CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Using Machine Learning for Accelerating the Detection Time of Unreliable Failure Detectors

Master's thesis in Computer science and engineering

CHENGBO REN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Using Machine Learning for Accelerating the Detection Time of Unreliable Failure Detectors

CHENGBO REN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Using Machine Learning for Accelerating the Detection Time of Unreliable Failure Detectors

CHENGBO REN

© CHENGBO REN, 2024.

Supervisor: Elad Michael Schiller, Department of Computer Science and Engineering
Advisor: Hasan Mahmood, Nexer Group
Examiner: Romaric Duvignau, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Using Machine Learning for Accelerating the Detection Time of Unreliable Failure Detectors

CHENGBO REN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This study evaluates Machine Learning-based Failure Detectors for detecting faulty nodes in distributed systems, focusing on metrics like Detection Time and Average Mistake Rate. Random Forest and Support Vector Machine Failure Detectors emerge as top performers. However, Machine Learning-based Failure Detectors exhibit higher error rates, especially with longer detection times. To address this, three ensemble methods are proposed, with one integrating top Machine Learning-based Failure Detectors and analytical Failure Detectors for better balance. The Precision Distribution module improves accuracy but struggles with completeness. One analytical Failure Detector is used as a fallback to address completeness concerns, although its effectiveness depends on the relationship between the analytical Failure Detector and Precision Distribution module thresholds. Additionally, the Li-Marin Long Short-Term Memory Failure Detector shows reduced detection time but with increased computational overhead, posing challenges in fine-tuning overestimation levels.

Keywords: Failure Detector, Machine Learning, Distributed Systems, Ensemble Methods.

Acknowledgements

On finishing the last word of my thesis report, I see a heart filled with gratitude beating in my chest.

I first need to thank my supervisor, Elad Michael Schiller, who guided me through the whole project with great patience and wisdom on not only academic research but also how to be a strong and honest man. His passion and devotion to his work have set a standard for me to reach regardless of my career choice in the future. I am so grateful to have him as my supervisor.

I extend my sincere appreciation to my company advisor, Hasan Mahmood. Hasan's practical insights, expert advice, and willingness to share his knowledge have played a pivotal role in enriching the relevance and applicability of my research within the corporate context. His support and encouragement have provided me with invaluable perspectives and guidance, enabling me to navigate the complexities of integrating academic theory with real-world practices effectively.

To my beloved wife and daughters, I owe an immeasurable debt of gratitude for their unwavering love, encouragement, and support throughout my thesis journey. Their constant belief in my abilities, unwavering patience during challenging times, and endless encouragement have been the cornerstone of my academic achievements. Whether it was offering words of wisdom, lending a listening ear, or providing practical assistance, my family's steadfast support has been the driving force behind my perseverance and success. I am profoundly grateful for their sacrifices, understanding, and unwavering belief in my dreams.

Chengbo Ren, Gothenburg, March 2024

Contents

List of Acronyms

ANN Artificial Neural Network. 8, 17

AUC Area Under Curve. 17, 22, 28, 30, 38

BDBD Blanchard, Dolev, Beauquier, Delaët. xv, 3, 7, 18, 19, 23–25, 27, 28, 30–33, 38, 39, 41

FD Failure Detector. xiii, xv, 1–7, 9, 11, 12, 15, 17–20, 22–33, 36–39, 41

FDR False Discovery Rate. 21, 38

FLP Fischer-Lynch-Paterson. 2

FN False Negative. xiii, xv, 21, 25–29, 34, 35, 37, 38

FOR False Omission Rate. 21, 29, 38

FP False Positive. xiii, xv, 12, 21, 25–29, 36–38

LSTM Long Short-Term Memory. xiii, xv, 1–4, 8–10, 15–17, 19, 25–27, 36–38, 41

ML Machine Learning. xiii, 1–4, 17–20, 22–25, 27–31, 34–36, 38, 39, 41

PD Precision Distribution. xiii, xv, 5, 12, 18, 23–25, 27, 28, 30, 33–35, 38, 41

PR Precision Recall. 17, 21, 22

QoS Quality of Service. 22, 27

ROC Receiver Operating Characteristic. 17, 21, 22, 28, 30, 38

RTC Round-Trip Count. 11, 12

SVM Support Vector Machine. xv, 17, 19, 27–30, 32, 33, 36–38, 41

TN True Negative. 21, 29, 38

TP True Positive. 12, 21, 29, 35, 38

TSH Time Since (last) Heard. 11, 12

TTF Timed and Time-Free. 7, 19, 23, 24

List of Figures

List of Tables

1

Introduction

Failure detectors (FDs) play a crucial role in identifying faulty nodes in distributed computing models. While analytical FDs can achieve remarkable accuracy, they usually need long detection times. To address this issue of prolonged detection times associated with analytical FDs, people integrate machine learning (ML) methods to balance the accuracy and the detection time. This thesis project primarily concentrates on ML-based FDs.

Section 1.1 provides an overview of fundamental concepts and challenges in distributed computing, as well as the significance of FDs. Following this, Section 1.2 and Section 1.3 briefly explain the previous work on FDs using ML classifiers and the Long Short-Term Memory (LSTM) model. We also describe our primary objective, which is to determine the suitable ML-based FDs for various scenarios. Our contributions are discussed in Section 1.4. Lastly, Section 1.5 outlines the structure of this entire thesis report.

1.1 Background

A distributed system is made up of a collection of distributed computing units, often referred to as entities. In the context of this thesis, we use a *process* or a *node* to represent an entity. These entities are interconnected by a communication medium (usually *channels*), constituting an abstraction of a physically distributed system. Entities may have *crash* failures and *Byzantine* failures [1]. Intuitively, When an entity stops its execution prematurely, we say it encounters the *crash* failure. When an entity does not follow the behavior assigned to it, we say it encounters the *Byzantine* failure. We only focus on crash failure in this thesis. A channel can be synchronous or asynchronous and reliable or unreliable. Synchronous means that there is an upper bound on message transfer delays, while asynchronous means there is no such bound. Reliable means there is no message loss, creation, modification, or duplication, while unreliable means there is no such guarantee.

