



Article

Fault Detection on the Edge and Adaptive Communication for State of Alert in Industrial Internet of Things

Yuri Santo ¹, Roger Immich ² , Bruno L. Dalmazo ³  and André Riker ^{1,*}¹ Institute of Exact and Natural Sciences (ICEN), Federal University of Pará, Belém 66075-110, Brazil² Metropole Digital Institute (IMD), Federal University of Rio Grande do Norte (UFRN), Natal 59078-970, Brazil³ Computer Science Center (C3), Federal University of Rio Grande, Rio Grande 96203-900, Brazil

* Correspondence: ariker@ufpa.br

Abstract: Industrial production and manufacturing systems require automation, reliability, as well as low-latency intelligent control. Industrial Internet of Things (IIoT) is an emerging paradigm that enables precise, low latency, intelligent computing, supported by cutting-edge technology such as edge computing and machine learning. IIoT provides some of the essential building blocks to drive manufacturing systems to the next level of productivity, efficiency, and safety. Hardware failures and faults in IIoT are critical challenges to be faced. These anomalies can cause accidents and financial loss, affect productivity, and mobilize staff by producing false alarms. In this context, this article proposes a framework called Detection and Alert State for Industrial Internet of Things Faults (DASIF). The DASIF framework applies edge computing to execute highly precise and low latency machine learning models to detect industrial IoT faults and autonomously enforce an adaptive communication policy, triggering a state of alert in case of fault detection. The state of alert is a pre-stage countermeasure where the network increases communication reliability by using data replication combined with multiple-path communication. When the system is under alert, it can process a fine-grained inspection of the data for efficient decision-making. DASIF performance was obtained considering a simulation of the IIoT network and a real petrochemical dataset.



Citation: Santo, Y.; Immich, R.; Dalmazo, B.L.; Riker, A. Fault Detection on the Edge and Adaptive Communication for State of Alert in Industrial Internet of Things. *Sensors* **2023**, *23*, 3544. <https://doi.org/10.3390/s23073544>

Academic Editors: Nancy Alonistioti, Spyros Panagiotakis and Evangelos K. Markakis

Received: 31 December 2022

Revised: 25 January 2023

Accepted: 28 January 2023

Published: 28 March 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: Industrial Internet of Things (IIoT); machine learning; edge computing

1. Introduction

Machine learning (ML) has been largely agreed upon as a key building block for the Industrial Internet of Things (IIoT). ML can support intelligent and quick decisions needed in IIoT environments [1]. Without fast and precise detection and decision-making, manufacturing is susceptible to all sorts of delays, which can cause financial loss and may represent a security threat.

In IIoT systems, the benefits of ML are potentiated by the capabilities of edge computing. By definition, edge computing has emerged as a paradigm to increase the computational power of IoT systems, providing extra storage and processing [2]. Edge computing devices are deployed near IoT devices to reduce the time necessary to complete computationally demanding tasks.

Hardware failures and faults in IIoT are critical challenges to be faced. A vast set of faults can occur in IIoT devices, including drift, spike, stuck, offset, bias, gain, out-of-bounds, saturation, and precision degradation [3]. All these anomalies can cause accidents, halt production, and mobilize staff by producing false alarms.

To mitigate the problems caused by IIoT faults, the next generation of IIoT systems requires: (i) precise and early fault detection; and (ii) autonomic response and reaction. The first requirement demands constant monitoring of the produced data, and the second, not least important, consists of autonomously reacting when the fault has been detected. In this context, modern industrial business models consider that solving a fault in its early stages prevents collateral damage and results in fewer costs.

Many works in the literature propose solutions to meet the requirements of IIoT. Most of them are based on ML classifiers to support precise fault detection. Only some works seek to minimize the response time during the detection. Additionally, a reduced number of works provides any autonomous reaction for detected faults. To fill this gap, this article proposes a framework called Detection and Alert State for Industrial Internet-of-Things Faults (DASIF). The DASIF framework applies highly precise and low latency machine learning models to detect IIoT faults and also autonomously enforce a state of alert in case of detecting the fault. The faults are double classified on the edge computing devices using a decision tree and Gaussian naive Bayes. The state of alert enforced by DASIF is the first-stage countermeasure for fine-grained data analysis. Under this state, the network decreases the communication interval and uses data replication and multiple-path communication to achieve higher communication reliability. In our previous work [4], we conducted a study to select the best ML models for the DASIF framework. In [4], we evaluated six machine learning classifiers measuring accuracy, precision, recall, F1 score, training time, and response time.

The contributions of this article are the following:

- to propose an ML-based framework running on edge computing for detecting Internet of Things (IoT) faults in industrial environments;
- to present a Markov chain-based algorithm to inject a set of faults into IIoT datasets;
- to enforce an adaptive communication policy, instituting a state of alert as an autonomous reaction when a fault has been detected.

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 presents the proposed framework, and Section 4 introduces the evaluation scenario, settings, and the obtained results. In the end, Section 5 presents the conclusions and potential directions for future research.

2. Related Work

A vast literature applies machine learning models to support smart decisions in industrial IoT environments. Most of these works propose solutions for anomaly detection related to security aspects. A reduced number of works aims to detect anomalies in the sensed data produced by IoT devices and caused by internal hardware and software malfunctions or external factors, such as vibration.

To detect faults and errors in IoT devices using machine learning classifiers, Jan et al. [5,6] propose a diagnosis system to detect sensor fault. The authors consider devices with limited computation resources, such as memory, processing, and energy. This solution is distributed and based on a Support Vector Machine (SVM) model, where response time is not the top priority for the application.

