Autoencoder-based method for online fault detection in discrete-event class Production Systems

Ramla SADDEM, Dylan Baptiste, Ange Patrick Wabo Teingua

2024 WODES
Workshop on Discrete Event Systems
Rio de Janeiro, Brazil

17th IFAC Workshop on Discrete Event Systems
April 29-May 1, 2024. Rio de Janeiro, Brazil

Autoencoder-based method for online fault detection in discrete-event class Production Systems

Ramla Saddem, Dylan Baptiste and Ange Patrick Wabo Teingua,

CReSTIC, University of Reims Champagne-Ardenne, France
(email : firstname.name@univ-reims.fr)

Abstract: In the context of discrete-event systems (DES), the terms detection and diagnosis refer to two distinct stages of handling faults and anomalies. Both steps are critical for ensuring the reliable and safe operation of complex systems. In this paper, we propose the use of autoencoders for fault detection in an automated production system with sensors and actuators delivering discrete binary signals that can be modeled as DES. We train an autoencoder exclusively on data representing normal behavior. The model learns to encode typical patterns and reconstruct input data with low loss. A predetermined threshold, determined by the characteristics of the training data, is set for the reconstruction error. During normal behavior, the autoencoder is expected to achieve low reconstruction error below this threshold. When a fault occurs, the autoencoder strives to accurately reconstruct faulty data, leading to a higher error. The detection of a reconstruction error exceeding the threshold signals a potential fault in the system. The results of applying our method to the Factory IO software sorting system demonstrate the significant contribution and the interest of this method for detecting faults.

Keywords: Autoencoder, Fault detection, Automated Production System, Deep Learning, discrete-event systems.

1. INTRODUCTION

Productivity reductions, delays, and system disruptions are all potential outcomes of system faults. To preserve the functionality and dependability of production systems, it is necessary to develop efficient fault detection techniques. A fault is a deviation that can cause a system to derive from its expected behavior, which could potentially lead to accidents and damage to equipment and people. In this study, we are interested in Automated Production Systems (APS) fault detection. The literature proposes different approaches dealing with this problem (Ghosh et al. (2020)) and distinguishes three classes according to the dynamics of the APS: the class of continuous systems, the class of discrete-event systems (DES) and the class of hybrid dynamic systems (HDS). In this paper, we focus on APS with sensors and actuators delivering logical signals, which fall under the DES.

1.1 Motivation

In previous works (Saddem and Baptiste (2022, 2023)), we have proposed a supervised learning method: recurrent neural networks (RNN) with short-term and long-term memory (LSTM) based models to predict the plant state: normal or faulty and diagnoses what fault has happened in case of fault in a DES plant. The diagnosis system is considered as a multi-class classifier that predicts the plant state: normal or faulty and diagnoses what fault has happened in case of fault. However, the classification produced by RNN is not yet easy for a human operator to explain. In this paper, we propose an unsupervised learning method based on autoencoders for fault detection in APSs of DES class. The next section provides a brief literature review of the context background. Section 2 delves into the fundamentals of Machine Learning (ML), followed by an introduction to encoders. Section 3 presents our approach. In section 4 we describe an example of an APS on which we will rely to illustrate our approach and we present the results. Finally, we conclude the paper with some prospects in section 5

1.2 Brief literature review

The state of the art in fault detection for DES continues to evolve with the integration of advanced technologies, including ML, big data analytics, and IoT sensors. DES diagnosis approaches can be categorized into three main families based on the reasoning mode: model based, knowledge-based, and data-based approaches. Model based approaches (Sampath et al. (1995); Debouk et al. (2000); Zaytoon and Lafortune (2013); Alzalab et al. (2021)) are efficient when there is sufficient knowledge of the system but require accurate and costly analytical models. Knowledge based approaches (Subias et al. (2014); de Souza et al. (2020)) exhibit high diagnosis capacity but face challenges in formalizing and updating expert knowledge. Data-based approaches Venkatasubramanian et al. (2003); Han et al. (2018); Saddem and Baptiste (2023) do not necessitate knowledge of internal workings and do not need an explicit formal model, relying on historical data and ML techniques for fault diagnosis. Despite not requiring

explicit models, they demand a data preparation step to extract relevant information for formatting in line with the chosen ML technique.

