mathematics

MDPI

Article

Accuracy Is Not Enough: Optimizing for a Fault Detection Delay

Matej Šprogar * and Domen Verber

Faculty of Electrical Engineering and Computer Science, University of Maribor, 2000 Maribor, Slovenia; domen.verber@um.si

* Correspondence: matej.sprogar@um.si

Abstract: This paper assesses the fault-detection capabilities of modern deep-learning models. It highlights that a naive deep-learning approach optimized for accuracy is unsuitable for learning fault-detection models from time-series data. Consequently, out-of-the-box deep-learning strategies may yield impressive accuracy results but are ill-equipped for real-world applications. The paper introduces a methodology for estimating fault-detection delays when no oracle information on fault occurrence time is available. Moreover, the paper presents a straightforward approach to implicitly achieve the objective of minimizing fault-detection delays. This approach involves using pseudo-multi-objective deep optimization with data windowing, which enables the utilization of standard deep-learning methods for fault detection and expanding their applicability. However, it does introduce an additional hyperparameter that needs careful tuning. The paper employs the Tennessee Eastman Process dataset as a case study to demonstrate its findings. The results effectively highlight the limitations of standard loss functions and emphasize the importance of incorporating fault-detection delays in evaluating and reporting performance. In our study, the pseudo-multi-objective optimization could reach a fault-detection accuracy of 95% in just a fifth of the time it takes the best naive approach to do so.

Keywords: artificial neural networks; deep learning; fault detection; accuracy; multi-objective optimization

MSC: 68T07

Citation: Šprogar, M.; Verber, D. Accuracy Is Not Enough: Optimizing for a Fault Detection Delay. *Mathematics* **2023**, *11*, 3369. https://doi.org/10.3390/math11153369

Academic Editors: Heung Soo Kim, Salman Khalid, Ananda Shankar and Prashant Kumar

Received: 29 June 2023
Revised: 25 July 2023
Accepted: 31 July 2023
Published: 1 August 2023

1. Introduction

Modern industrial systems are increasingly more complex and prone to failure, which can lead to significant dangers or high costs. Detecting faults is crucial in these systems [1], but it is challenging due to the vast amount and complex nature of data the systems handle and produce. Once a fault is detected, further analysis and decision-making processes are necessary to identify the specific fault type and prevent it from spreading.

In recent years, deep learning (machine learning using neural networks with many hidden layers) has showcased its remarkable ability to learn complex data representations, revolutionizing various learning tasks. Deep learning was also successfully applied to fault detection, which must handle a broad range of multivariate data [2,3]. Multivariate data are a sequence of chronologically recorded observations of interrelated and interacting multidimensional variables.

Although deep learning has shown success in fault detection, it is important to acknowledge that researchers using deep fault-detecting models sometimes overlook the crucial temporal aspect of fault detection. Specifically, they fail to leverage the existing temporal dependencies among variables using only a single data sample in one time step or, conversely, too much data in too many time steps [4]. Consequently, they fail to optimize for the time it takes to detect a fault. In many applications, the fault-detection delay—the time gap between the actual occurrence of a fault and its recognition by the

fault-detecting component—can be quite dangerous and therefore this delay should be as small as possible [5].

If attainable, fault prediction is better than fault detection as it allows us to prevent potentially expensive faults from occurring. We can consider fault prediction as a negative fault-detection delay. However, if predictions are unattainable, a low fault-detection delay is just as important as high sensitivity and a low false alarm rate.

Quick detection of faults while minimizing false alarms is paramount in fault-tolerant systems. A functional fault-detection system raises alarms with acceptable delays. Our objective is to demonstrate the inadequacy of approaches that ignore fault-detection delays when evaluating fault-detection (and prediction) performance. We experimented on a large and renowned synthetic dataset [6] and tried to identify the main factor influencing the accuracy and delay of the fault-detection process.

The main contributions of this paper are as follows:

(1) Emphasizing the significance of fault-detection delays when considering deep-learning fault-detection models. By disregarding the temporal aspect of the solution, standard loss functions produce solutions with little practical value.
(2) Introducing a methodology for estimating deep fault delays. In cases where the timestamps of faults are unknown, it is only possible to estimate the fault-detection delay to a certain extent.
(3) Proposing a pseudo-multi-objective approach to fault detection with any deep-learning model, although we have only validated it with Long Short-Term Memory on a single dataset. Deep models should only have access to short training sequences, or they will not learn short-term relations needed for short fault-detection delays.
(4) Providing a clear integration of machine learning concepts with fault detection. Bridging these two domains facilitates better knowledge exchange and helps prevent experimental errors.

This paper delves into the temporal aspect of deep fault detection. Additionally, we examined the influence of data windowing on fault-detection accuracy and delay. Section 2 introduces the concept of a monitoring component and provides an overview of artificial neural networks. Section 3 introduces a uniform notation for describing the context and data windows in sequential data and estimating fault-detection delays for any model. Section 4 presents a pseudo-multi-objective optimization approach that surpasses certain naive approaches observed in the literature. Finally, Section 5 illustrates the application of the proposed concepts using the widely used Tennessee Eastman Process (TEP) dataset [7].

2. Background

Fault handling commonly includes fault detection, fault identification, and fault diagnosis [8]. In general, fault detection evaluates the system's operational status to determine its normal functionality; fault identification determines the specific type of fault that has occurred; and fault diagnosis identifies the root cause of the fault and traces its potential propagation path. Sometimes it is possible to identify a fault during the fault-detection step. Regardless of the chosen approach, it is crucial to incorporate a component that monitors the entire system.

2.1. The Monitoring Component

To effectively address the system's complexity and safety concerns, we must design, evaluate, and implement the safety-related aspects independently and in parallel with the functional components. It is also essential to acknowledge the susceptibility to faults of the control process itself. An independent monitoring component (MC) is a general approach to overseeing the control function and assisting in fault handling.

The component actively monitors the inputs of the main system (a), the control system's response ($b = F(a)$), and, if available, the internal state variables (c) of the control system (refer to Figure 1). It is important to note that while the MC possesses the same information as the control function F, it does not perform the controlling task. The primary

role of the component is to confirm the validity of its inputs $X = (a, b, c)$. Consequently, the MC reports the operational diagnosis $d = f(X)$ to a higher fault management layer with the necessary expertise to handle a fault. Although a binary diagnosis indicates the presence or absence of a fault, a more intricate output may even identify a specific fault.

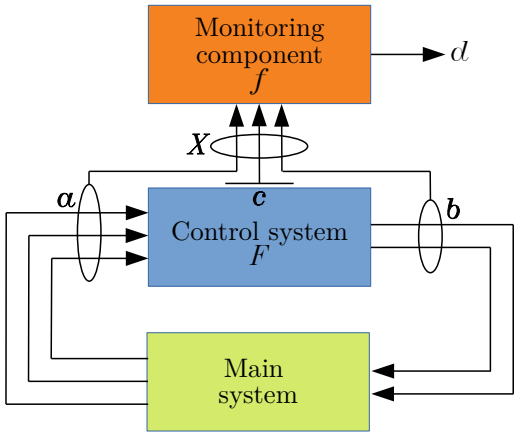

Figure 1. Data flows and the positioning of the monitoring component above the control system.

To ensure minimal interference with the control environment, the monitoring component should strive to employ physically separate and dedicated hardware exclusively for its operation. Furthermore, the monitoring component needs to operate at a speed that matches or surpasses the monitored system.