Saeed et al. [7] analyze different classifiers using a dataset with drift fault injection. The faults were injected using real data from a digital relative temperature/humidity sensor (DHT22) and an Arduino controller. For detection, a Raspberry Pi is used. The performance was measured using a large set of machine learning models: SVM, ANN, naive Bayes, KNN, and decision tree. The authors compare the performance of the machine learning models in terms of precision, recall, F1 score, and total accuracy.

Javaid et al. [3] aim to detect and diagnose faults based on decision fusion with different classification techniques named: Enhanced K-Nearest Neighbor (EKNN), Enhanced Extreme Learning Machine (EELM), Enhanced Support Vector Machine (ESVM), and Enhanced Recurrent Extreme Learning Machine (ERELM). The authors consider the offset, gain, stuck, and out-of-bounds faults.

Zidi et al. [8] apply machine learning for fault detection in wireless sensor networks. The solution considers an SVM model for dealing with offset, gain, stuck-at, and out-of-bounds faults in a real dataset from the University of North Carolina at Greensboro [9].

In a nuclear power plant scenario, Naimi et al. [10] propose fault detection and diagnosis based on neural networks and a KNN algorithm applied to a pressurized water reactor. First, the neural network performs detection. Second, the KNN algorithm classifies

the faults. The KNN performance is also compared to neural networks and SVM. This work includes bias, drift, actuator offset, and saturation faults.

Khodabaksh et al. [11] consider a method for real-time data validation, gross error detection, and classification. This work is based on data from petrochemical power plants of an oil refinery. The injection of bias, drift, and precision degradation failures was based on statistical studies of sensor data and the observation of changes in mean and variances. The classification relies on the complex decision tree, neural network, and KNN algorithms. The performance was measured using precision and recall. The authors made available the dataset for academic use [12].

So far, all the mentioned works do not consider any countermeasure to be applied after the faults have been detected. Dofe et al. [13] present a comprehensive perspective on countermeasures against IoT attacks. Mustafa et al. [14] propose three countermeasures to preserve privacy when abnormal behavior is detected in assisted living applications.

Summing up, many works explore fault detection through the SVM classifier as can be observed in Jan et al. [5,6] and Zidi et al. [8]. However, an industrial scenario requires a sensitive time application to deal with faults, but the SVM demands higher computation time. Naimi et al. [10] also do not consider the necessary time to apply a two-phase solution. Additionally, how to react after the fault has been detected is an important aspect, but countermeasures are often considered only from the security perspective. Therefore, there is a lack of novel solutions capable of detecting and reacting against possible faulty situations in IIoT environments.

3. Detection and Alert State for Industrial Internet of Things Faults (DASIF)

The main goal of the proposed framework is to provide fast and precise fault detection and reaction in industrial internet of things devices. The rest of this section is organized as follows. Section 3.1 details the industrial scenario and the overview of the proposed solution. Section 3.2 introduces the machine learning classifiers used in DASIF. Section 3.3 presents the details of the industrial dataset and how the faults have been injected. Section 3.4 describes the adaptive communication policy that sets a state of alert when a fault has been detected.

3.1. Overview and Proposed Framework

Figure 1 illustrates an industrial environment with silos, conveyor belts, boilers, pipes, turbines, energy production, storage boxes, vehicles, robots, and oil barrels. All these objects and machinery are monitored and actuated by a set of IoT devices, which send and receive traffic via a wireless network to the system control.

Typically, the IoT network traffic feeds the system control with the pressure in the pipes, the turbines, rotation, the vehicles, position, the temperature of the boilers, the speed of the conveyor belt, and many other data.

The system control offloads the IoT data to edge computing since it provides cloud services near the network devices, supporting high computational power and low delay. The processing power in the edge is capable of analyzing a massive amount of IoT traffic and providing multiple data services, including intrusion and fault detection. Edge computing can successfully apply algorithms to support dynamic resource allocation, e.g., processing and memory, to meet energy or response time requirements.

In this context, the proposed framework, called Detection and Alert State for Industrial Internet of Things Faults (DASIF), is a solution designed to detect IoT faults in industrial environments. As shown in Figure 2, the DASIF framework has a system control in which IoT traffic is inspected and forwarded to the edge. Additionally, the system control provides data visualization and issues commands for the automation of the factory.

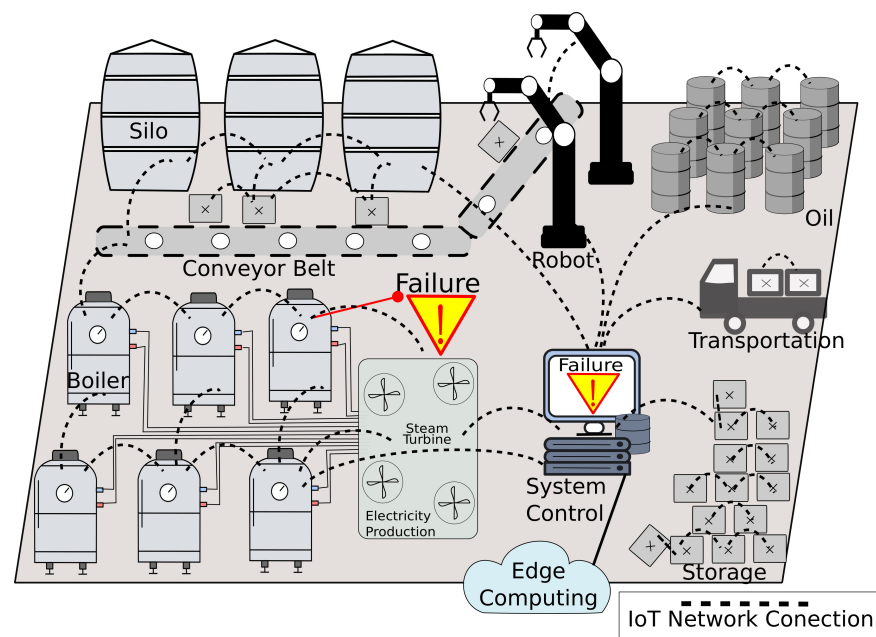


Figure 1. Industry scenario.

