C. and Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30, 79–82.

Wotawa, F. and Lewitschnig, H. (2022). Monitoring hierarchical systems for safety assurance. In D. Camacho, D. Rosaci, G.M.L. Sarné, and M. Versaci (eds.), *Intelligent Distributed Computing XIV*, 331–340. Springer International Publishing, Cham.

Zeng, W. and Wang, Y. (2009). Design and implementation of server monitoring system based on snmp. In *2009 International Joint Conference on Artificial Intelligence*, 680–682.