2. PRELIMINARIES

2.1 Machine learning models steps

In this paper, we propose a new solution for online fault detection of APSs that have discrete dynamics. ML models typically involve a series of steps or phases in their development and deployment (Xie et al. (2021)). These steps can be summarized as follows:

i Data Collection: Gathering relevant and high-quality data is the first step. This data serves as the foundation for training and testing the ML model. It may involve data from various sources, including databases, sensors, etc.
ii Data Preprocessing: Cleaning and preparing the data to ensure it is suitable for training. This may involve handling missing values, normalizing data, encoding categorical variables, and other data transformations.
iii Feature Engineering: Selecting or creating the most informative features for the model. Feature engineering aims to improve the model's performance by providing relevant informations to the learning algorithm.
iv Model Selection: Choosing the appropriate ML algorithm or model architecture for the task. This decision depends on the problem nature, the available data, and the desired outcome.
v Model Training: Using prepared data to train the chosen model. During this phase, the model learns the underlying patterns and relationships in the data.
vi Model Evaluation: Assessing the model's performance using various metrics, such as accuracy, precision, recall, or mean squared error, depending on the specific problem. This step helps determine if the model is performing well and if adjustments are needed.
vii Hyperparameter Tuning: Fine-tuning the model's hyperparameters, such as learning rates or regularization parameters, to optimize its performance.
viii Model Validation: Ensuring the model's generalization ability by testing it on a separate dataset (validation set) not used during training. This helps detect issues like overfitting.
ix Model Deployment: Integrating the trained model into a production environment or application where it can make predictions on new, unseen data.
x Monitoring and Maintenance: Continuously monitoring the model's performance in the real-world application, and updating it as needed to account for concept drift or changes in the data distribution.
xi Interpretability and Explainability: Understanding and explaining how the model makes predictions, which is crucial for building trust and compliance in some applications

These steps represent a high-level overview of the ML model development process. The specific details and iterations can vary based on the complexity of the problem and the chosen ML approach. In this paper, we use autoencoders.

2.2 Autoencoder

An autoencoder (Fig 1) is an artificial neural network architecture. It is designed to learn efficient representations of data, typically for dimensionality reduction, feature learning, or data compression tasks. Autoencoders consist of an encoder and a decoder, and they work by attempting to reconstruct their input data. The input data is first passed through an encoder, which reduces the dimensionality of the data and produces a compressed representation, often referred to as a bottleneck or latent space. The compressed representation in the latent space encodes essential features of the input data. This space is typically of much lower dimension than the input data, capturing the most salient information. The compressed representation is then passed through a decoder, which attempts to reconstruct the original input data from this reduced representation. The goal during training is to minimize the reconstruction error, which is the difference between the input and the reconstructed output.

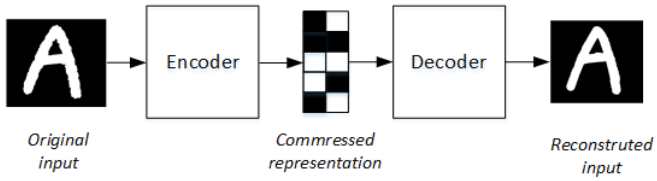

Fig. 1. Auto-encoder schematics

Autoencoder's objective is to minimize the difference between the input data and the data reconstructed by the decoder. This is typically achieved by using a loss function to measure the reconstruction error. During training, the autoencoder adjusts its weights and biases to minimize this error, thus learning a compact representation of the data. Autoencoders have various applications, including dimensionality reduction, feature learning, anomaly detection, Image Denoising, Data Generation, etc.

3. PROPOSED APPROACH

3.1 Automated Production System

An APS system consists of three parts: the operative part (OP), the control part (CP) and the Human Machine Interface (HMI) (Fig 2). The OP part represents all material resources that physically operate on the plant. The CP is the set of information processing and acquisition means that ensure the piloting and control of the process. There are two types of Information exchanges between the CP and the OP. The CP sends orders to the actuators and pre-actuators of the OP to obtain the desired effects. The OP sends sensors values to the CP. HMI allows the communication between the CP and the human operators. Human operator gives instructions via the HMI and receives from CP various signals such as light indicators, sound indicators, messages displayed on the screens, etc. Most of APS's that have sensors and actuators delivering binary (On/Off) signals, are controlled by Programmable Logic Controllers (PLC) that perform three successive operations: a) Reading the inputs, which consists in the recording of the states of sensors. b) Executing the program. C) Updating the outputs (actuators). These opera-