Distributed computing focuses on a problem in terms of distributed entities with many parameters involved, but each entity has only a partial knowledge of these parameters. To solve such a problem, distributed entities exchange messages via the channels. The problem in the world of distributed computing has a set of properties such as *liveness* (sometimes called *termination*), *safety* (sometimes called *validity*, *agreement*), etc. The liveness property indicates the assurance that a problem yields a result, and the

safety properties establish safeguards ensuring that specific conditions must never be violated. We say that an algorithm solves a problem in the distributed computing model if, assuming the inputs are correct, any run of the algorithm in the distributed computing model satisfies all the properties of the problem. For example, consider the problem of reaching consensus [2] for several entities over the asynchronous and unreliable channels that have message loss problem. To simplify this problem, we assume that the involved entities do not have crash failure or Byzantine failure, and the messages sent over the channel might be lost, but no other errors. The safety properties of this problem state the correct outputs of an algorithm: all entities should agree, and in no cases two or more entities disagree. The liveness property states that all entities eventually and permanently enter into a done state.

Whether an algorithm exists to solve problems within the distributed computing model, such as the consensus problem, has become a question of interest to many. Fischer, Lynch, and Paterson delved into the consensus problem in a distributed computing model, where entities can only have crash failures and the channels are both asynchronous and reliable. They posited the *FLP impossibility* in [3], saying that it is impossible to design a deterministic consensus algorithm that can simultaneously guarantee safety, liveness, and fault tolerance in an asynchronous system. One prominent approach to circumvent the FLP impossibility is the introduction of the *failure detector* (FD) concept, as proposed by Chandra and Toueg in [4]. Functionally, a failure detector can be seen as an oracle made up of several modules, with each module being associated with a process (an entity). These modules offer hints and insights into the respective processes regarding the failures of other processes within the system. Failure detectors (FDs) are categorized into different classes, each class dedicated to the particular type of failure information they provide. The choice of a particular class of failure detector is contingent on the characters of different problems within the context of a fault-prone asynchronous distributed system model.

1.2 Problem Statement

Since Chandra and Toueg first formally defined *unreliable failure detectors* in [4], many analytical failure detectors that rely on analytical or mathematical models to make predictions have been developed. For instance, there is the $\diamondsuit P$ failure detector that is based on Blanchard et al. [5], the Ω failure detector based on algorithms by Aguilera et. al [6], as well as Mostefaoui, Mourgaya, and Raynal [7], or their combination, as described at [8], Chapter 18.9. While analytical FDs can achieve impressive accuracy, they usually need long detection times. Consequently, people integrate machine learning (ML) into the implementation of FDs to balance between detection time and accuracy. Häger and Köre studied how to use ML classifiers to identify the fault entities [9]. In their work, Häger and Köre compared ML-based FDs with an analytical FD developed by Blanchard *et al.* [5]. Furthermore, Li and Marin [10] proposed a timeout-based heartbeat failure detector that utilizes the LSTM model [11].

The results of Häger and Köre show that although FDs based on different ML classifiers consistently achieve an accuracy rating exceeding 0.999, along with a

specificity surpassing 0.999, they exhibit varying levels of accuracy. The best classifier among all is the Random Forest, achieving an accuracy of 0.99999 and a specificity of 0.9999. Meanwhile, the Random Forest classifier has the ability to reduce detection time by as much as 86.3% compared to the analytical solution proposed by Blanchard *et al.* [5], while maintaining an error rate of 1×10^{-6}. The result of Li and Marin also shows that the LSTM-based FD achieves much shorter detection times for a similar probability of availability than Chen's analytical solution [12], but at a significant computation cost.

In previous studies, the focus was mainly on comparing a single ML-based FD with a conventional analytical FD, without delving into a comprehensive comparison among various ML-based FDs. In this project, our main objective is to implement and evaluate FDs based on different machine-learning methods as proposed by Häger-Köre and Li-Marin. Our comparison is not just a comparison of different classification metrics values like specificity, accuracy, sensitivity, etc (refer to table 4.1). Instead, we also conduct a pair-wise comparison of different FDs on different data sets. On top of that, we construct ensemble FDs by combining top-performing FDs, aiming to find the most efficient FD tailored to different scenarios.

1.3 Motivation and Related Work

Earlier for this project, Häger and Köre [9] studied eight different kinds of statistical classifiers that aim at facilitating the implementation of $\Diamond P$. Specifically, Häger and Köre modeled failure detection as a binary classification problem and implemented FDs that use Machine Learning (ML) as the mechanism for failure detection. They employed several binary classifiers in the implementation of failure detectors to distinguish between correct or faulty entities and compared their implementation with the analytical FD by Blanchard *et al.* [5] (refer to as BDBD in this thesis) in terms of accuracy and detection time. Li and Marin [10] used a machine learning method different from classification to implement FDs. Their solution is to let entities send heartbeats to each other regularly, use the LSTM model to predict the arrival time of a heartbeat, and compare it with the actual arrival time to infer the status of an entity.

To our knowledge, the integration of ML methods into FD implementation is still in the initial stage, and comparisons of FDs based on various ML methods are even rarer. We hope our research will provide insights into selecting the most suitable ML-based FDs across diverse scenarios.

1.4 Contribution of This Work

This project continues the work conducted by Häger and Köre [9] in the field of FDs. Building upon their work, our contributions are as follows:

- Evaluate and compare the performance of FDs that are based on various ML classifiers. We address the data leakage issue in Häger and Köre's solution and

filter out FDs with obviously poor performance.

- Combine multiple machine learning models to create a combination of models for the FD that outperforms any individual model.
- Cluster FDs based on different ML classifiers during crash and non-crash intervals, particularly when these FDs report false negatives and false positives, respectively.
- Implement and evaluate an unreliable failure detector using LSTM [11]. We evaluate its accuracy (using the probability of availability as an indicator) and the computation time.