The most challenging aspect is developing a reliable decision model within the MC. This decision model is responsible for recognizing the system's state as either normal or abnormal. Deep learning with artificial neural networks has enabled many effective solutions that address this decision-making challenge [8,9].

2.2. Artificial Neural Networks

Modern machine learning encompasses a broad spectrum of approaches, among which artificial neural networks have emerged as a successful and widely adopted method. Artificial Neural Networks (ANNs) are computational models inspired by the complex network of biological neurons in animal and human brains. These networks are comprised of interconnected nodes with weighted connections that transmit signals (represented as real numbers) from one node to another. ANNs are highly suitable for addressing diverse problem domains. A general and detailed description of artificial neural networks is available in [10].

Although ANNs are fast to execute and can learn to compute nonlinear and complex functions, it is important to acknowledge the potential pitfalls associated with their use. Ref. [11] These pitfalls include:

- Data: ANNs rely on sufficient high-quality data that accurately represent the problem. Data availability and quality can be a challenge, particularly because obtaining large, labeled datasets may be impractical or costly [4].
- Learning: The success of the learning phase in ANNs is not guaranteed and can be influenced by factors such as the chosen network type and topology. Selecting the appropriate architecture and configuring the network hyperparameters can significantly impact its learning capabilities and overall performance [12].
- Verification and Overfitting: Validating the solutions produced by ANNs can be challenging. Due to the complexity of their internal workings, it can be difficult to understand how and why an ANN arrived at a specific output. Additionally, ANNs are susceptible to overfitting, resulting in reduced generalization of unseen data [11].

Addressing these issues requires careful consideration during the design and implementation of ANNs. Strategies such as data preprocessing, regularization techniques,

cross-validation, and model evaluation can help mitigate these challenges and improve the reliability and generalizability of ANNs in real-world applications.

There are two general ANN architecture types, each with distinct advantages and disadvantages. The first type is feed-forward networks, which propagate signals directly from the input to the output layer. Due to their inherent simplicity, these networks can only extract cross-dimensional correlations from large, external sequence-encompassing contexts using more complex internal mechanisms, such as convolution [13] or attention [14]. It is worth noting that feed-forward designs require substantial amounts of data and significant offline learning efforts, but they are cost-effective to use afterward.

Recurrent neural networks (RNN) can naturally capture autocorrelations by utilizing recurrent connections that loop the propagated signals. As a result, RNNs are well-suited for handling sequential data, as they can effectively leverage an internal context that they build autonomously from the supplied inputs. Chung et al. [15] observed that many effective recurrent neural networks use advanced, recurrent hidden units. Two widely used types of these hidden units are the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU). These specialized units, created for handling time-series data, have greatly simplified the development and training of recurrent networks. Recently, however, transformer-based [14] networks started to prevail also because they are faster to train [16].

One particular application of data-driven deep learning is deep anomaly detection, which involves using deep neural networks to obtain feature representations or anomaly scores. Anomaly detection focuses on identifying data instances that deviate significantly from the norm [2]. In the context of anomaly detection for multivariate time-series, Tian identifies three major challenges [4]. First, the scarcity of labeled data arising from the rarity of outlier instances. Second, identifying complex interactions and relationships among features proves to be a challenging task. Last, detecting hidden and inconspicuous anomalies in high-dimensional data presents a formidable obstacle. These challenges severely limit our ability to extract meaningful insights from the available data.

3. Deep Fault-Detection Models for Time-Series Data

Data-based fault detection using machine learning can go two ways. The first approach involves constructing a classifier using supervised learning techniques. This classifier is trained on labeled data and learns to classify a given data sample as either normal operating conditions (NOC) or a potential fault. Predicting faults in the (near) future is even more advantageous than simply detecting existing faults, and classification models can be trained accordingly. However, supervised learning relies on high-quality labeled data representing all possible system states. Obtaining such data in large quantities can be challenging, especially in many industrial environments where faults are infrequent and come at a high cost. Nevertheless, supervised deep-learning approaches for fault detection have been successfully applied in various areas, including chemical production systems [17], semiconductor manufacturing processes [18], and high-performance computing [19].

On the other hand, unsupervised learning provides a more suitable alternative for fault detection, particularly in scenarios with few or no faulty samples available for training. In this approach, autoencoders (AE) are widely used as a technique for deep fault detection [20,21]. Autoencoders are neural network architectures that encode the input signal into a latent representation and then attempt to reconstruct the original signal from this compressed information. A level of successful reconstruction of the input signal is a measure of similarity between the test signal and the pre-learned normal signals.

By training autoencoders exclusively with normal operating condition data, they learn the underlying structure or principal components of the signals that describe the normal operation of a plant. When presented with an anomalous signal, their ability to accurately recreate the input will be lower than with non-anomalous signals used in training. This difference in reconstruction performance enables us to distinguish anomalous samples from the NOC samples based on the reconstruction error alone.

Consequently, the performance of autoencoders heavily relies on selecting an appropriate error threshold, which leads researchers to employ Receiver Operating Characteristic (ROC) analysis by varying these thresholds. Unlike classifiers that can only distinguish between normal operating conditions and expected faults, autoencoders can detect even unanticipated states of operation.

3.1. Performance Metrics

Machine learning (ML) strongly emphasizes accuracy, primarily due to the generalization challenges that artificial neural networks face when confronted with unseen data. The main concern in ML is to mitigate incorrect outputs caused by the unstable propagation of features through ANNs, rendering them ineffective in the presence of minor input perturbations [22,23].

Several studies have exclusively concentrated on established machine learning evaluation metrics, such as accuracy, while neglecting the inherent sequential nature of the data and failing to consider fault-detection delays [24–26].

3.2. The Data

The time-series data are a sequence of data samples denoted by $S = [X^{(1)}, \ldots, X^{(T)}]$, where each sample $X^{(t)}$ represents the input signals at time t, and T is the total number of time ticks. Specifically, the sample $X^{(t)} = [x_1^{(t)}; \ldots; x_N^{(t)}], X^{(t)} \in \mathbb{R}^N$, represents the signal values of the N features at time tick t. The fault-detection model takes an input sample and generates the output $y = f(X), y \in \mathbb{R}$. Each input sample corresponds to one fault-detection case. Finally, a dataset, which consists of d sequences, can be denoted as $D = [S_1; S_2; \ldots; S_d]$.

Let us define a sliding window function that selects the last w samples from a sequence S at time t:

$$W_w^{(t)}(S) := [X^{(t-w+1)}, \ldots, X^{(t-1)}; X^{(t)}], \quad w \leq t \leq T. \tag{1}$$

This allows us to denote $S_w^{(t)} = W_w^{(t)}(S)$. In a sequence consisting of T samples, the window $W_T(S)^{(T)}$ selects the entire sequence: $S = W_T(S)^{(T)}$, and $W_1(S)^{(T)}$ selects only the last sample: $X^{(T)} = W_1^{(T)}(S)$. The sliding window utilized for sample selection is similar to, but should not be confused with, the fault-absorbing sliding windows [27].

3.3. The Context

All deep-learning models rely on a context that serves as the reference frame for the inputs they process. Samples preceding the most recent sample $X^{(T)}$ form the simplest context. Specifically, the context includes the w samples visible through the window $W_w^{(T-1)}$.