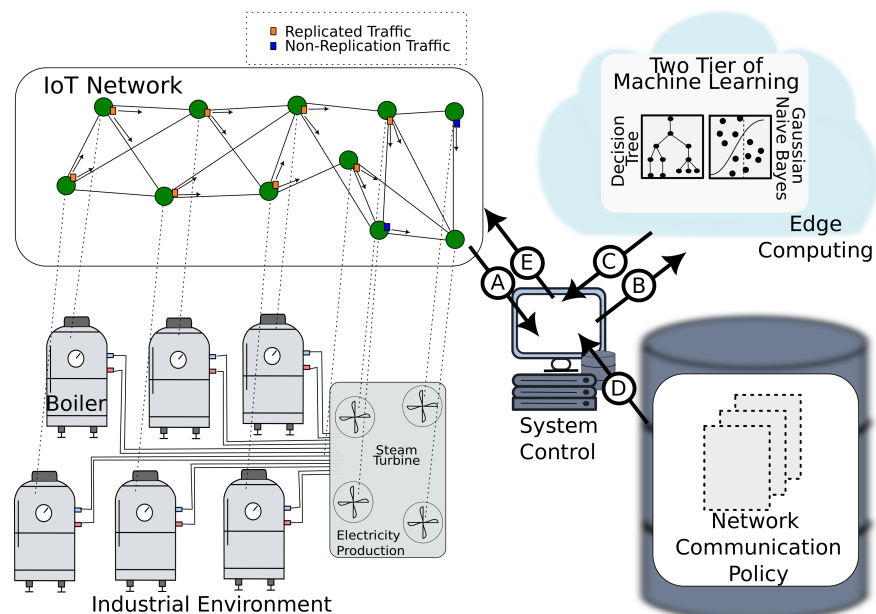


Figure 2. Detection and Alert State for Industrial Internet of Things Faults (DASIF) framework.

The IoT devices deployed in an extensive number support monitoring, automation, and communication. However, these devices are prone to errors and faults, which distort the measured values. For instance, IoT devices can abruptly stop monitoring the target, sending invalid pressure values in a pipe or a valve. These errors and faults can cause severe damage to manufacturing, including accidents, financial loss, and false alarms.

In this framework, edge computing provides fast processing of machine learning classifiers. Based on the study performed in our previous work [4], DASIF relies on a two-tier machine learning for precise and fast IoT fault detection. The fault classification is double-checked using a decision tree and Gaussian naive Bayes.

Another component of this framework is the repository of communication policy. In this repository, the network communication settings are defined for a set of situations the industrial system must deal with. In this work, DASIF foresees normal and alert policies. A detected IoT fault triggers the alert policy. Under an alert policy, the IoT

network communicates using data replication and with more intense traffic production. This reaction enables the system to perform fine-grained checks on the error and provide information for critical decision-making, including triggering an unmanned aerial vehicle (UAV) to visit the location where the alert is being issued to verify in loco the hardware conditions and make adjustments to the IoT devices.

To achieve IoT fault detection, DASIF executes five tasks as indicated by the labels in Figure 2 and are described as follows.

- **Label A:** The system controller receives and inspects the traffic produced by the deployed IoT network.
- **Label B:** The relevant data for fault detection are sent to the edge.
- **Label C:** The machine learning output containing the fault classification is sent back to the control system.
- **Label D:** The system control selects an appropriate communication policy according to the classification received from the machine learning models.
- **Label E:** The control system enforces the communication policy, issuing messages to the nodes.

3.2. Machine Learning Classifiers for Fault Detection

DASIF has a machine learning layer that detects the faults of IoT devices. This layer applies a double classification using a decision tree and Gaussian naive Bayes (GNB). Decision trees are a top-down, divide-and-conquer approach to supervised classification. At the beginning of the training phase, all training samples are assigned to the root node. These training samples are partitioned and assigned to child nodes to increase the purity of the resulting child nodes. The procedure is repeated at each node until the leaf nodes have training examples of a single class, i.e., the leaf nodes are pure [15]. A naive Bayesian classifier is a simple and efficient classifier based on the Bayesian theory. However, it is well known that NBC is based on the assumption that all attributes are independent of each other [16]. A Gaussian naive Bayes classification is a case of the naive Bayes method with an assumption of having a Gaussian distribution of attribute values given the class label.

The single classifier has a limited ability to deal with the problem of a larger amount of data. Multiple classifier systems can improve the performance of a single classifier, especially in critical systems such as industrial internet of things. In DASIF, the combination of these two machine learning models assures that decision-making is based on two distinct classifiers, increasing diversity in assumptions and training processes. The DASIF final decision is Fault or Normal. DASIF output is a fault if any of the classifiers produces a fault classification.

3.3. Dataset Description and Fault Injection

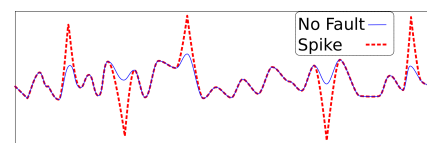
This article uses the IIoT dataset from Khodabakhsh et al. [11], which provides real-world measurements from over 1000 devices deployed at Turkish Petroleum Refineries Inc. (TUPRAS) power plants. The TUPRAS dataset is a sample of real-world data measured every minute and is available for academic use at [12].

This petrochemical dataset contains 200,000 flow sensor records (water, superheater, steam) sampled every 60 s in the TUPRAS power plant for approximately 5 months. These data are replicated 5 times to form 1 million rows to represent actual sensor loads better. Each row has records from 3 flow sensors in the power plant dataset and 17 flow sensors in the petrochemical dataset.