1.5 Disposition

This thesis comprises six chapters, each with its distinct focus and content. The structure of the thesis is as follows:

Chapter 1 - Introduction: In this chapter, we provide essential background information, outline the research's aims and objectives, and present a list of our contributions.

Chapter 2 - Methods: This chapter covers basic concepts related to Fault Detectors (FDs) and Machine Learning (ML) as well as the application of ML methods in FDs.

Chapter 3 - Implementation: In this chapter, we provide comprehensive details on the implementation aspects, including specifics about the LSTM-based unreliable FD and the cluster and comparison algorithm.

Chapter 4 - Evaluation: This chapter elucidates the experiments conducted and the configurations employed for evaluating FDs.

Chapter 5 - Results: The outcomes of our research are presented and discussed in this chapter, providing insights into our findings.

Chapter 6 - Conclusion: This final chapter encapsulates the discussions and conclusions drawn from our research, summarizing the key takeaways and contributions made.

Each chapter contributes to the overall narrative of the thesis, enabling a comprehensive understanding of the research and its outcomes.

2

Methods

This chapter provides an overview of concepts and methods used by failure detectors, along with insights into data point generation and PD modules developed by Häger and Köre. In Section 2.1, we present the formal definition of crash failures. Section 2.2 introduces the concepts of failure detectors (FDs), explores their properties of completeness and accuracy, and gives a definition of eventually perfect failure detectors based on specific completeness and accuracy. Within this section, we also introduce the heartbeat failure detector, a mechanism for detecting the status of a process by continuously sending heartbeat messages. Section 2.3 briefs two analytical FDs belonging to class $\diamondsuit P$.

Moving forward, Section 2.4 delves into the machine learning methods for both binary classification and regression tasks. Section 2.5 introduces three ensemble methods used to combine the results of multiple machine learning models.

Next, the procedure of generating data points and the PD module are described in Section 2.6 and Section 2.7, respectively.

Lastly, Section 2.8 introduces the concepts of data leakage.

2.1 Crash Failures

From the perspective of a failure detector, a process (an entity) is considered to be in one of two states: if a process terminates prematurely, it is called as being crashed or faulty, otherwise, it is considered to be correct or non-faulty. Chandra and Toueg utilized a discrete global clock, represented by the set of natural numbers, in the definition of the failure pattern. This failure pattern describes the processes that have crashed at any time. Formally, let $\mathcal{T}$ be the clock's ticks and let Π be the set of all processes in the system, then the failure pattern F is a function from $\mathcal{T}$ to 2^{Π}, where $F(t)$ denotes the set of crashed processes up to time t. We define $crashed(F) = \cup_{t \in \mathcal{T}} F(t)$ and $correct(F) = \Pi - crashed(F)$. Since once a process crashes, it does not “recover", we have $\forall t$: $F(t) \subseteq F(t+1)$.

2.2 Failure Detectors

Failure Detectors (FDs) are an abstraction in distributed computing that help systems identify faulty processes in a distributed environment. The FD in distributed systems

is a module attached to a process that provides information about which processes in the system have crashed. At any given time, a FD can either trust a process to be functioning correctly or suspect it to have crashed. The classes of FDs are differentiated by their properties of *completeness* and *accuracy* [4]. Completeness refers to the eventual suspicion of every process that has actually crashed, while accuracy refers to the number of correct decisions made by the FD regarding crashed processes.

2.2.1 Unreliable Failure Detectors

The unreliable failure detector abstraction was introduced by Chandra and Toueg in [4]. A failure detector is unreliable in the sense that it may either fail to identify a faulty process or falsely identify a correct process as faulty. The failure detector module maintains a list of processes that are suspected to be faulty, with this list being dynamically updated to reflect the addition or removal of processes as needed. However, it is noteworthy that the failure detection module can produce errors in its operation, such as mistakenly suspecting a running process as faulty or vice versa.

2.2.2 Eventually Perfect Failure Detectors

According to Chandra and Toueg, there are two kinds of completeness and four accuracy properties are defined, resulting in eight distinct classes of failure detectors. We are more interested in the eventually perfect failure detectors ($\Diamond P$), which is defined as follows:

Definition 2.2.1 (Eventually Perfect Failure Detectors) *A failure detector provides each process p_i a local variable, a set denoted $trusted(i)$. Let $trusted(i,t)$ be the value of $trusted(i)$ at time t. Remember that $correct(F)$ denotes the set of non-faulty processes in a run, and $crashed(F)$ denotes the set of faulty processes. With these denotations, the eventually perfect failure detector is defined by the following properties:*

- ***Completeness:*** $\forall F$, $\exists t \in \mathcal{T}$, $\forall i \in crashed(F)$, $\forall j \in correct(F)$, $\forall t^{'} \geq t$: $i \notin trusted(j, t^{'})$
- ***Accuracy:*** $\forall F$, $\exists t \in \mathcal{T}$, $\forall i, j \in \Pi - F(t^{'})$, $\forall t^{'} \geq t$: $i \in trusted(j, t^{'})$

Intuitively, completeness means that every faulty process is suspected by every correct process after a certain period of time, and accuracy means that no non-faulty process is suspected after a certain period of time since the last crash occurs.

2.2.3 Heartbeat Failure Detector

A heartbeat failure detector is a type of failure detector in distributed systems that utilizes the heartbeat mechanism to determine the liveness of network nodes. The timeout-based heartbeat failure detector [13] is achieved by periodically sending heartbeat messages. These heartbeat messages are typically sent continuously from the time of the originator's startup until its shutdown. If the monitoring process does

not receive a heartbeat message within a specified number of intervals, it assumes that the process that should have sent the message has failed, or crashed.

The time-free heartbeat failure detector [14] simply keeps track of the total number of heartbeat messages received from each counterpart process. The monitoring process outputs a vector of counters, one for each monitored process. In the event that a monitored process does not crash, its counter will continue to increase without bounds. Conversely, if a monitored process crashes, its counter will stop increasing.