In the case of feed-forward (FF) models, the context is external and must be provided at each time step. One way to incorporate it is by introducing it as a second parameter to the fault-detection function, such as $y = f(S_1^{(T)}, S_w^{(T-1)})$. However, for simpler implementation, machine learning flattens the context and appends it to the current sample, resulting in $y = f(S_{w+1}^{(T)})$, or even more simply $y = f(S_w^{(T)})$. It is important to note that flattening the external context removes the temporal axis from the data.

In contrast, recurrent models utilize a hidden state H, eliminating the need for an external context. The operation of recurrent models can be represented as $(y^{(T)}, H^{(T)}) = f(S_1^{(T)}, H^{(T-1)})$. As H is an internal state, we can omit it from the notation, resulting in $f(S_1^{(T)}) = f(S_1^{(T)}, H^{(T-1)})$. Although recurrent models process time-series data and update their state by considering one sample $X^{(T)}$ at a time, they still require the processing of several consecutive samples to detect a fault because the state $H^{(t)}$ depends on $H^{(t-1)}$. Considering that all the previous states influence the internal state, we can express the entire history of processing as $y = y^{(T)} = f(S_1^{(1)}) \circ \ldots \circ f(S_1^{(T)})$. Random initialization of $H^{(0)}$ makes $H^{(T-w)}$ a possible starting point for sequentially processing

the sequence $S_w^{(T)}$, leading (again) to $y \approx f(S_w^{(T)})$. Unlike feed-forward networks, which require flattening consecutive samples into one wide input, recurrent models retain the temporal organization of the data.

The 'context data' size, determined by the hyperparameter w, limits the model's ability to capture long relationships. Feed-forward models have a limitation in that they can only capture correlations within the constrained context ($w << T$) and are unable to detect correlations that extend beyond it. It is important to note that the implementation of the model may impose restrictions on the context size. The context size limitation is observed in popular models like ChatGPT as well [28].

3.4. Fault-Detection Delay

Artificial neural networks are extremely quick at generating output. Unless a highly responsive system is being monitored, the time required for this transformation can be ignored, if we compare it to the duration of a single time step.

To quantify the fault-detection delay δ, which refers to the time elapsed between the actual onset of a fault and its detection [5], it is necessary to possess accurate information regarding the fault's exact occurrence time τ within the system. Unless an external source or an oracle provides this temporal data, determining the exact fault occurrence time becomes extremely challenging. Faults typically require a certain amount of time to be propagated through the system and are seldom detectable immediately upon initiation. Furthermore, there are instances when the process control system actively masks the fault by compensating for its adverse effects before the system enters a visibly anomalous state that can be detected.

In machine learning, there is a common assumption that the associations within a system can be acquired by recording its features and that the comprehensive data captures all relevant relationships. Consequently, utilizing *all* available data to make informed decisions through the execution of $f(S_T^{(T)})$ establishes the performance baseline for fault detection, with the fault-detection delay $\delta = T - \tau$. It is important to note that the baseline model exhibits the maximum delay when applied to pre-recorded data. The true fault-detection delay occurs only when the model is employed live on a data stream.

To determine the model's fault-detection delay when the fault time is unknown, we must identify the minimum number of samples required to provide sufficient information for accurate enough fault detection at any time step:

$$f(S_\delta^{(t)}) = f(S_{\delta+i}^{(t)}), \quad \forall (t,i) \in [\delta, T] \times [0, T-\delta] \tag{2}$$

Unfortunately, the brute force approach to determine δ using Equation (2) is very slow, because we must determine δ for all possible training inputs:

$$\delta = \max_K \delta(D), \tag{3}$$

where K is the set of all possible inputs of size w obtainable from the T time steps of long sequences from D. Using the sampling stride of 1 gives $|K| = |D| \cdot (T - w + 1)$ possible inputs. However, if one has knowledge of τ, one can determine δ much simpler, because the model requires δ time steps to detect a fault:

$$\begin{array}{ll} f(S_\delta^{(\tau+i)}) & = NOC \quad \forall i \in [0, \delta[\\ f(S_\delta^{(\tau+\delta)}) & \neq NOC \end{array} \tag{4}$$

If we do not have τ, we should try to put a lower bound on δ by measuring the delay at time step T. This is carried out by finding the smallest input that still produces the same output as the baseline decision $f(S_T^{(T)})$; we shall denote this statistic as λ:

$$\begin{array}{ll} f(S_\lambda^{(T)}) & = f(S_{\lambda+i}^{(T)}), \quad \forall i \in [0, T-\lambda] \\ & \neq f(S_{\lambda-1}^{(T)}). \end{array} \tag{5}$$

High λ suggests that the model is unstable, while a lower λ value is a sign of a simple problem. λ is not $\delta^{(T)}$—the fault-detection delay at time T—but a lower bound (If the model performs fault identification, the lower bound corresponds to the highest λ of all faults) for fault-detection delay at time T. Although the delays for other inputs are probably different, the worst-case fault-detection delay will be at least λ:

$$\lambda \leq \delta. \tag{6}$$

Because no (deep) model is perfect, the decision regarding a fault can change with additional input(s). Raising premature alarms is bad but can only be mitigated by waiting for more data that would confirm the alarm but delay the decision. We need a measure of how fast the model's output stabilizes. A simple boundary can be determined by testing the model using inputs $S_t^{(t)}, t \in [1, T]$, which include the first t samples. Let us denote with μ the time step when the output becomes *stable*–subsequent samples do not change the outcome of the fault-detection process:

$$\begin{aligned} f(S_\mu^{(\mu)}) &= f(S_{\mu+i}^{(\mu+i)}), \quad \forall i \in [0, T-\mu] \\ &\neq f(S_{\mu-1}^{(\mu-1)}). \end{aligned} \tag{7}$$

In contrast to λ, μ retains older samples and discards more recent ones from the input. When the precise time of fault initiation τ is known (e.g., provided by an oracle), the fault-detection delay can be calculated by straightforward subtraction:

$$\delta = \mu - \tau. \tag{8}$$

The delay estimation strategy applies when the fault occurrence time is unknown (Figure 2a). After determining the λ and μ statistics, we need $\lambda \leq \mu$ to interpret the results. Because λ and μ represent the delay's lower and upper bound ($\lambda \leq \delta \leq \mu$), a situation with $\lambda > \mu$ indicates problems with fault detection in the underlying model. These problems could be due to overfitting [29], instability [23], generalization problems [30], or other issues we must address. When we know the fault time τ, we can measure the detection time and directly compute the delay (Figure 2b).

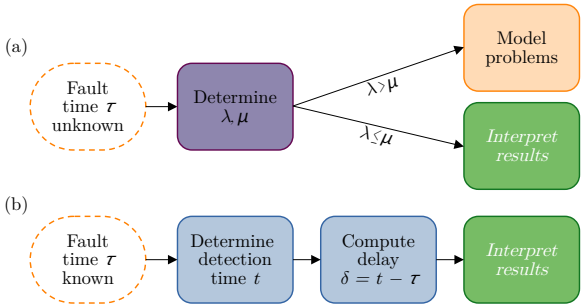

Figure 2. The approach to fault-detection delay analysis depends on the availability of the fault occurrence time. (**a**) Strategy if fault time is unknown; (**b**) procedure when the fault time is known.