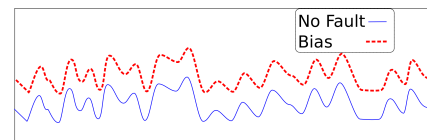
According to [5], faults are defined as deviations from expected behavior in the device output. The faults are data corruption behavior, which is related to the physical defects of the devices and their operational conditions.

In this paper, three faults have been considered, namely spike, stuck, and bias. As shown in Figure 3a, a spike fault is an effect observed as a large-amplitude value occurring at time intervals in the sensor output. Figure 3b illustrates the bias fault. In this fault, a shift from the normal value is observed since a constant value is added to the normal output. A

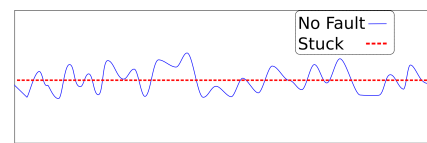
stuck fault (see Figure 3c), also called a complete fault, blocks the sensor output at a fixed value. This fault can be temporary or permanent.



(a) Spike fault.



(b) Bias fault.



(c) Stuck fault.

Figure 3. Illustration of faults.

The original TUPRAS dataset does not identify, i.e., label, any faults or errors. However, machine learning models that use supervised methods require labeled data to identify errors. Based on a statistical dataset analysis, we have developed Algorithm 1 to inject faults into the dataset and the appropriate labels for training and testing.

This algorithm is responsible for injecting stuck, bias, and spike faults. In order to achieve this goal, the ratio between the maximum value (l. 5) and the mean (l. 4) of the dataset is used. This ratio (l. 6) represents the percentage of max increase to the mean. Aiming to replicate the fault behavior, the functions presented in the algorithm deal with different calculations based on the same ratio, replacing some dataset values with fault values according to the Markov chain. The stuck function performs a decrease in the sensed value (l. 14) and becomes constant in the subsequent minutes. While the fault lasts, the bias function executes an increase in the sensed value (l. 29). The spike function performs increment (l. 44) and decrement (l. 47). These functions return a modified dataset with faults.

As can be observed, Algorithm 1 relies on Markov chain to determine the distribution of faults along the dataset time series. A two-state Markov chain has been used for this algorithm, as illustrated in Figure 4. The N and F denote the normal and fault behavior states, respectively. The states can be maintained, or the transitions between states can occur according to a defined probability.

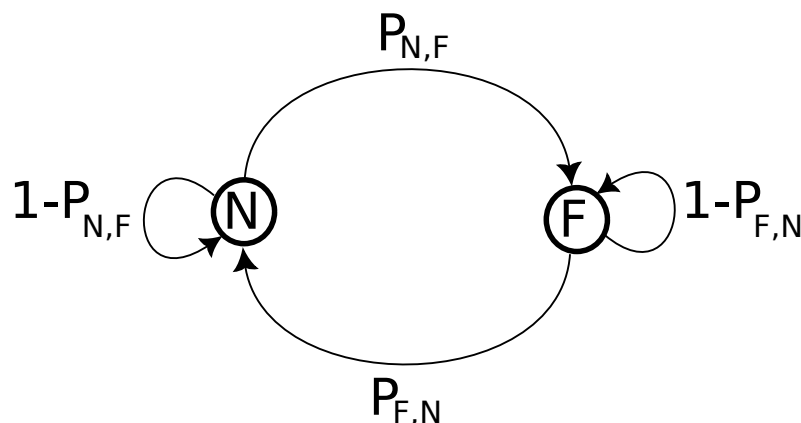


Figure 4. A two-state Markov chain to inject faults.

Algorithm 1 Fault Injection Algorithm

```

1: Input: original dataset.
2: Output: dataset with fault injection.
3: Start
4:   mean  $\leftarrow$  dataset mean
5:   max  $\leftarrow$  dataset maximum value
6:   ratio  $\leftarrow$  max / mean
7:
8:   function STUCK(dataset, ratio)
9:     stuckRate  $\leftarrow$  ratio
10:    stuckTemp  $\leftarrow$  0
11:    for i=1:dataset size do
12:      if index i is in Markov chain then
13:        if stuckTemp == 0 then
14:          stuckTemp  $\leftarrow$  ith value * (2-stuckRate)
15:          ith value of dataset  $\leftarrow$  stuckTemp
16:        else
17:          ith value of dataset  $\leftarrow$  stuckTemp
18:        end if
19:      end if
20:    end for
21:    return dataset
22:  end function
23:
24:  function BIAS(dataset, ratio)
25:    biasRate  $\leftarrow$  ratio
26:    biasTemp  $\leftarrow$  0
27:    for i = 1:dataset size do
28:      if index i is in Markov chain then
29:        biasTemp  $\leftarrow$  ith value * biasRate
30:        ith value of dataset  $\leftarrow$  biasTemp
31:      end if
32:    end for
33:    return dataset
34:  end function
35:
36:  function SPIKE(dataset, ratio)
37:    spikeRate  $\leftarrow$  ratio
38:    spikeTemp  $\leftarrow$  0
39:    random  $\leftarrow$  0
40:    for i = 1:dataset size do
41:      if index i is in Markov chain then
42:        random  $\leftarrow$  random integer number between 0 and 1
43:        if random == 0 then
44:          spikeTemp  $\leftarrow$  ith value * spikeRate
45:          ith value of dataset  $\leftarrow$  spikeTemp
46:        else
47:          spikeTemp  $\leftarrow$  ith value * (2-spikeRate)
48:          ith value of dataset  $\leftarrow$  spikeTemp
49:        end if
50:      end if
51:    end for
52:    return dataset
53:  end function
54: End

```

3.4. Network Reaction Policy for Detected Faults

DASIF is designed to be autonomous since an IoT network deployed in an industrial environment requires the management of hundreds or thousands of devices. Being autonomous means to be self-sufficient or self-healing, and self-protective. The main idea is to provide fast resource management of the system with low or no human intervention.