2.3 Analytical Failure Detectors

Häger and Köre [9] introduced two analytical FDs belonging to class $\Diamond P$: BDBD and TTF.

2.3.1 BDBD

The BDBD FD, inspired by Blanchard *et al.* [5], is named after its authors: Blanchard, Dolev, Beauquier, and Delaët. This FD suspects a process by counting the number of round-trip messages. Suppose the process $p_i \in \mathcal{P}$ uses BDBD FD to detect the status of the process $p_j \in \mathcal{P} \backslash \{p_i\}$, it records the number of completed round-trips between p_i and other processes $p_k \in \mathcal{P} \backslash \{p_i, p_j\}$ since the last completed round-trip between p_i and p_j. If p_j falls behind other processes a predefined number of round-trips, then it is suspected by p_i.

To use the BDBD FD, we compute the value of the predefined number of round-trips that is needed to reach a desired specificity level for the training set. A certain value of this specificity is a threshold, referred to as θ parameter.

2.3.2 TTF

The TTF (Timed and Time-Free) FD extends the BDBD FD by introducing a timer that tracks the time elapsed since the last completed round-trip between p_i and p_j. That is to say, if p_j falls behind other processes a predefined number of round-trips and the duration since the last completed round-trip between p_i and p_j exceeds a predefined threshold, p_i considers p_j as suspected.

The TTF FD incorporates both round trips and a timer. To achieve a desired specificity level for the training data using TTF, we must establish appropriate values for both the round-trips and the timer. Similar to BDBD, a specific threshold value is defined for this specificity, referred to as the 2D probability threshold.

2.4 Machine Learning

In the context of this project, we employ machine learning models to perform classification and regression. Classification aims to assign an input vector $\mathbf{x}$ to one of K discrete classes denoted as C_k, where $k = 1, 2, \ldots, K$. Analogously, the purpose of regression is to predict one or more continuous target variables $\mathbf{t}$ based on the

values of a D-dimensional input vector $\mathbf{x}$. For classification, please refer to Section 2.4.1, which provides a list of the classifiers used in this project. Additionally, Section 2.4.2 gives a brief introduction to the application of the Long Short-Term Memory (LSTM) model [11] for regression.

2.4.1 Binary Classifiers

For the purpose of our project, we employ a range of binary classifiers in the implementation of failure detectors to distinguish between correct or faulty nodes. The classifiers are trained using an algorithm that presents it with input data or "data points", along with a label indicating what the data represents. We will study multiple off-the-shelf classifiers rather than focus on optimizing a few. The machine learning classifiers that are used in our work are:

- **Decision Tree**: a classifier that reaches its decision through the use of a tree, which is strategically constructed upon training [15].
- **Random Forest**: a classifier that constructs multiple Decision Trees during training, and then makes its decision based on the trees' majority vote [15].
- **AdaBoost**: a classifier, which is a meta-estimator, is employed by first fitting the classifier on the original data set and then fitting additional copies on the same data set, but with the weights of the misclassified instances adjusted. This process helps to direct subsequent classifiers to focus more on challenging cases [16].
- **Logistic Regression**: a classifier that involves fitting a logistic function to a data set through the use of Stochastic Gradient Descent [17], [18]. The output is bound between 0 and 1 and represents the predicted probability of a given data point belonging to one of the two predefined classes [15].
- **Naive Bayes**: a classifier that is based on the assumption of independence between the parameters of a data point. Using Bayes's Theorem, the classifier determines the most probable class for a new data point, based on observations obtained during training [19].
- **Support Vector Machine(SVM)**: a classifier that maps the input vectors of two classes, representing the data points, into a multi-dimensional space and creates a decision boundary plane to distinguish between the two classes [20].
- **Artificial Neural Network(ANN)**: a classifier that utilizes the principles of a Neural Network, in which the input vector is processed through the network, a classification can be made by looking at the values of the neurons in the output layer [15].
- **Long Short-Term Memory(LSTM)**: The utilization of an LSTM model as a binary classifier involves the implementation of a single neuron in the output layer, equipped with a sigmoid activation function. The sigmoid function transforms the input into a continuous value between 0 and 1, which can be interpreted as a probability of the positive class.

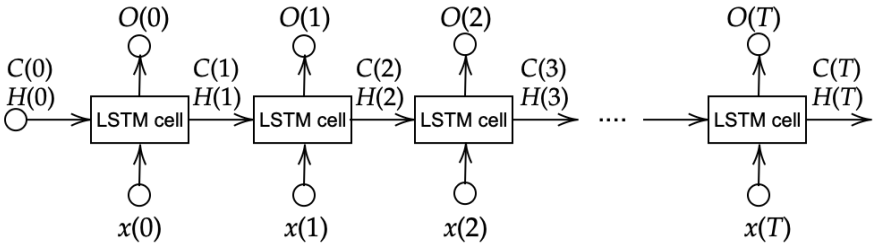


Figure 2.1: The architecture of the LSTM model.

- **FedDyn**: FedDyn model [21] can be trained on data that is distributed across multiple devices. In the context of binary classification, the FedDyn model can be configured with a single neuron in the output layer, equipped with a sigmoid activation function. This configuration is similar to that of the Long Short-Term Memory (LSTM) model, which has been previously proposed for binary classification tasks.

2.4.2 LSTM for Regression

Now we are going to discuss the application of LSTM for regression. As showed in Figure 2.1, the LSTM model takes the sequential data $x(i)$ as input, generates $O(i)$ as the outputs, with $i \in \{0, 1, \ldots, T\}$. $C(i)$ and $H(i)$ are the cell states and the hidden states respectively, and their purpose is to transfer information between step $i-1$ and step i.