4. Pseudo-Multi-Objective Optimization

In multi-objective optimization, a single model cannot simultaneously achieve the best performance in all dimensions [31]. Optimizing for accuracy and low delay in fault detection involves a trade-off, as these objectives are inherently conflicting. Consequently, exploring multiple Pareto optimal solutions that offer a balanced trade-off between accuracy and delay becomes crucial.

Various universal techniques for tackling multi-objective optimization exist. One common approach involves using weighting methods, such as adaptive weighting techniques proposed by Xie et al. [32], or using multi-objective instance weights as discussed by

Lee et al. [33]. By incorporating such techniques, solutions that balance competing objectives form the Pareto front of candidate solutions to the optimization problem. Figure 3 positions various models according to their accuracy and fault-detection delay performance. If the trivial model solution, which never signals a fault, has no fault-detection delay, the ideal model would detect all faults instantly. Learning increases the models' accuracy because explicit ML loss optimization pushes models horizontally toward higher accuracy. Only an orthogonal incentive (explicit or implicit) would push the models towards short fault-detection delays.

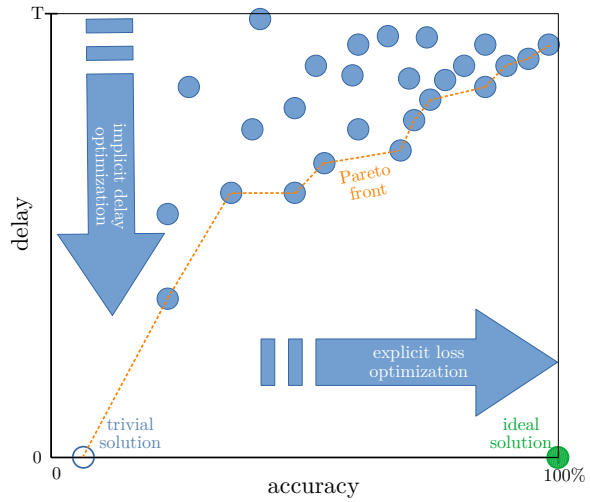

Figure 3. Pareto front connects non-dominating models on the accuracy and fault-detection delay plane.

In this context, we adopt the implicit approach. We prioritize utilizing established deep optimization techniques to maximize fault-detection accuracy while aiming for a short delay as an implicit objective. This implicit optimization for short delays aligns harmoniously with the accuracy-optimizing objectives of existing ML libraries without necessitating any modifications to the existing ML code.

When training a fault-detection model with recordings of historical data that stretch over many time steps, the conventional ML approach poses the question, "Do these historical data contain any faults?" However, in real-time fault detection, it is important to rephrase the question: "Do these historical data indicate an imminent or a recently occurred fault?" This shift in emphasis redirects the focus from analyzing the distant past to the near future or present.

When training the model with historical data with the fault introduced relatively early compared to the overall length of the sequence ($\tau << T$), we effectively ask the first question. To ask the second question, we would ideally need training data with the fault occurring in the last time step ($\tau = T$) or, even better, in the future ($\tau > T$). The acquisition of such data, especially in sufficient quantities for deep learning, can be very challenging, however.

However, when working with a large volume of historical data, it is important to exercise caution in its utilization. Instead of treating a single sequence as a single learning case, where the input $S = S_T^{(T)}$ includes all T samples, one can reorganize data into multiple smaller inputs [34]. Figure 4 shows a sliding window that samples at every time step and produces $T - w + 1$ distinct yet overlapping training cases $S_w^{(t)}$ of size w, where $t \in [w, T]$. We denote models trained on inputs of size w as M_w.

Strictly speaking, utilizing shorter inputs only partially falls under the umbrella of multi-objective optimization. Nevertheless, training with shorter inputs implicitly encourages machine learning algorithms to uncover short correlations for rapid fault detection.

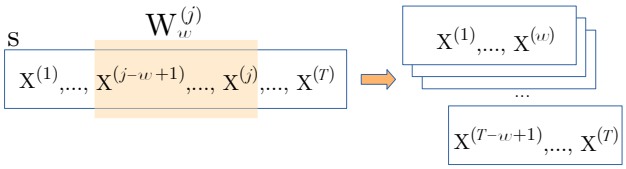

Figure 4. Sliding a window of size w over a long sequence with T samples produces $T - w + 1$ sequences.

5. Case Study

The data used in our study are obtained from the Tennessee Eastman Process, originally introduced by Downs and Vogel [7] and extensively described by Chiang [35]. TEP is a widely recognized benchmark for researching process monitoring and control. It replicates natural processes by incorporating modified components, kinetics, and operating conditions. TEP is a synthetic dataset where all dynamic behavior arises from software-based simulations. Since its inception, the simulation code has undergone several enhancements, solidifying TEP as one of the most frequently employed benchmarks for studying highly nonlinear and strongly coupled data.

Our decision to utilize the Tennessee Eastman Process dataset in our study is based on several factors. First, TEP has been widely adopted by many researchers, as evident from the works of Heo et al. [36,37], Sun et al. [38], and Park et al. [39]. Second, the published papers we reviewed did not adequately address or fulfill the specific objectives of our research. Lastly, acquiring high-quality datasets is often a challenging endeavor. To ensure repeatability and facilitate transparency, we opted to employ an extensive recording of TEP simulation data provided by Dataverse [40]. Notably, this dataset is also employed in the MATLAB Help Center as an illustrative example for demonstrating the application of deep learning with time series and sequences using the Deep-Learning Toolbox [24].

The TEP dataset represents a chemical plant, where the overarching control strategy, as outlined in Downs and Vogel [7], aims to optimize overall performance. The plant's control system diligently monitors and logs 52 distinct features, comprised of 41 sensor measurements and 11 manipulated variables at 3-minute intervals. Within the TEP environment, the plant can operate under normal operating conditions, denoted as fault 0, or encounter any of the 20 preprogrammed faults (faults 1–20). Upon the occurrence of a fault, the control system attempts to mitigate the disturbance, by either successfully restoring the system to the NOC state or allowing the fault to escalate beyond the NOC boundaries.

The dataset includes a substantial amount of training and test data. Specifically, the training phase consists of 500 simulated plant runs for each combination of normal operating conditions and 20 fault scenarios, resulting in a total of 10,500 simulation runs. Each run spans 25 h, providing 500 samples per training sequence. In the case of a faulty run, the fault is intentionally triggered at $\tau = 20$ time steps into the normal plant operation.

The test data follow a similar structure, with 500 independent simulations conducted for each NOC/fault scenario. However, these simulations have a longer duration, consisting of 960 samples, and faults are introduced at a later stage, precisely after $\tau = 160$ time steps of normal operation.

The training data were divided into 400 training samples and 100 validation samples for effective model development and evaluation. Each simulation run was an individual classification case throughout all study phases, including training, validation, and testing.

The Tennessee Eastman Process has been the subject of extensive study by numerous researchers. Several authors excluded faults 3, 9, and 15 from their research [24,39]. The rationale behind this decision stems from the observation that the plant's control system can effectively handle the disturbances caused by these faults. Consequently, distinguishing these faults from the plant's normal operation proves to be a challenging task. We also removed faults 3, 9, and 15 from our dataset in light of these findings. Consequently, the training and test dataset consisted of 18 distinct plant states, resulting in 9000 recorded sequences for each dataset. Since the TEP data includes information about the faults and

their occurrence times, the task goes beyond simple fault detection and involves fault identification through classification.