To achieve that, DASIF has two network communication policies, called normal and alert policies. The normal is applied when no fault has been detected. The alert policy is enforced after any fault has been detected, triggering a state of alert in the system, which is an early countermeasure stage. The communication policy sets the configuration for the communication interval (e.g., 60 s), data replication, data aggregation, multiple path communication, and channel check rate. These communication settings allow the system to perform fine-grained data analysis.

In this work, the DASIF alert policy applies a communication interval of 30 s, data replication, no data aggregation, multiple path communication, and 32 hz as the channel check rate. The multiple path communication is executed using two instances of the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL) protocol. Data are double-communicated via these two instances in order to successfully deliver the notifications even if some node in the path is compromised.

To find two different routes, DASIF forces the RPL instances to select different routes. The main idea for the route selection in the additional instance is: if the main path has the same nodes selected by the secondary path, then DASIF chooses the next best nodes for the secondary path among those devices previously passed in the RPL conditions. The only exception that allows the node parent in both paths is when the node is the sink node (i.e., the path length is 1).

As faults are rare, the normal policy is executed most of the time. This policy sets a communication able to detect the faults and preserve the resources of the devices, i.e., energy.

4. Performance Evaluation

DASIF has been evaluated in a simulation environment, considering the related characteristics of an industrial IoT environment. The details of how this evaluation has been conducted are shown in the following sections. Section 4.1 presents the evaluation environment and settings. Section 4.2 describes the metrics used to assess the proposed solution. Section 4.3 presents and discusses the obtained results.

4.1. Evaluation Settings

Regarding the dataset, Table 1 shows the settings used to run the tests. The original dataset has 44,643 values for 6 data types: water flow, water temperature, water pressure, steam flow, steam temperature, and steam pressure. The dataset was divided into 70% for training, 20% for tests, and 10% for validation, and it was injected with 446 (1%) faults.

Table 1. Parameters and settings.

Parameter	Value
Original Dataset	44,643 data values; TUPRAS dataset [11]
Data Type	Water flow, water temperature, water pressure Steam flow, steam temperature, steam pressure
Type of Fault	Spike, bias, stuck
Injected Faults	446 (1%)
Implementation	Python, scikit-learn library
Edge Computing	Hardware: i7-8700 3.2 GHz, 12 GB RAM
IoT Network	50 nodes

As shown in Matrix (1), the Markov chain states can be maintained, i.e., NN and FF, or a transition can occur, i.e., NF and FN, according to a probability. Matrices (2)–(4) present the Markov chain probabilities used in this evaluation for spike, stuck, and bias faults.

$$\text{Probabilities} = \begin{bmatrix} P_{N,N} & P_{N,F} \\ P_{F,N} & P_{F,F} \end{bmatrix} \quad (1)$$

$$\text{Stuck} = \begin{bmatrix} 0.9966 & 0.0034 \\ 0.3 & 0.7 \end{bmatrix} \quad (2)$$

$$\text{Spike} = \begin{bmatrix} 0.99 & 0.01 \\ 1.0 & 0.0 \end{bmatrix} \quad (3)$$

$$\text{Bias} = \begin{bmatrix} 0.9966 & 0.0034 \\ 0.3 & 0.7 \end{bmatrix} \quad (4)$$

The machine learning models were implemented in Python using the scikit-learn library [17] running on the Linux operating system. The hardware used for the tests seeks to reflect edge computing hardware. The following hardware was used for these tests: i7-8700 3.2 GHz, 12 GB RAM.

The IoT network has been implemented in Contiki OS and tested in the Cooja simulator [18]. The IoT network has 50 nodes deployed in the area of 100×100 m in a grid topology. The RPL multiple instances have been adapted from the solution proposed by Junior et al. [19].

4.2. Performance Metrics

The following set of metrics has been defined to measure the performance of the machine learning models:

- **Accuracy:** Indicates the number of correct predictions divided by the total number of predictions.
- **Precision:** Represented by the ratio between the number of correct positive classifications and the number of total positives.
- **Recall:** Represents the number of true positives divided by true positives and false negatives.
- **F1 Score:** Constitutes a harmonic mean between precision and recall. In this metric, 1.0 means excellent performance.

The following metrics assess the IoT network performance:

- **Success of notification delivery:** computed as the division between the number of messages reaching the sink and the total number of sent messages. The duplicated messages count as one notification since they will be eliminated in case of duplicated reception in the sink.
- **Energy consumption:** calculated by the sum of all spent energy of every node during the entire simulation time. This evaluation used the kinetic battery model implemented by Riker et al. [20].
- **Delay:** measures how long it takes to send a message from the IoT node, deliver it to the edge, and detect a fault.

4.3. Obtained Results

Figure 5 presents the spike fault detection performance in terms of accuracy, precision, recall, and F1 score.

As can be observed, all models achieve higher than 0.90 when all performance metrics are considered, except for water temperature. A similar performance is observed in Figure 6 for bias fault detection. In general, the performance of fault detection is high. It is important to notice that DASIF considers both classifiers' outputs in order to detect any fault. This

means that if any of the classifiers produce a fault classification, DASIF triggers the state of alert for the system, enforcing the alert communication policy.

Regarding the worst detection performance, i.e., water temperature, the decision tree obtained, on average, 0.54 considering precision, recall, and F1 score for spike faults. For the same datatype and fault type, GNB obtained 0.37. Considering the bias fault, the results for water temperature are 0.49 and 0.38 for the decision tree and GNB, respectively. These results show that the performance in terms of precision, recall, and F1 score is poor for water temperature, but accuracy is high since it is more than 0.98 for both classifiers in spike and bias faults. This occurs because accuracy is a percent value of correct predictions; it is not recommended for an unbalanced dataset. For instance, in a dataset with 1% faults, the accuracy would be 99% if the model predicts zero faults.