The LSTM's capability to easily capture the data patterns is mainly attributed to its special cell structure. Figure 2.2 shows the structure of the LSTM cell. This cell consists of three gates: the forget gate, the input gate, and the output gate. The forget gate determines whether information from the preceding step should be preserved or discarded as irrelevant. Consequently, only the relevant preceding data points in the sequential data set are kept, and the irrelevant preceding data points are discarded. Meanwhile, the input gate tries to learn new information from the input to the cell. Finally, the output gate conveys the updated information from the current step to the subsequent one ($C(i)$ and $H(i)$) and generates the output ($O(i) = H(i)$).

In our application of the LSTM model to the heartbeat FD in Section 2.2.3, we input a sequence of consecutive heartbeat message arrival times into the LSTM model. Then this model outputs the prediction of the arrival time of the next heartbeat message. To clarify, suppose we have a set of consecutive heartbeat messages m_i, where $i \in \{1, 2, \ldots, N\}$, each associated with the arrival time t_i. After feeding these arrival times (t_i) into the LSTM model, it outputs O_i, which are the predicted arrival times of the subsequent heartbeat messages (m_{i+1}).

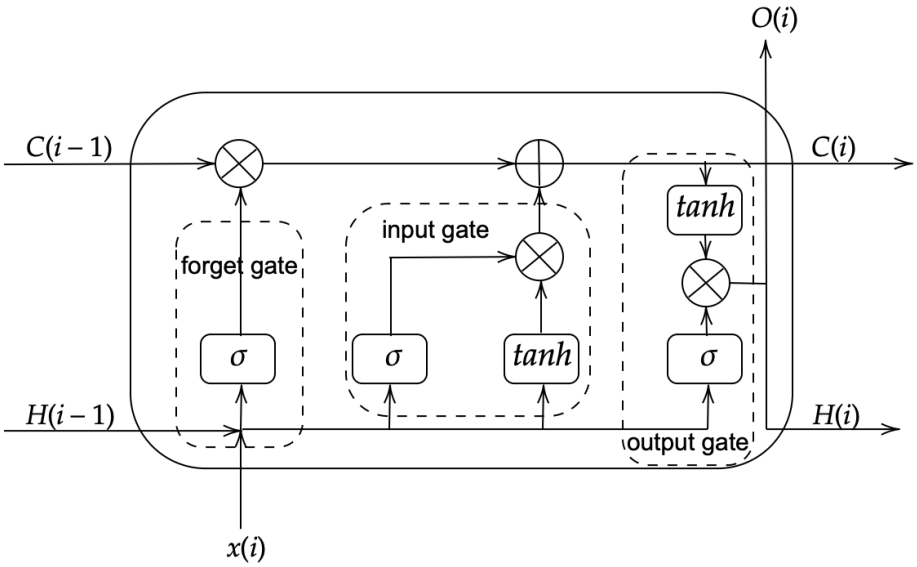


Figure 2.2: The structure of the LSTM cell.

2.5 Ensemble Methods

The ensemble method combines the predictions of multiple machine learning models to create a more robust and accurate model. By aggregating the predictions of multiple machine learning models, the idea is that the weaknesses of some models can be compensated by the strengths of other models.

In this project, we will employ three ensemble methods. The first one is an unweighted majority voting algorithm, in which multiple machine learning models are trained independently on the same data set. When it predicts a new data point, each model gives its own classification prediction (vote). This algorithm counts the number of votes for each class and chooses the class with the highest count as the final predicted class. We call it ensemble.

The second method is Stacking. This method is also known as a stacked generalization, a two-layer model. The first layer involves multiple models (base models) that are trained to make predictions, and the second layer is the meta-model, which is trained on the combined predictions of base models. Each base model may capture different aspects of the data set, and the meta-model can correct or weigh the predictions of individual base models, leading to an improved overall performance. Figure 2.3 shows the overall architecture of the stacking method.

The third method is Blending, which is similar to stacking. The main difference is the training of the meta-model is applied on a separate holdout set instead of the full data set. Specifically, the training data set is partitioned into base models training and holdout sets. Base models are trained using the base models training set and subsequently make predictions on the holdout set. Then, the meta-model is trained on predictions from the holdout set. In general, the blending method is believed to be simpler and quicker than the stacking method. In the following description, we

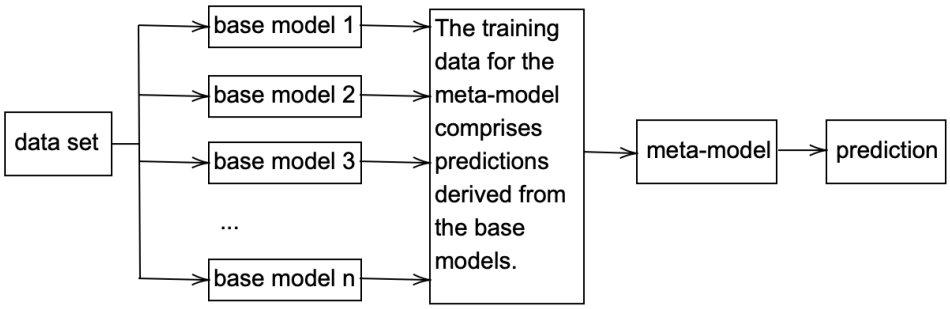


Figure 2.3: The architecture of stacking.

call it BlendStacking.

2.6 Data Points Generation

Häger and Köre [9] proposed a parameter module to derive parameters from the message-passing system log data. These parameters are categorized as timed and time-freed and extend the timed and time-freed windows. Subsequently, they developed a simulator to generate a set of data points from these timed and time-free windows to train the machine learning models.

2.6.1 Parameter Module

The proposed parameter module represents an approach for transforming message-passing system data into parameters suitable for training machine learning models for failure detection. In addition to leveraging the parameter module, the machine learning-based failure detectors also incorporate an analytical solution as a fallback mechanism. This hybrid approach guarantees that in the event of a node crash, it will eventually be suspected, regardless of the output produced by the machine learning classifier.