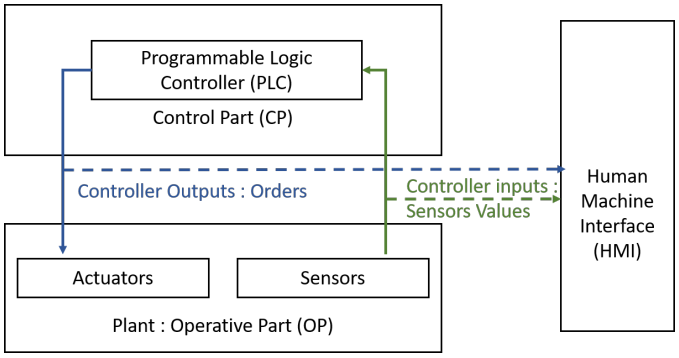

Fig. 2. Structure of an APS

tions are cyclically, i.e., one cycle after the other. Detection task consists in cyclically reading the sensors values and the CP's orders, and analyzing them in order to detect faults. Four faults are possible for each APS component: stuck to 0; stuck to 1; an unexpected move from 0 to 1 and an unexpected move from 1 to 0.

3.2 proposal

Our approach consists of two phases: offline phase and online phase. Offline phase includes data collection, data preprocessing and autoencoder training. For data collection, we use data acquisition application developed in (Saddem and Baptiste (2022)), to retrieve data from a PLC and save the input and output data in a database. For data preprocessing, we format the data to rows as shown in Fig 3. Then, we divide data into two sets: training data set and validation data set. Autoencoder is trained on training data set by minimizing the reconstruction error between the input data and the corresponding reconstructed output. Validation data set is used to monitor the model's performance during training and to tune hyperparameters or early stopping criteria. The reconstruction error on the validation set gives an indication of how well the model is generalizing to data it hasn't seen during training. There is no traditional test set for evaluating the final performance of an autoencoder.

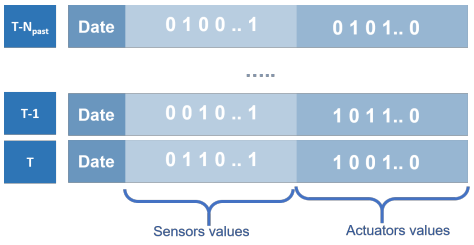

Fig. 3. Representation of a timed input vector

Collected data is unbalanced, some components keep true or false value most of the time. The model may be biased toward the most common value for his prediction. To counter this undesirable effect, a weighted binary cross-entropy (WBCE) loss function was used. This function considers this unbalanced representation and can compensate it. For time prediction, which is a classical regression problem, we have used the Mean Squared Error (MSE) function. Let define:
N: number of plant components
N_{past}: number of past observations to be given to autoencoder.
W_P: weights for the positives (true or 1) value for a feature.
W_N: weights for the negatives (false or 0) for a given feature.
For each component, its W_P and W_N associated weights are proportional to the inverse frequency of the presence of 1 and 0 in the dataset. Let consider x a plant component. Let n_0 and n_1 be the numbers of occurrences of 0 and 1 for x in the dataset. Then $W_P(x)$ and $W_N(x)$ were defined as follows:

$$W_P(x) = \frac{n_0 + n_1}{2n_1} \; and \; W_N(x) = \frac{n_0 + n_1}{2n_0} \tag{1}$$

Equation (2) presents the loss used for the training.

$$Loss(y, \hat{y}) = (1 - \lambda).WBCE(y_c, \hat{y}_c) + \lambda.MSE(y_T, \hat{y}_T) \tag{2}$$

With :

- $WBCE(y_c, \hat{y}_c) = -\frac{1}{N.N_{past}} \sum_{i=1}^{N} \sum_{j=1}^{N_{past}} W_P(i) y_{i,j} log(\hat{y}_{i,j}) + W_N(i)(1 - y_{i,j}) log(1 - \hat{y}_{i,j})$
- $MSE(y_T, \hat{y}_T) = \frac{1}{N} \sum_{t=1}^{N_{past}} (y_t - \hat{y}_t)^2$
- $\hat{y}$ the prediction vector,
- $\hat{y}_c$ the component prediction vector,
- $\hat{y}_T$ the time prediction vector,
- $\lambda \in [0, 1]$ is a parameter that allows to modulate the contribution of MSE regarding $WBCE$ to the overall loss.