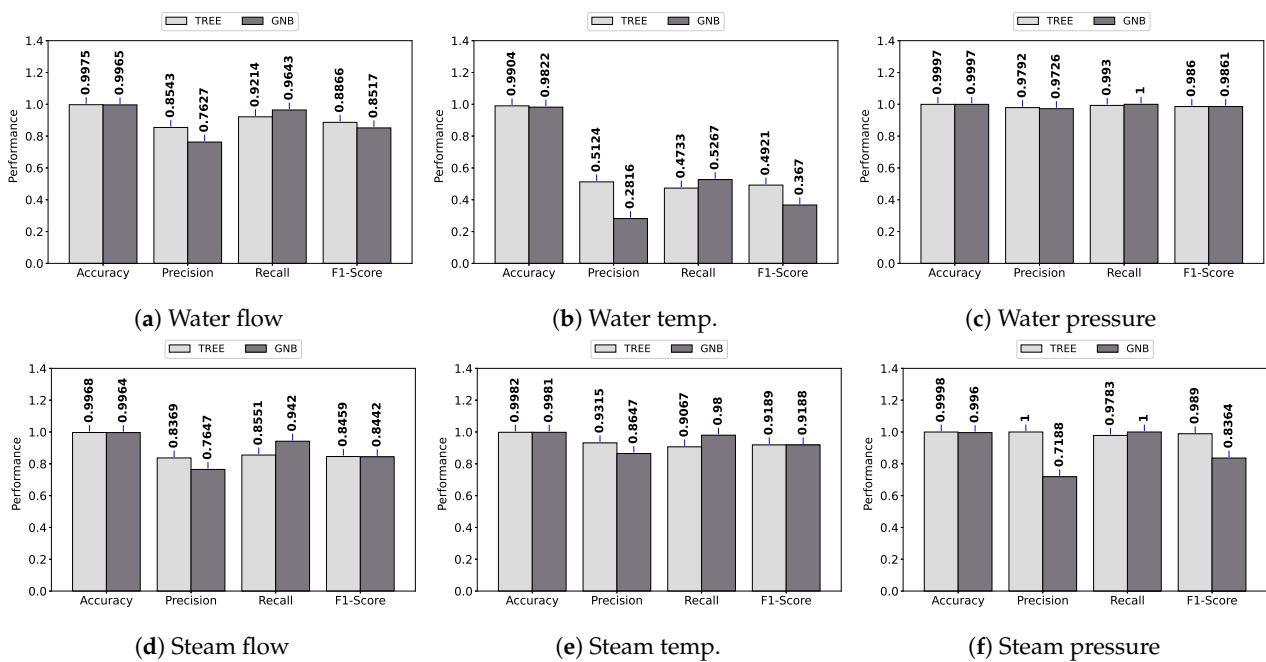


Figure 5. Spike fault.

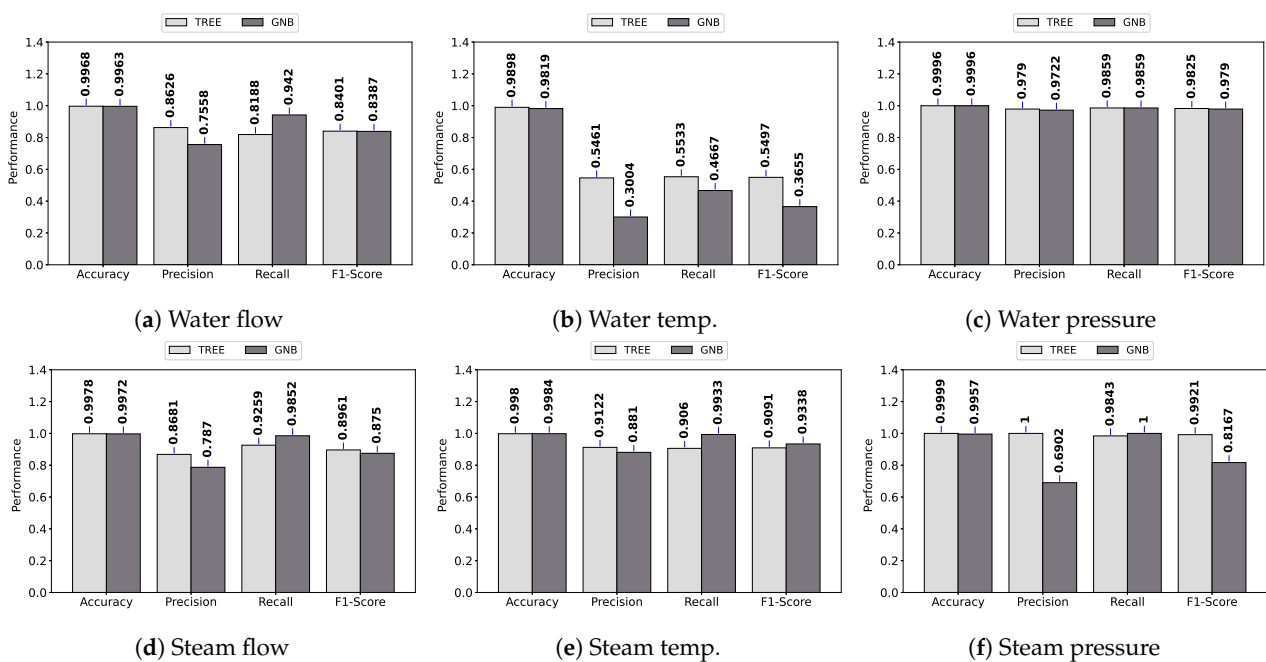


Figure 6. Bias fault.