2.6.2 Timed and Time-free Windows

The training data set for the machine learning models are generated from timed and time-free parameters. The timed parameters are computed through the utilization of clocks or timers, such as Time Since last Heard (TSH). While time-free parameters, which are also known as round-trip-based parameters, are derived from the number of round trips completed in the system, without relying on clocks or timers. An example of a time-free parameter is the Round-Trip Count (RTC). This solution requires that the FD is queried at a fixed time interval. During each query, the FD captures and records the system state, resulting in a timed sample. A fixed number of consecutive samples form a timed window. Figure 2.4 gives an example of a timed window. Similarly, a time-free window includes a sequence of a certain number of complete round-trip counters. For each node $p_j \in P \backslash \{p_i\}$ that a node p_i

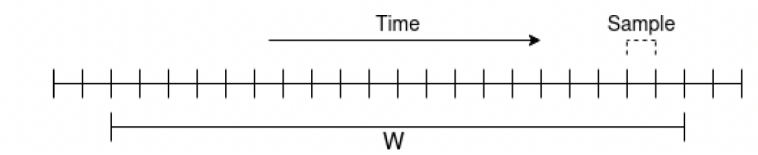


Figure 2.4: A window W with a size of 20 samples, note that the system does not know about the samples outside of W.

is communicating with, the FD keeps a count of how many round-trips other nodes $p_k \in P \backslash \{p_i, p_j\}$ have completed with p_i since p_j last completed a round-trip with p_i.

2.6.3 Generating Data Points

The data points used to train machine learning classifiers are generated from the values of the timed windows described in Section 2.6.2. In detail, the failure detector is queried at a fixed time interval, say, δ seconds, a data point is created based on the values of the timed windows every δ seconds. The population of the values of the windows is performed by iterating a log file in a chronological order, a method described in Section 2.6.2.

2.7 Precision-Distribution Module

Precision-Distribution (PD) module was developed by Häger and Köre [9] to reduce the probability of falsely suspecting a non-faulty node. The PD module takes the form of a 2D-Histogram, which has two dimensions: RTC and TSH. This histogram is populated by having a classifier predict classes for the training set and recording frequencies of suspicions that were either FPs or TPs based on the RTC and TSH values associated with each bucket. Each bucket effectively represents precision $(TP/(TP + FP))$, signifying the probability of a suspicion being correct. Buckets with no TP or FP entries during training are conservatively assigned a precision value of 1.0. To use the PD module, a threshold (referred to as a histogram threshold) is set to determine when a raised suspicion is deemed valid; if the precision falls below this threshold, the suspicion is ignored.

2.8 Data Leakage

The phenomenon of data leakage [22] occurs when information from sources outside of the training dataset is utilized in the training of a machine-learning model. It is a source for poor generalization and overestimation of expected performance, as this extraneous information can lead the model to acquire knowledge that it would not have in a real-world production setting, resulting in overly optimistic models or the invalidation of the model's estimated performance.

Häger and Köre [9] pointed out that a potential issue related to data leakage exists in the generation of training dataset for the machine learning classifiers. To address this

issue, we plan to rectify the data leakage problem, regenerate the training dataset, conduct the experiments and collect the corresponding results.

3

Implementation

We present the implementation in this chapter. In Section 3.1, we explained the Li-Marin LSTM-based FD and our implementation. Next, the implementation of FDs based on binary classifiers and ensemble methods are introduced in Section 3.2 and Section 3.3.

3.1 The LSTM-based Unreliable Failure Detector by Li and Marin

Li and Marin [10] proposed a timeout-based heartbeat failure detector that utilizes the LSTM model [11] to predict the arrival time of next heartbeat message.

3.1.1 Li-Marin Solution Overview

The solution is centered around the use of regular heartbeat messages transmitted by nodes in a distributed system. A group of nodes (monitored nodes) generate a heartbeat message h containing the sender ID and a sequence number and send it to a monitoring node every Δ second. Upon reception of a heartbeat, the monitoring node records its arrival time in the log associated with the sender. This process is illustrated in Figure 3.1.

To determine the status of monitored node p (trusted or suspicious), the monitoring

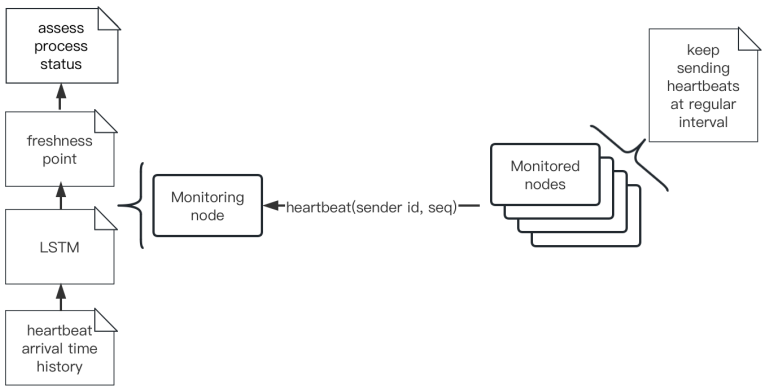


Figure 3.1: Monitored nodes keep sending heartbeats to the monitoring node.

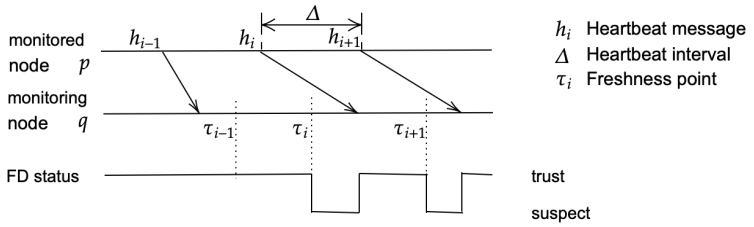


Figure 3.2: Estimating the arrival time of the next heartbeat to suspect failures.

node q calculates an expected arrival time for the next heartbeat τ (referred to as the *freshness point*). If the heartbeat message is received by monitoring node q before the estimated time, then monitored node p is considered to be trusted. Otherwise, if the heartbeat message is not received by the estimated time, q suspects p of failure. This procedure is illustrated in Figure (3.2). With the objective of skewing the output of the failure detector towards over-estimations of the next arrival time, Li and Marin chose to employ a unilateral penalty loss function instead of the commonly used mean squared loss function.