We only train the autoencoder on data of normal behavior. The autoencoder learns to encode the normal patterns in the data and reconstruct them with low loss. The loss threshold is determined based on the characteristics of the training data. During normal behavior, the autoencoder should reconstruct the input data with low loss. The reconstruction error is expected to be below the threshold. When a fault occurs, the autoencoder might struggle to reconstruct the faulty data accurately, resulting in a higher reconstruction error. If the reconstruction error exceeds the predetermined threshold, it signals a potential fault in the system. The choice of the threshold is crucial and may require tuning based on the characteristics of the normal behavior and fault tolerance. Accuracy is a metric used to evaluate the performance of a ML model. It is a measure of how well the model correctly predicts the outcome for a given set of data. In Table 1 we have illustrated the accuracy measure on a simple three components system to explain it. In blue, the good prediction, in red the false prediction.

Table 1. Accuracy Measures

Measure	Ground Truth ⇒ Prediction
	[0,1,0]⇒[0,1,0] [0,1,1]⇒[1,1,0] [1,1,0]⇒[0,0,0]
accuracy	1 vector/3 33%

4. APPLICATION

4.1 Use case description

We applied our approach to the box sorting system, a virtual system simulated using the 3D simulation software Factory IO (Riera and Vigário (2017)) (Fig 4). The objective of the box sorting system is to bring boxes from an entry conveyor to an exit conveyor by sorting them according to their heights. The system has 16 sensors ($c0$ to $c15$) to determine boxes size (small or large), the box entry or exit in different conveyors (feeding, intermediate and evacuation) or turntable, and various dashboard buttons (Fig 5). It has also 7 actuators ($A0$ to $A6$) to activate the various conveyors and the turntable.