5.1. Setup

In developing and training our deep fault-detection models, we strictly adhered to the setup and procedure outlined in the tutorial [24], except that we employed Python in conjunction with the Keras/TensorFlow framework. The code snippet below illustrates the model creation process, faithfully reflecting the prescribed methodology. Each model had three layers of LSTM cells and 43,788 trainable parameters.

$$
\begin{aligned}
&model = tf.keras.Sequential([\\
&\quad LSTM(units = 52, return\_sequences = True),\\
&\quad Dropout(rate = 0.2),\\
&\quad LSTM(units = 40, return\_sequences = True),\\
&\quad Dropout(rate = 0.2),\\
&\quad LSTM(units = 25, return\_sequences = False),\\
&\quad Dropout(rate = 0.2),\\
&\quad Dense(units = 18, activation = tf.nn.softmax),\\
&])
\end{aligned}
$$

During the training phase, the simulation records were processed in batches of 32. Like in tutorial [24], we employed the mean squared error loss function and the Adam optimizer to optimize the model's parameters.

5.2. Fault-Detection Performance

The results presented in Table 1 are the average values obtained from 30 trained networks. At first glance, the selected model exhibited remarkable learning capabilities on the TEP data, yielding outstanding accuracy. This was confirmed by Matthew's Correlation Coefficient (MCC) score, which is known to be more effective in describing performance on multiclass and unbalanced datasets compared to traditional metrics like the F1 score [41].

The limitation of training the model within 30 epochs was sufficient in discovering a solution that appears to be close to the global optimum. The model achieved a validation loss of 2×10^{-5} and a training loss of 0.00027. The test results for this model are summarized in Table 2. We will designate this specific model as Model #1, representing the baseline M_{500} family, where all members are trained on cases of 500 samples.

Although the MATLAB tutorial [24] focuses exclusively on classification accuracy by employing all samples per sequence for a single fault classification, it ignores the fault-detection delay. The important question to answer is when deep models begin detecting faults.

Table 1. Training, validation, and test results.

Metric	Min	Max	Avg	Median	StDev
Training: 7200 simulations, 500 time steps each					
MCC:	0.7248	0.9999	0.9446	0.9569	0.0581
Acc:	0.7324	0.9999	0.9464	0.9590	0.0565
Validation: 1800 simulations, 500 time steps each					
MCC:	0.7257	1.0000	0.9416	0.9575	0.0575
Acc:	0.7333	1.0000	0.9435	0.9597	0.0559
Test: 9000 simulations, 960 time steps each					
MCC:	0.7214	0.9979	0.9395	0.9503	0.0594
Acc:	0.7289	0.9980	0.9415	0.9529	0.0578

Table 2. The confusion matrix of the reference Model #1 on the test data; 99.733% accuracy, MCC 0.99718, asterisk denotes NOC and fault categories 1, 2, 4, 5, 7, 8, 11, 14, 17, 19, and 20.

		ACTUAL						
	Fault	**6**	**10**	**12**	**13**	**16**	**18**	*
PREDICTED	**6**	498	0	0	0	0	0	0
	10	0	500	0	0	1	0	0
	12	0	0	496	0	0	0	0
	13	2	0	0	483	0	0	0
	16	0	0	0	0	499	0	0
	18	0	0	4	17	0	500	0
	*	0	0	0	0	0	0	6000

5.3. Fault-Detection Delay Analysis

During the training and validation phase, faults were introduced in the TEP dataset at time step $\tau = 20$. However, detecting these faults was delayed until step 500, when the last training record became accessible. We conducted the λ and μ analyses on the training, validation, and test data to investigate the behavior of the taught Model #1.

To demonstrate the λ score, Figure 5 depicts the performance of the reference classifier on a single test sequence labeled as 'fault 1'. The sequence was divided into 960 subsequences, denoted as $S^{(960)}_{960-t}, t \in [1, 960]$, and subsequently classified using the reference Model #1. Inputs with $\lambda = 16$ or more samples accurately supported the baseline 'fault 1' classification.

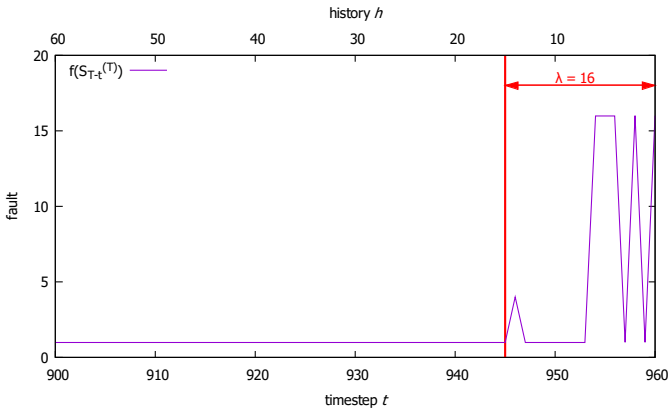

Figure 5. Fault detection using $S^{(T)}_{T-t}$ with windows of size $h = T - t$; fault 1 test case, $\lambda = 16$.

Table 3 presents the λ scores of the reference model for different faults on the validation data, which was otherwise perfectly classified. Surprisingly, the maximum λ score of 500 was observed for 'fault 12', indicating that the network required at least 500 steps to correctly classify a sample belonging to the 'fault 12' category. This finding is unexpected because the training/validation data encompass 20 steps of NOC data at the beginning of every 500-step sample. These initial 20 steps should not have influenced fault categorization, meaning that the network is overfitting the training data.

For completeness, Table 4 displays the λ values for the reference model on the test data, including a few incorrect baseline predictions.

Figure 6 shows our reference classifier's μ performance on the 'fault 1' test sample. Slices with less than 425 records were insufficient to recognize 'fault 1'; however, if at least $\mu = 425$ or more time steps were available, the network could predict 'fault 1' correctly.

Table 3. Reference classifier's λ results on the validation data with 100 sequences per fault.

Fault	#	Min	Max	Avg	Median	StDev
NOC	100	11	392	135.0	127.0	83.3
1	100	1	25	3.0	2.0	5.3
2	100	1	1	1.0	1.0	0.0
4	100	1	23	2.0	2.0	3.1
5	100	1	73	8.0	4.0	11.7
6	100	1	467	102.0	3.0	148.2
7	100	2	17	2.0	2.5	1.8
8	100	1	279	59.0	42.0	63.0
10	100	33	471	190.0	148.5	123.0
11	100	12	307	84.0	75.5	47.8
12	100	178	500	406.0	412.0	51.9
13	100	1	291	38.0	26.0	46.2
14	100	2	35	4.0	3.0	4.5
16	100	41	471	210.0	195.5	107.8
17	100	26	453	131.0	83.5	109.6
18	100	1	275	58.0	2.0	77.6
19	100	12	385	116.0	94.5	84.4
20	100	14	140	53.0	49.5	26.2

Table 4. Reference classifier's λ results on the test data with 500 samples per fault.