In their effort to achieve online predictions, Li and Marin limited the size of the training dataset, as the computational time of the Long Short-Term Memory (LSTM) model [11] needed to be smaller than the expected arrival time of the heartbeats. To fulfill this requirement, they fed the LSTM model with the nearest fixed size of data (500). Although this limited dataset size also meant that data that falls out of the range was discarded, its impact on the parameters of the LSTM model persists.

Additionally, Li and Marin have incorporated a dynamic safety margin into their LSTM based failure detector. This safety margin allows the model to accommodate unexpected and variable delays, as it utilizes the ϵ most recent errors to adjust in time. The freshness point (τ) is calculated as per the following equation:

$$\tau \approx LSTM.predict + \frac{1}{\epsilon} \sum_{i=k-\epsilon}^{k} error_i \tag{3.1}$$

3.1.2 Our Implementation with Simulative Approaches

We use PyTorch [23] to implement the LSTM model as described in Section 2.4.2. The input sequential data includes the arrival times of previous heartbeat messages, while the output predicts the arrival time of the next heartbeat message. In our implementation of the unilateral penalty loss function, we introduce a parameter to adjust the degree of overestimation, aiming to reach a balance between errors in identifying faulty nodes as non-faulty and vice versa.

Unlike Li and Marin who assessed their approach on top of real traces: heartbeat transmission logs on PlanetLab.eu [24], we run and evaluate our solution on the data collected by Häger and Köre in [9]. Häger and Köre logged events and wrote them to files, and each event is attached with a timestamp of when it occurred. We choose one type of event that happened periodically and extract its timestamp for the simulation of the arrival time of a heartbeat message.

3.1.3 Advantages and Disadvantages Analysis

The comparison between the heartbeat failure detector proposed by Chen, Toueg, and Aguilera in [25] and the machine learning-based heartbeat failure detector reveals that the latter exhibits a significant improvement in terms of detection time while maintaining a comparable level of probability of availability. The probability of availability is defined as the ratio of safe predictions to the total number of predictions.

However, it is important to recognize the limitations of the solution proposed by Li and Martin in [10]. Their approach, which is based on fixed interval heartbeat messages, may not be sufficiently robust due to potential network delays and the lack of guarantee of regular message transmission in multi-tasking environments. Although the model may function optimally when the network conditions are favorable and the sender process p is not hindered by other processes, it may not produce accurate results in less ideal circumstances.

3.2 ML-based Failure Detectors

The ML-based FDs in this thesis refer to the FDs based on the binary classifiers in Section 2.4.1.

3.2.1 ML Models Construction

We implemented our binary classifiers, including *Decision Tree*, *Random Forest*, *AdaBoost*, *Logistic Regression*, *Naive Bayes*, and *SVM*,utilizing the models provided in Scikit-Learn. We customized their hyperparameters, deviating from the default values. The other models *ANN*, *LSTM* and *FedDyn* were implemented using PyTorch models.

To improve the stability, performance, and interpretability of classifier models, we used the StandardScaler from Scikit-Learn to standardize features in the generated data points, as detailed in Section 2.6, specifically for Scikit-Learn classifiers.

Furthermore, we devised one classifier optimized for AUC PR (Precision-Recall) and another for AUC ROC (Receiver Operating Characteristic) for each Scikit-Learn classifier. Initially, we partitioned the training set into 80% for training and 20% for validation. Both optimized classifiers were then trained on the training subset and evaluated on the validation set. Subsequently, the evaluation results of the validation set were used to generate both PR and ROC curves. The decision threshold from the PR curve aimed to minimize the gap between precision and recall, whereas from the ROC curve, the threshold aimed to maximize the difference between sensitivity and fall-out. Finally, these thresholds were applied to the corresponding classifiers trained on the entire training set.

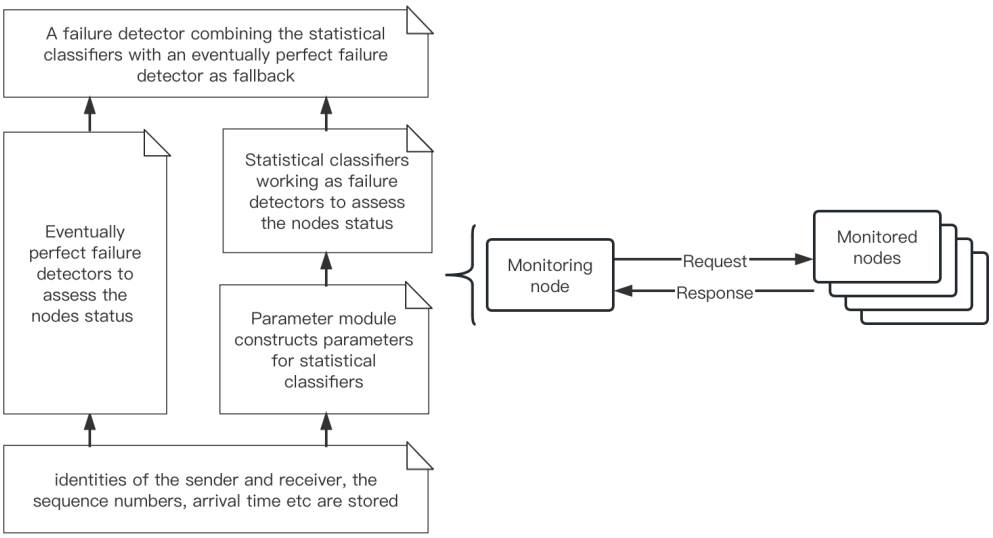


Figure 3.3: A failure detector combines the ML-based FD with analytical fallback.