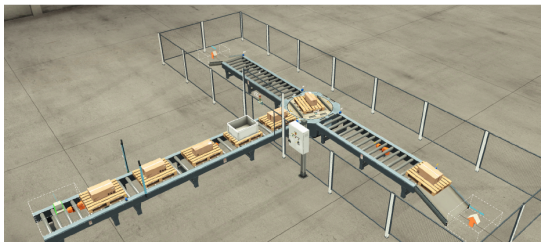

Fig. 4. Sorting by height plant

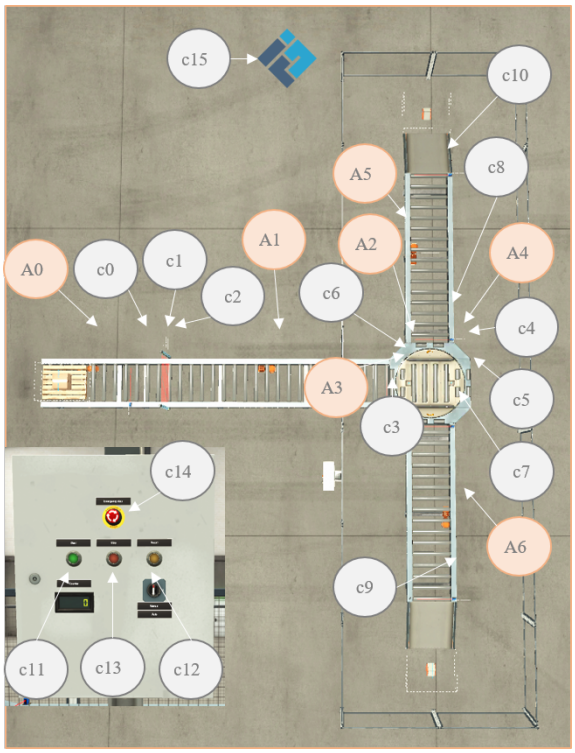

Fig. 5. Location of the various sensors and actuators in Sorting by height plant

A present gantry on the entry conveyor is composed of two sensors $c1$ and $c2$ at different heights. If $c2$ undergoes a rising edge the box is considered high enough and will be sent to the left by the rotary conveyor. If only $c1$ undergoes a rising edge, then the box will be sent to the right. The rotary conveyor is composed of a motor allowing its rotation at 90 degrees controlled by $A4$. $c4$ and $c5$ report its orientation (0 degree for $c4$ and 90 degrees for $c5$). $c6$ indicates if a box is loaded on the rotary conveyor. $A2$ and $A3$ allow respectively to move the box forward and backward on the rotating conveyor when it is in its initial position. When the rotating conveyor is oriented at 90 degrees, $A2$ moves the case to the left and $A3$ moves the case to the right. All other actuators ($A0$, $A1$, $A5$,$A6$) are used to operate the static conveyors. Each conveyor has a sensor at the beginning and at the end of its path.

4.2 Results

This section resumes the results of the application of our approach on the plant described previously. We have chosen $N_{past} = 5$ and ignored dashboard and Factory IO running sensors, to focus on system OP components. The encoder and decoder are each made up of 3 LSTMs layers, the latent layer is made up of 8 neurons. We have created a 2 dimensional vector of 95 elements corresponding to a 5-vector sliding window of past records (Fig 3). Each record consists of 19 values, including the timestamp, resulting in a total of 95 tensors. We have also converted the absolute time into a relative time. After training, we have obtained the accuracy curves (Fig 6). The training accuracy is represented in blue, and the validation accuracy is shown in orange.

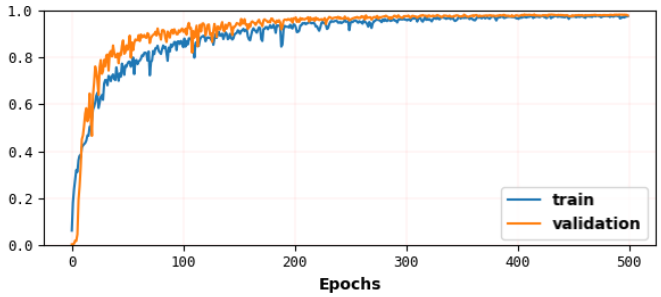

Fig. 6. Evolution of the accuracy

After training the autoencoder on data representing normal system behavior, we have assessed its capability to detect faulty components. To achieve this, we have simulated various sensor and actuator faults, leveraging Factory IO to emulate scenarios such as sensors or actuators stucking at either 0 or 1. The faults are denoted as Fi_j, with i representing 0 for sticking at 0 or 1 for sticking at 1, and j indicating the specific sensor or actuator. It's important to note that we have simulated only one fault at a time. Fig 7 shows losses distributions values for each simulated fault and the normal data. When the loss value is high, the fault is easy to detect. If we take a threshold of 0.3 for example, faults loss value below 0.3, as $F0\_c5$ or $F1\_A2$, are not detected and those above 0.3 are detected as $F1\_A4$ or as $F1\_c10$ or as $F1\_c8$. It remains for the operator to determine the threshold at which the system can be considered as faulty. We notice that some errors are more significant in terms of time loss, others in terms of component loss. To improve detection performance, we could consider the loss that best identifies the fault between time loss and component loss.

We have performed a two-dimensional projection of the compressed representations of our input data, which form the output of the encoder. Fig 8 illustrates the projection