Figure 7 shows a higher performance for stuck faults compared to spike and bias. As the stuck fault produces a constant, its faulty pattern can easily be identified by the classifiers.

Table 2 presents the delay results. It is worth mentioning that it refers to the time to communicate and detect a fault. It can be observed that the average delay values are 3.71 and 4.18 for paths 1 and 2, respectively. Path 2 has a higher delay because it tends to select longer paths. The DASIF algorithm for path selection avoids the same nodes selected for path 1. This multiple-path communication strategy enables path diversification, which contributes to higher reliability.

Another aspect in Table 2 is that the time difference between the decision tree and GNB is less than 0.1 milliseconds. Additionally, machine learning classification running on edge computing takes about 2 to 4 milliseconds, while communication is responsible for more than 3.7 to 4.1 s.

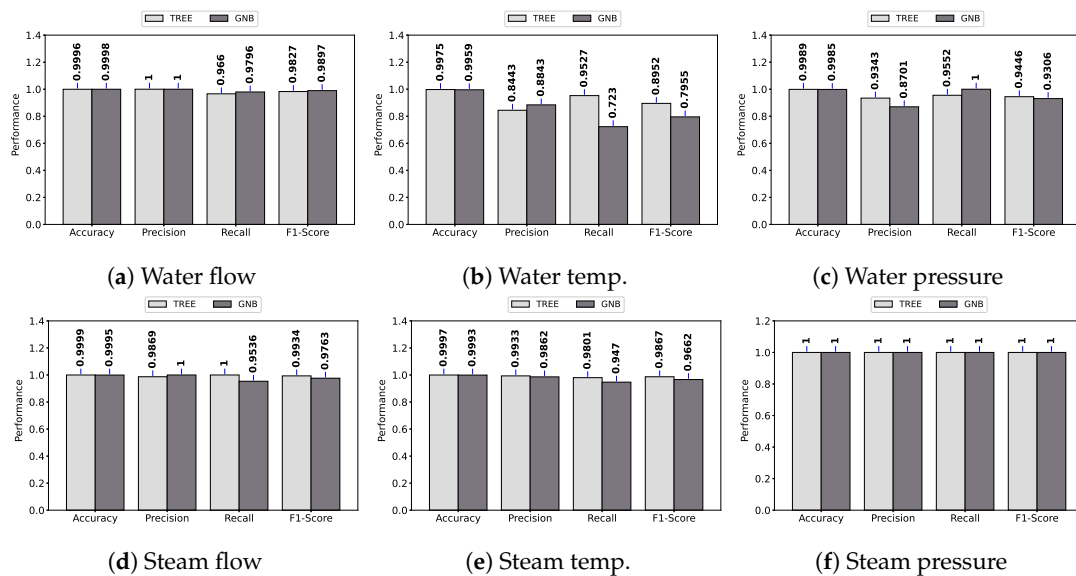


Figure 7. Stuck fault.

Table 2. Delay (seconds) to communicate and detect the fault.

Data Type	Stuck		Spike		Bias	
	TREE	GNB	TREE	GNB	TREE	GNB
Water flow						
Path 1	3.7156	3.7157	3.7159	3.7153	3.7162	3.7154
Path 2	4.1839	4.1840	4.1841	4.1835	4.1844	4.1837
Water temperature						
Path 1	3.7155	3.7152	3.7157	3.7154	3.7157	3.7154
Path 2	4.1838	4.1835	4.1840	4.1837	4.1839	4.1837
Water pressure						
Path 1	3.7157	3.7153	3.7162	3.7155	3.7157	3.7153
Path 2	4.1840	4.1836	4.1844	4.1837	4.1839	4.1836
Steam flow						
Path 1	3.7156	3.7153	3.7159	3.7154	3.7160	3.7154
Path 2	4.1839	4.1835	4.1842	4.1837	4.1842	4.1837
Steam temperature						
Path 1	3.7156	3.7153	3.7155	3.7155	3.7158	3.7157
Path 2	4.1839	4.1836	4.1838	4.1838	4.1841	4.1839
Steam pressure						
Path 1	3.7157	3.7153	3.7156	3.7154	3.7157	3.7156
Path 2	4.1839	4.1836	4.1838	4.1836	4.1840	4.1838

Table 3 presents the results of the success of notification delivery and energy. The success rate is 92.76%, considering all communicated application traffic related to the 50 nodes. Regarding energy, the sum of energy consumed by all nodes is 139.17 microA.

Table 3. Success of notification delivery and energy.

Performance	Value
Message Delivery Ratio (%)	92.76
Battery (microA)	139.17

5. Conclusions and Future Work

Internet of Things and machine learning algorithms applied to industrial environments have caused a revolution in manufacturing systems over the last few years. It is agreed that Industrial Internet of Things can meet some crucial requirements using machine learning and adding intelligence to the continuous monitoring and controlling process. In this context, edge computing is another trend, mainly due to the necessity of achieving low latency on computational tasks in industrial IoT applications.

However, new challenges emerge when the IoT paradigm is added to industrial systems. One of the challenges is that manufacturing systems are prone to faults in IoT devices. These faults represent a risk for the factory since they can result in dangerous events and stop production. Therefore, it is essential to rely on precise and fast systems to detect and react to the occurrence of IoT faults.

This article proposes a framework called Detection and Alert State for Industrial Internet of Things Faults (DASIF). This framework applies machine learning models to detect IIoT faults and also autonomously enforces a state of alert in case of detection of faults. The faults are double classified on the edge computing devices using decision tree and Gaussian naive Bayes. As a reaction, the state of alert enforced by DASIF is the first stage countermeasure for fine-grained data analysis.