Fault	#	Min	Max	Avg	Median	StDev
NOC	500	11	570	157.0	137.0	97.7
1	500	1	53	3.0	2.0	6.8
2	500	1	1	1.0	1.0	0.0
4	500	1	89	3.0	2.0	8.0
5	500	1	131	9.0	4.0	14.4
6	498	1	835	246.0	3.5	298.8
7	500	2	19	2.0	2.0	1.3
8	500	1	367	65.0	47.0	67.0
10	501	7	783	194.0	140.0	152.0
11	500	2	261	91.0	83.5	52.0
12	496	88	857	724.0	749.0	93.8
13	485	1	476	46.0	24.0	64.7
14	500	2	41	4.0	3.0	4.2
16	499	16	810	228.0	178.0	161.2
17	500	17	750	141.0	90.0	140.1
18	521	1	650	51.0	2.0	128.1
19	500	5	663	159.0	124.5	127.0
20	500	2	237	55.0	50.0	31.2

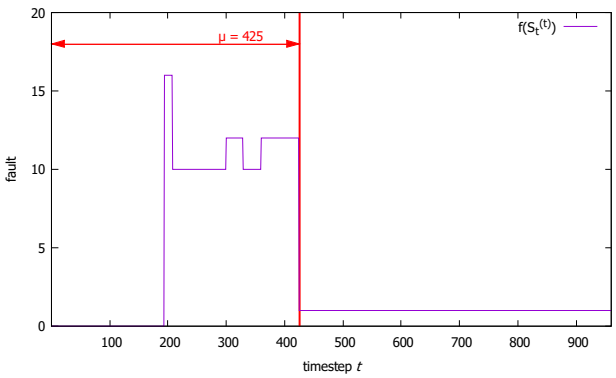

Figure 6. Reference classifier's decisions for inputs $S_t^{(t)}$; fault 1 test case, $\mu = 425$.

Table 5 presents the μ results obtained from the test data. It is important to note that the μ metric does not assess classification accuracy but focuses on its consistency over time. Similar to the observations with λ, we can again identify unexpected behavior. Despite training the network on 500 time step sequences, the average stability of classification can only be achieved well beyond 500 steps, as indicated by the value of $Avg(Max(\mu)) = 636.7$. This counterintuitive behavior further suggests that the network is not aligning with our initial expectations.

Given that we are aware of the fault introduction time of $\tau = 160$ in the TEP test data, we can calculate the average fault-detection delay for each fault by subtracting 160 from the highest μ score achieved by the classifier, as shown in Equation (8). On average, our best classifier would require $Avg(Max(\mu) - 160) = 476.7$ time steps to detect a fault. It is worth noting that only the NOC data slices were detectable before the 160th time step, with an average detection occurring after just 15 samples. One NOC sample, however, required at least 939 records to be correctly classified.

Table 5. Reference classifier's μ results on the test set.

Fault	#	Min	Max	Avg	Median	StDev
NOC	500	1	939	15.0	6.0	67.6
1	500	337	535	411.0	409.0	29.9
2	500	262	342	303.0	304.0	7.2
4	500	238	335	284.0	284.0	17.2
5	500	356	511	411.0	412.0	16.0
6	498	213	437	250.0	236.0	36.7
7	500	359	542	437.0	442.0	30.7
8	500	200	944	291.0	278.5	76.7
10	501	206	942	305.0	284.0	80.3
11	500	244	579	295.0	288.0	35.9
12	496	311	916	659.0	651.0	106.9
13	485	218	900	279.0	268.0	54.2
14	500	204	308	225.0	225.0	8.7
16	499	203	954	312.0	284.0	96.3
17	500	217	416	273.0	269.5	28.2
18	521	315	924	437.0	424.0	82.6
19	500	221	509	272.0	266.0	34.3
20	500	258	427	342.0	340.5	31.2

5.4. Comparison with Other Studies

There is a need for more directly comparable studies on fault-detection delay. For instance, the study by Heo et al. [37] mentions that linear PCA and p-NLPCA detected fault 5 as early as at time step 162, which corresponds to only two samples (equivalent to 6 min) after its introduction into the system. In contrast, our reference Model #1 needed 196 additional samples to detect the fault correctly.

Providing more insightful results are the findings from Park et al. [39], where a combined autoencoder and LSTM network was employed. According to their report, the average detection delay for faults 01, 02, 05, 06, 07, 12, and 14 was less than 30 min, showcasing superior performance compared to our baseline. However, it is worth noting that their model exhibited lower accuracy at 91.9%, which likely accounts for the disparities in the achieved detection delays. Our network, on the other hand, required more information to ensure higher classification accuracy. The question is, would shorter training cases make a difference?

5.5. Using Smaller Windows

The baseline Model #1 exhibited a notable fault-detection delay, highlighting the inadequacy of training on long sequences with faults embedded early in the process. Conversely, training on an individual sample per case fails to capture autocorrelations.

Given that our objective is not to find the optimal model but to illustrate the impact of shorter training cases, we chose to utilize the W_5 window to create models in the M_5 category. In the TEP dataset, a training case spanning 5 time steps represents 15 min of plant operation.

Generating training cases from a single sequence allowed us to augment the training dataset to include 3,571,200 cases. Similarly, the validation and test sets increased to 892,800 and 8,604,000 cases, respectively, resulting in a dataset with over 13 million fault-describing sequences. To create a representative Model #2 for the M_5 category of models, we followed the same procedure as when generating the baseline models of the M_{500} family. The final representative Model #2 models achieved a training loss of 0.00294 and a validation loss of 0.00293. With an accuracy of 97.88% and an MCC score of 0.8955, the test performance of Model #2 was lower than that of the reference Model #1.

To facilitate a more thorough comparison between the two models, we must assess their usability for fault detection. This evaluation should consider accuracy and delay based on the consecutive input samples that describe the operation of the plant.

In Figure 7, we can observe the variations in accuracy over time for the models. Figure 7a illustrates the models' performance on 500 steps of the training and validation data, while Figure 7b depicts the corresponding analysis for the test data, encompassing 960 steps. The accuracy at each time step is calculated based on 7200, 1800, and 9000 sample recordings from the training, validation, and test datasets, respectively.

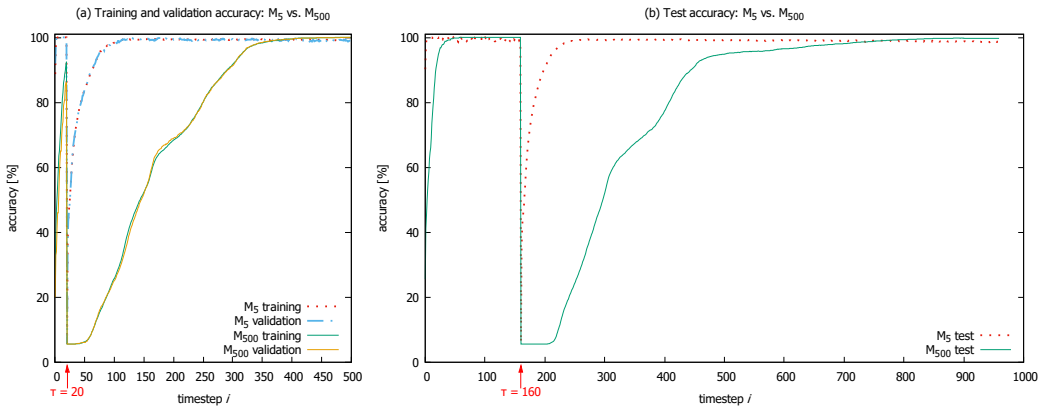

Figure 7. Running overall accuracy scores for the models #1 and #2 representing the respective M_{500} and M_5 category at each time step. (**a**) Results on training and validation data; (**b**) test data results.