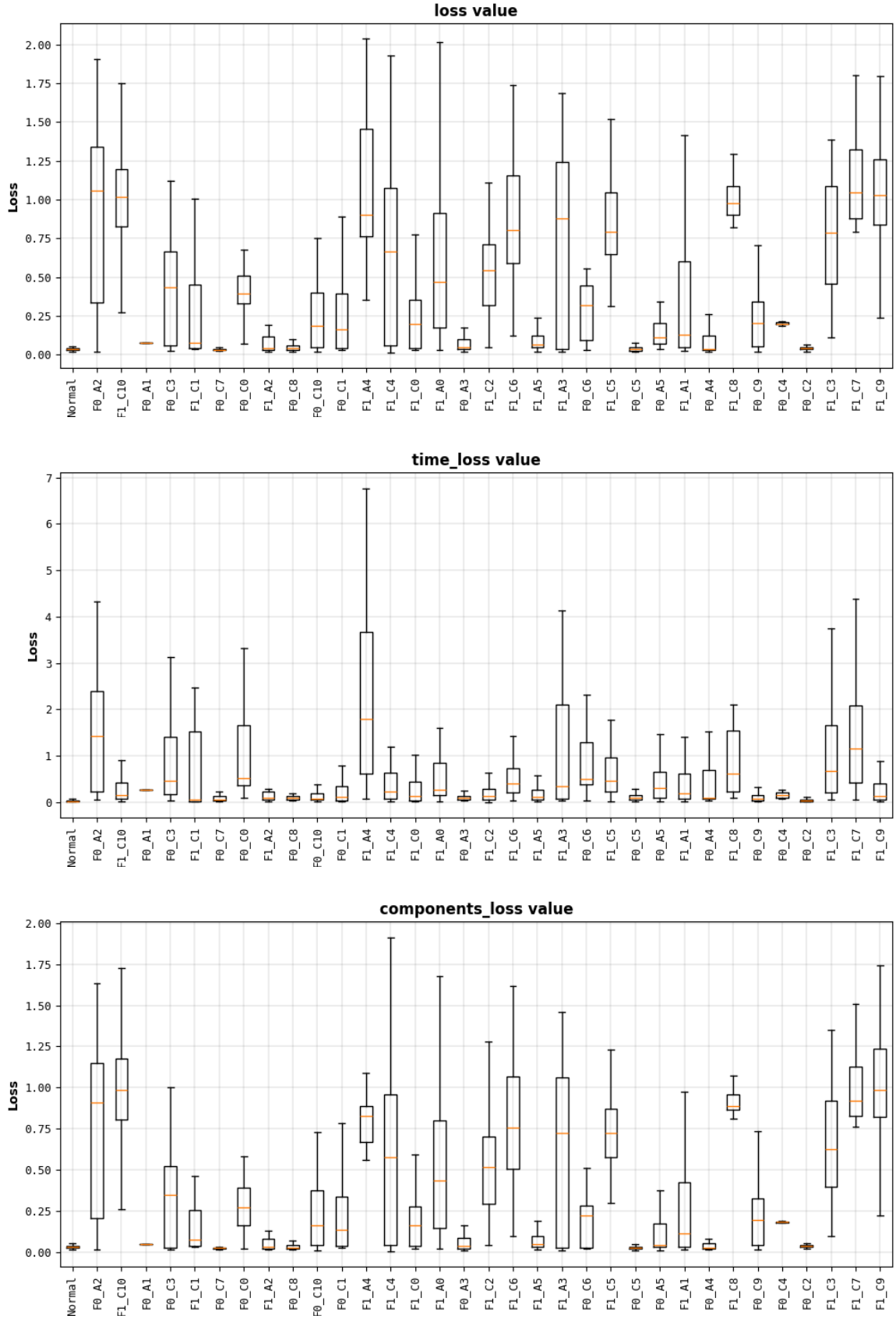

Fig. 7. Losses distributions for each fault and normal data

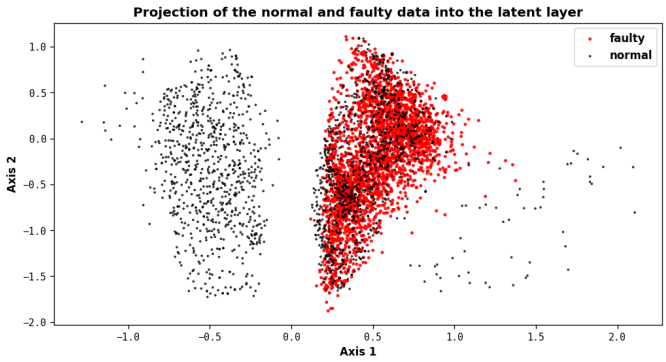

Fig. 8. Projection of normal and faulty data into the latent layer

of both normal and faulty data into the latent layer, which serves to encode crucial features extracted from the input data. The visualization clearly distinguishes between normal behavioral data (depicted in black) and faulty data (depicted in red). These extracted features represent a compression of the original features into a 2-dimensional space. However, understanding the relationships among these features necessitates the application of explicability and interpretability techniques such as SHAP and LIME. This area will be explored in the near future. In Fig 8, observed on the right, certain normal data points (represented by black dots) reside within regions predominantly occupied by faulty data. This observation suggests that the system's behavior closely resembles normal behavior, rendering it challenging to discern whether the system is normal or it is faulty. These instances represent non-detectable, non-diagnosable faults, posing significant challenges for fault detection and diagnosis.