In future work, we intend to use federated learning to create a detection system able to run distributively, which can be executed on many industrial sites, maintaining data privacy. Additionally, we will conduct tests in a real environment and design the detection of a more extensive list of faults.

Furthermore, for a functional federated learning solution with heterogeneous data from different machines, semantic schemes and interoperability challenges must be tackled. Some possible approaches are meta-based learning and semantic feature representation.

Author Contributions: Conceptualization, Y.S., B.L.D. and A.R.; methodology, Y.S., B.L.D. and A.R.; software, Y.S.; validation, Y.S. and A.R.; investigation, Y.S., B.L.D., R.I. and A.R.; writing—original draft preparation, Y.S. and A.R.; writing—review and editing, Y.S., B.L.D., R.I. and A.R.; supervision, Y.S., B.L.D. and A.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by PROPEP/UFP (PAPQ).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Fang, W.; Xue, F.; Ding, Y.; Xiong, N.; Leung, V.C. EdgeKE: An on-demand deep learning IoT system for cognitive big data on industrial edge devices. *IEEE Trans. Ind. Inform.* **2020**, *17*, 6144–6152. [[CrossRef](#)]
2. Fazio, M.; Ranjan, R.; Girolami, M.; Taheri, J.; Dustdar, S.; Villari, M. A note on the convergence of IoT, edge, and cloud computing in smart cities. *IEEE Cloud Comput.* **2018**, *5*, 22–24. [[CrossRef](#)]

3. Javaid, A.; Javaid, N.; Wadud, Z.; Saba, T.; Sheta, O.E.; Saleem, M.Q.; Alzahrani, M.E. Machine learning algorithms and fault detection for improved belief function based decision fusion in wireless sensor networks. *Sensors* **2019**, *19*, 1334. [[CrossRef](#)] [[PubMed](#)]
4. Santo, Y.; Dalmazo, B.L.; Immich, R.; Riker, A. On the Performance of Machine Learning at the Network Edge to Detect Industrial IoT Faults. In Proceedings of the 2022 IEEE 21st International Symposium on Network Computing and Applications (NCA), Boston, MA, USA, 14–16 December 2022; Volume 21, pp. 291–295.
5. Jan, S.U.; Lee, Y.D.; Koo, I.S. A distributed sensor-fault detection and diagnosis framework using machine learning. *Inf. Sci.* **2021**, *547*, 777–796. [[CrossRef](#)]
6. Jan, S.U.; Lee, Y.D.; Shin, J.; Koo, I. Sensor fault classification based on support vector machine and statistical time-domain features. *IEEE Access* **2017**, *5*, 8682–8690. [[CrossRef](#)]
7. Saeed, U.; Jan, S.U.; Lee, Y.D.; Koo, I. Machine learning-based real-time sensor drift fault detection using Raspberry PI. In Proceedings of the 2020 International Conference on Electronics, Information, and Communication (ICEIC), Barcelona, Spain, 19–22 January 2020; pp. 1–7.
8. Zidi, S.; Moulahi, T.; Alaya, B. Fault detection in wireless sensor networks through SVM classifier. *IEEE Sens. J.* **2017**, *18*, 340–347. [[CrossRef](#)]
9. Suthaharan, S.; Alzahrani, M.; Rajasegarar, S.; Leckie, C.; Palaniswami, M. Labelled data collection for anomaly detection in wireless sensor networks. In Proceedings of the 2010 Sixth International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Brisbane, QLD, Australia, 7–10 December 2010; pp. 269–274.
10. Naimi, A.; Deng, J.; Shimjith, S.; Arul, A.J. Fault Detection and Isolation of a Pressurized Water Reactor Based on Neural Network and K-Nearest Neighbor. *IEEE Access* **2022**, *10*, 17113–17121. [[CrossRef](#)]
11. Khodabakhsh, A.; Ari, I.; Bakir, M.; Ercan, A.O. Multivariate sensor data analysis for oil refineries and multi-mode identification of system behavior in real-time. *IEEE Access* **2018**, *6*, 64389–64405. [[CrossRef](#)]
12. Tupras Refineries Dataset. Available online: <https://www.openml.org/d/41170> (accessed on 1 October 2022).
13. Dofe, J.; Nguyen, A.; Nguyen, A. Unified Countermeasures against Physical Attacks in Internet of Things—A survey. In Proceedings of the 2021 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS), Jaipur, India, 18–22 December 2021; pp. 194–199.
14. Mustafa, M.A.; Konios, A.; Garcia-Constantino, M. IoT-Based Activities of Daily Living for Abnormal Behavior Detection: Privacy Issues and Potential Countermeasures. *IEEE Internet Things Mag.* **2021**, *4*, 90–95. [[CrossRef](#)]
15. Li, Y.; Dong, M.; Kothari, R. Classifiability-based omnivariate decision trees. *IEEE Trans. Neural Netw.* **2005**, *16*, 1547–1560. [[CrossRef](#)] [[PubMed](#)]
16. Yang, G.; Gu, X. Fault diagnosis of complex chemical processes based on enhanced naive Bayesian method. *IEEE Trans. Instrum. Meas.* **2019**, *69*, 4649–4658. [[CrossRef](#)]
17. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
18. Österlind, F. A sensor network simulator for the Contiki OS. *SICS Res. Rep.* **2006**.
19. Junior, S.; Riker, A.; Silvestre, B.; Moreira, W.; Oliveira, A., Jr.; Borges, V. DYNASTI—Dynamic Multiple RPL Instances for Multiple IoT Applications in Smart City. *Sensors* **2020**, *20*, 3130. [[CrossRef](#)]
20. Riker, A.; Curado, M.; Monteiro, E. Neutral operation of the minimum energy node in energy-harvesting environments. In Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC), Heraklion, Greece, 3–6 July 2017; pp. 477–482.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.