The training and validation datasets were structured to include an initial period of 20 steps representing a faultless operation. For the baseline Model #1, the accuracy in identifying normal operating conditions started at 20% for the first time step and improved to 86% after 20 time steps. In contrast, Model #2 achieved 92% accuracy at the first time step and reached 100% accuracy after three samples. On the test data, it took the baseline Model #1 58 time steps to achieve a perfect (100%) accuracy score, while the accuracy of Model #2 had already started to decline by then.

The introduction of faults (indicated by arrows in Figure 7) resulted in a significant decrease in the accuracy of both models. Model #1 kept classifying all inputs as NOC for a while, whereas Model #2 started to improve immediately. On the validation data, it re-reached the 95% accuracy level with a delay of 55 time steps, whereas the baseline model's delay was 298 steps to reach the same level of performance.

5.6. Discussion

The subpar performance of the baseline model during the initial time steps can be caused by the inability of the S_{500} window to specify the first $X^{(i<\tau)}$ samples as belonging to the NOC category when the whole sequence was faulty. Per design, all samples, including the normal ones, were categorized as faulty. This combination of data could have caused the model to mistakenly associate the NOC state with a faulty condition, resulting in inferior start-up performance.

In all three datasets, there was a point where the baseline model started outperforming the M_5 representing model, reaching its peak accuracy at the final time step. This outcome aligns with the primary objective of machine learning, which aims to achieve high overall accuracy based on all available data. However, these graphs clearly illustrate that the standard ML loss criterion is inadequate for effective fault detection. If a model is not penalized for delays, it will prioritize marginal accuracy increases over significant delay improvements.

Recognizing faults becomes progressively easier the longer they persist in the system. However, training machine learning models to identify long-standing faults tends to result in slower detection. To address the detrimental effects of long-standing faults, we can employ data windowing and restrict the model's access to information during training and operation. Windowing creates multiple smaller fault-detection cases with reduced information, which may not capture long autocorrelations but encourage the model to focus on learning and leveraging short-term correlations. The "less means more" principle holds, as less data can lead to shorter fault-detection delays.

All models in this study utilize LSTMs with recurrent architecture, allowing them to generate outputs starting from the first sample. However, it is important to note that their internal contexts still require a warm-up period. Interestingly, the warm-up periods for both models differ significantly. The baseline model exhibits a much slower warm-up process, requiring a larger number of samples to achieve comparable levels of accuracy.

A pseudo-multi-objective optimization can improve delay and accuracy simultaneously. It uses the data window size as a hyperparameter that significantly influences the temporal behavior of the model. Tweaking other deep-learning hyperparameters, such as the number of training epochs or the size of the neural network, can improve accuracy [36] but can also negatively affect the delay. However, as our objective did not involve finding the optimal TEP fault-detection model, we refrained from conducting an extensive analysis of various window settings.

6. Conclusions

Although a control system can handle various disturbances, developing a dedicated monitoring system specifically designed for fault detection and identification is crucial. When applying deep-learning approaches to solve fault-detection problems, it is important to consider an additional objective other than accuracy. Standard metrics are inadequate and, as a result, misleading. They primarily focus on the correctness of results but neglect the importance of delays in fault detection. Furthermore, it is unreasonable to expect machine learning toolkits to excel in all domains universally; their effectiveness can vary significantly.

The monitoring component must recognize a fault from data samples collected during the fault-detection delay. The MC receives less information in the short period of time after a recently occurred fault than in the longer period of time after an old fault. Moreover, recent faults manifest less detectable anomalous traits than the older ones. Consequently, long delays support better detection of older faults. Fault detection is inherently a bicriteria optimization problem, where the fault-detection delay objective conflicts with the fault-detection accuracy objective.

The comparison of two deep neural network models, which were trained differently in some and identically in other aspects, highlights the need to understand the fundamental limitations of the machine learning approach for optimization. In this context, we described

a simple alternative to a more complex multi-objective methodology that would have been required otherwise. Use of shorter training cases implicitly encourages deep-learning models to detect and leverage shorter correlations. Additionally, this approach aligns well with readily available machine learning frameworks.

Although we were able to replicate the high-accuracy results on the TEP dataset reported elsewhere, it is important to acknowledge that the baseline solution is not suitable for real-life applications. This becomes evident when observing the accuracy scores over time. The case study illustrates the flaw of the baseline deep fault-detection concept. We can produce better models only by circumventing the issue or applying multi-objective optimization. Following the No Free Lunch theorem [42], however, it is important to recognize that there is no universally best approach. We aim to highlight why fault detection and identification warrant special attention.

Author Contributions: Conceptualization, M.Š.; methodology, M.Š.; software, M.Š.; validation, M.Š. and D.V.; formal analysis, M.Š.; investigation, M.Š. and D.V.; resources, M.Š. and D.V.; data curation, M.Š.; writing—original draft preparation, M.Š.; writing—review and editing, M.Š. and D.V.; visualization, M.Š.; supervision, D.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Slovenian Research Agency (research core funding No. P2-0057).

Data Availability Statement: The data presented in this study are openly available in Dataverse at https://doi.org/10.7910/DVN/6C3JR1, reference number [40].

Acknowledgments: The authors acknowledge the financial support from the Slovenian Research Agency.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AE	Autoencoder
ANN	Artificial Neural Network
FF	Feed-Forward
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MC	Monitoring Component
MCC	Matthew's Correlation Coefficient
ML	Machine Learning
NOC	Normal Operating Conditions
ROC	Receiver Operating Characteristic
RNN	Recurrent Neural Network
TEP	Tennessee Eastman Process

References

1. Abid, A.; Khan, M.T.; Iqbal, J. A Review on Fault Detection and Diagnosis Techniques: Basics and Beyond. *Artif. Intell. Rev.* **2021**, *54*, 3639–3664. [CrossRef]
2. Pang, G.; Shen, C.; Cao, L.; Hengel, A.V.D. Deep learning for anomaly detection: A review. *Acm Comput. Surv. (CSUR)* **2021**, *54*, 1–38. [CrossRef]
3. Qiu, S.; Cui, X.; Ping, Z.; Shan, N.; Li, Z.; Bao, X.; Xu, X. Deep Learning Techniques in Intelligent Fault Diagnosis and Prognosis for Industrial Systems: A Review. *Sensors* **2023**, *23*, 1305. [CrossRef]
4. Tian, Z.; Zhuo, M.; Liu, L.; Chen, J.; Zhou, S. Anomaly detection using spatial and temporal information in multivariate time series. *Sci. Rep.* **2023**, *13*, 4400. [CrossRef]
5. Stoorvogel, A.; Niemann, H.; Saberi, A. Delays in fault detection and isolation. In Proceedings of the 2001 American Control Conference, (Cat. No.01CH37148), Arlington, VA, USA, 25–27 June 2001; Volume 1, pp. 459–463. [CrossRef]
6. Yin, S.; Ding, S.X.; Haghani, A.; Hao, H.; Zhang, P. A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark Tennessee Eastman process. *J. Process. Control* **2012**, *22*, 1567–1581. [CrossRef]
7. Downs, J.J.; Vogel, E.F. A plant-wide industrial process control problem. *Comput. Chem. Eng.* **1993**, *17*, 245–255. [CrossRef]