5. CONCLUSION

In this paper, we have proposed a new data-driven unsupervised learning approach for online fault detection of DES class APSs. This approach is based on the autoencoder. The input data is first passed through an encoder, which reduces its dimensionality and produces a compressed representation, referred to as a latent space. The compressed representation is then passed through a decoder, which attempts to reconstruct the original input data from this reduced representation. The reconstruction error is then used to detect the occurrence of a fault in the plant. The results of the application of the proposed method on the sorting system of Factory IO show the significant contribution and the interest of this method to detect faults.

Several perspectives are possible. An exhaustive search to find the optimal value of the hyper-parameters (number of hidden layers, number of neurons on each hidden layer, the size of past observations...) could be performed. The choice of the threshold is crucial and may require tuning based on the characteristics of the normal behavior and fault tolerance. Cross-validation or using a separate validation set with known faults can be helpful in determining an appropriate threshold. We'd like to extend fault detection to fault diagnosis. We have applied the approach to a simulated system, an application on real systems is possible. As part of this project, an extension of the approach proposed in this paper has been expanded and applied to hybrid systems. Our objective in near future, is to provide a comprehensive understanding of the observed new behaviors in the monitored system to the human operator, presenting them with insightful "explanations" or interpretations. This shift toward intelligent diagnosis aims to enhance efficiency and efficacy while overcoming the limitations of traditional solutions.

REFERENCES

Alzalab, E.A., El-Sherbeeny, A.M., El-Meligy, M.A., and Rauf, H.T. (2021). Trust-based petri net model for fault detection and treatment in automated manufacturing systems. *IEEE Access*, 9, 157997–158009.

de Souza, R.P., Moreira, M.V., and Lesage, J.J. (2020). A hierarchical approach for discrete-event model identification incorporating expert knowledge. *IFAC-PapersOnLine*, 53(4), 275–281.

Debouk, R., Lafortune, S., and Teneketzis, D. (2000). Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete event dynamic systems*, 10(1-2), 33–86.

Ghosh, A., Wang, G.N., and Lee, J. (2020). A novel automata and neural network based fault diagnosis system for plc controlled manufacturing systems. *Computers & Industrial Engineering*, 139, 106188.

Han, T., Jiang, D., Zhao, Q., Wang, L., and Yin, K. (2018). Comparison of random forest, artificial neural networks and support vector machine for intelligent diagnosis of rotating machinery. *Transactions of the Institute of Measurement and Control*, 40(8), 2681–2693.

Riera, B. and Vigário, B. (2017). Home i/o and factory i/o: a virtual house and a virtual plant for control education. *IFAC-PapersOnLine*, 50(1), 9144–9149.

Saddem, R. and Baptiste, D. (2022). Machine learning-based approach for online fault diagnosis of discrete event system. *IFAC-PapersOnLine*, 55(28), 337–343.

Saddem, R. and Baptiste, D. (2023). Benefits of using digital twin for online fault diagnosis of a manufacturing system. In *Artificial Intelligence for Smart Manufacturing: Methods, Applications, and Challenges*, 255–269. Springer.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. (1995). Diagnosability of discrete-event systems. *IEEE Transactions on automatic control*, 40(9), 1555–1575.

Subias, A., Travé-Massuyès, L., and Le Corronc, E. (2014). Learning chronicles signing multiple scenario instances. *IFAC Proceedings Volumes*, 47(3), 10397–10402.

Venkatasubramanian, V., Rengaswamy, R., Kavuri, S.N., and Yin, K. (2003). A review of process fault detection and diagnosis: Part iii: Process history based methods. *Computers & chemical engineering*, 27(3), 327–346.

Xie, Y., Cruz, L., Heck, P., and Rellermeyer, J.S. (2021). Systematic mapping study on the machine learning lifecycle. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*, 70–73. IEEE.

Zaytoon, J. and Lafortune, S. (2013). Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2), 308–320.