8. Ge, Z. Review on data-driven modeling and monitoring for plant-wide industrial processes. *Chemom. Intell. Lab. Syst.* **2017**, *171*, 16–25. [CrossRef]
9. Webert, H.; Döß, T.; Kaupp, L.; Simons, S. Fault Handling in Industry 4.0: Definition, Process and Applications. *Sensors* **2022**, *22*, 2205. [CrossRef] [PubMed]
10. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: https://www.deeplearningbook.org/ (accessed on 12 July 2023).
11. Saufi, S.R.; Ahmad, Z.A.B.; Leong, M.S.; Lim, M.H. Challenges and Opportunities of Deep Learning Models for Machinery Fault Detection and Diagnosis: A Review. *IEEE Access* **2019**, *7*, 122644–122662. [CrossRef]
12. Ganaie, M.; Hu, M.; Malik, A.; Tanveer, M.; Suganthan, P. Ensemble deep learning: A review. *Eng. Appl. Artif. Intell.* **2022**, *115*, 105151. [CrossRef]
13. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
14. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5998–6008.
15. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. In Proceedings of the NIPS 2014 Workshop on Deep Learning, Montreal, QC, Canada, 8–13 December 2014.
16. Zeyer, A.; Bahar, P.; Irie, K.; Schlüter, R.; Ney, H. A comparison of transformer and lstm encoder decoder models for asr. In Proceedings of the 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Singapore, 14–18 December 2019; pp. 8–15. [CrossRef]
17. Lv, F.; Wen, C.; Bao, Z.; Liu, M. Fault diagnosis based on deep learning. In Proceedings of the 2016 American Control Conference (ACC), Boston, MA, USA, 6–8 July 2016; pp. 6851–6856.
18. Lee, K.; Cheon, S.; Kim, C. A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes. *IEEE Trans. Semicond. Manuf.* **2017**, *30*, 135–142. [CrossRef]
19. Borghesi, A.; Bartolini, A.; Lombardi, M.; Milano, M.; Benini, L. Anomaly Detection Using Autoencoders in High Performance Computing Systems. *Proc. AAAI Conf. Artif. Intell.* **2019**, *33*, 9428–9433. [CrossRef]
20. Qian, J.; Song, Z.; Yao, Y.; Zhu, Z.; Zhang, X. A review on autoencoder based representation learning for fault detection and diagnosis in industrial processes. *Chemom. Intell. Lab. Syst.* **2022**, *231*, 104711. [CrossRef]
21. Han, P.; Ellefsen, A.L.; Li, G.; Holmeset, F.T.; Zhang, H. Fault Detection With LSTM-Based Variational Autoencoder for Maritime Components. *IEEE Sens. J.* **2021**, *21*, 21903–21912. [CrossRef]
22. Colbrook, M.J.; Antun, V.; Hansen, A.C. The difficulty of computing stable and accurate neural networks: On the barriers of deep learning and Smale's 18th problem. *Proc. Natl. Acad. Sci. USA* **2022**, *119*, e2107151119. [CrossRef]
23. Akai, N.; Hirayama, T.; Murase, H. Experimental stability analysis of neural networks in classification problems with confidence sets for persistence diagrams. *Neural Netw.* **2021**, *143*, 42–51. [CrossRef]
24. MathWorks. Chemical Process Fault Detection Using Deep Learning. 2023. Available online: https://www.mathworks.com/help/deeplearning/ug/chemical-process-fault-detection-using-deep-learning.html (accessed on 12 July 2023).
25. Yan, W.; Guo, P.; Gong, L.; Li, Z. Nonlinear and robust statistical process monitoring based on variant autoencoders. *Chemom. Intell. Lab. Syst.* **2016**, *158*, 31–40. [CrossRef]
26. Torabi, H.; Mirtaheri, S.L.; Greco, S. Practical autoencoder based anomaly detection by using vector reconstruction error. *Cybersecurity* **2023**, *6*, 1. [CrossRef]
27. Barrera, J.M.; Reina, A.R.; Maté, A.; Trujillo, J.C. Fault detection and diagnosis for industrial processes based on clustering and autoencoders: A case of gas turbines. *Int. J. Mach. Learn. Cybern.* **2022**, *13*, 3113–3129. [CrossRef]
28. Bulatov, A.; Kuratov, Y.; Burtsev, M.S. Scaling Transformer to 1M tokens and beyond with RMT. *arXiv* **2023**, arXiv:2304.11062.
29. Rice, L.; Wong, E.; Kolter, Z. Overfitting in adversarially robust deep learning. In Proceedings of the International Conference on Machine Learning, Virtual Event, 13–18 July 2020; pp. 8093–8104.
30. Kawaguchi, K.; Bengio, Y.; Kaelbling, L. Generalization in Deep Learning. In *Mathematical Aspects of Deep Learning*; Grohs, P., Kutyniok, G., Eds.; Cambridge University Press: Cambridge, UK, 2022; pp. 112–148. [CrossRef]
31. Deb, K. *Multi-Objective Optimization Using Evolutionary Algorithms*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2001.
32. Xie, Z.; Chen, J.; Feng, Y.; Zhang, K.; Zhou, Z. End to end multi-task learning with attention for multi-objective fault diagnosis under small sample. *J. Manuf. Syst.* **2022**, *62*, 301–316. [CrossRef]
33. Lee, K.; Han, S.; Pham, V.H.; Cho, S.; Choi, H.J.; Lee, J.; Noh, I.; Lee, S.W. Multi-Objective Instance Weighting-Based Deep Transfer Learning Network for Intelligent Fault Diagnosis. *Appl. Sci.* **2021**, *11*, 2370. [CrossRef]
34. TensorFlow. timeseries_dataset_from_array. 2022. Available online: https://www.tensorflow.org/api_docs/python/tf/keras/utils/timeseries_dataset_from_array (accessed on 12 July 2023).
35. Chiang, L.; Russell, E.; Braatz, R. *Fault Detection and Diagnosis in Industrial Systems*; Springer: London, UK, 2001.
36. Heo, S.; Lee, J.H. Fault detection and classification using artificial neural networks. *IFAC-PapersOnLine* **2018**, *51*, 470–475. [CrossRef]
37. Heo, S.; Lee, J. Statistical process monitoring of the Tennessee Eastman Process using parallel autoassociative neural networks and a large dataset. *Processes* **2019**, *7*, 411. [CrossRef]

38. Sun, W.; Paiva, A.R.; Xu, P.; Sundaram, A.; Braatz, R.D. Fault detection and identification using Bayesian recurrent neural networks. *Comput. Chem. Eng.* **2020**, *141*, 106991. [CrossRef]
39. Park, P.; Marco, P.; Shin, H.; Bang, J. Fault Detection and Diagnosis Using Combined Autoencoder and Long Short-Term Memory Network. *Sensors* **2019**, *19*, 4612. [CrossRef]
40. Rieth, C.; Amsel, B.; Tran, R.; Cook, M. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation, Harvard Dataverse, 2017. Available online: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6C3JR1 (accessed on 12 July 2023).
41. Jurman, G.; Riccadonna, S.; Furlanello, C. A Comparison of MCC and CEN error measures in multi-class prediction. *PLoS ONE* **2012**, *7*, e41882. [CrossRef]
42. Wolpert, D.; Macready, W. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]