3.2.2 ML-based FDs with Analytical Fallback

Note that ML-based FDs cannot guarantee FD completeness. To address this problem, we use the analytical FDs that are of class $\Diamond P$ as a fallback for all ML-based FDs, which means that suspicions made by ML-based FDs are overridden by the fallback. The BDBD FD falls under class $\Diamond P$, as it can ensure the properties of FD completeness and accuracy. Therefore, we use the BDBD solution as a fallback for all ML-based FDs. Additionally, The PD module can help ML-based FDs reduce the probability of falsely suspecting non-faulty nodes by using a threshold, preventing FDs from raising suspicions when the precision is low. With these considerations, our performance evaluation encompasses varying levels of mistake rates with PD enabled and BDBD FD used as a fallback. The structure of the ML-based FD with analytical fallback is shown in Figure 3.3.

3.3 Ensemble Failure Detectors

As outlined in Section 2.5, we introduced three ensemble methods, leveraging the StackingClassifier provided by Scikit-Learn for their implementation. Notably, the outputs of these ensemble methods are contingent upon the outputs of ML-based FDs. Considering the influence of the PD module on the outputs of ML-based FDs, it is evident that the outputs of the ensemble methods are likewise affected by the PD module. Consequently, there is no need for a separate PD module specifically tailored for the ensemble methods.

4

Evaluation

This chapter presents our empiric evaluation plan.

Units Under Evaluation. We assess the ability of ML-based classifiers, including *SVM*, *Naive Bayes*,*AdaBoost*, *Neural Network*, *Logistics Regression*, *Random Forest* and *Decision Tree*, to solve the problem of failure detectors (FDs). We refer to these FDs as individual ML-based FDs. Additionally, We consider two non-ML-based FDs, named BDBD and TTF, as described in Section 2.3. We combine these individual FDs using ensemble techniques that are called unweighted majority voting, stacking, and blending (refer to Section 2.5).

Chapter Organization. We start by describing the evaluation environment, which includes collected real data that is enriched using stimulative approaches. Then, we state the evaluation criteria, which is based on prevailing ML metrics. We describe different methods for examining the structural properties of the studied data and our ML-based FDs using clustering and ranking. Considering the unique nature of the LSTM FD by Li and Marin [10], an extra metric, named Probability of Availability, is used for its evaluation.

4.1 Evaluation Environment

In alignment with Häger and Köre, we conducted our evaluation using simulative approaches, executing both "time cost" and "accuracy" suites in our experiment plan.

As the nodes on PlanetLab.eu exhibit considerable stability and are unlikely to crash, we opt to simulate the crashes through the use of synthetic crash failures via the simulator developed by Häger and Köre. This simulator is used to "play back" the collected logs and create the data points employed in training and evaluating the proposed failure detectors. Given that these data points are produced through simulated rather than actual failures, this approach is referred to as simulative as it employs both collected and simulated data. In cases where the collected data contains failures, these data will be used; if no data contains failures, the simulated data will be employed.

During the "play back" of the collected logs, any messages received from a node during a period in which the node is synthetically crashed (muted) are ignored by the broadcasting algorithm. Consequently, the broadcaster encounters a lack of

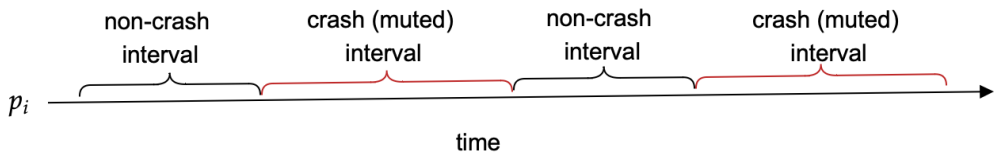


Figure 4.1: The chronological order of the "crash" and "non-crash" intervals.

response from the currently "crashed" node. Specifically, a crash interval refers to a specific time span during which the broadcaster encounters the absence of responses from the presumed "crashed" node. Conversely, a crash-free interval signifies a time span during which the broadcaster can reliably receive messages from the node as expected. Note that within the context of this study, these crash intervals and crash-free intervals occur in an alternating manner throughout the observation duration. Each crash or non-crash interval is called a period in the subsequent description. Figure 4.1 shows the chronological order of these intervals for a node p_i.

4.2 Evaluation Criteria

We evaluate the individual ML-based FDs and ensemble FDs from two perspectives. The first aspect involves examining the overall probability of raising a false suspicion, which we refer to $\Pr(FS)$. An FD with lower $\Pr(FS)$ is considered more favorable than one with higher $\Pr(FS)$. The second aspect is to investigate the relationship between $\Pr(FS)$ and the detection time for all FDs. An FD that achieves a faster detection time may involve implementing simpler algorithms or processing techniques that prioritize speed over thorough analysis. This could lead to a higher $\Pr(FS)$. Conversely, an FD that focuses on improving $\Pr(FS)$ may require more complex algorithms or extensive data analysis, which could increase the time needed for detection. This may result in a slower detection time as the detector thoroughly evaluates the data to minimize errors. A preferable FD is one that balances this trade-off, either by having shorter detection times while maintaining $\Pr(FS)$, or by decreasing $\Pr(FS)$ without incurring longer detection time.

4.2.1 Classification Metrics

We employ classification metrics, which are described below, to evaluate the $\Pr(FS)$ of our individual ML-based detectors and the ensemble FDs. The machine learning classifiers used in our failure detectors will be evaluated based on metrics derived from a confusion matrix. A confusion matrix is a table that is used to define the performance of a classification solution, which is shown in Figure 4.2. The derived metrics from the confusion matrix include Accuracy, Sensitivity, Specificity, Precision, Fall-Out, False discovery rate, and False omission rate. The definitions of these metrics are in Table 4.1.

Binary classifiers use the discrimination threshold to classify a data point. The discrimination threshold is the probability or score at which the positive class is chosen over the negative class. We can gain further insights into the performance

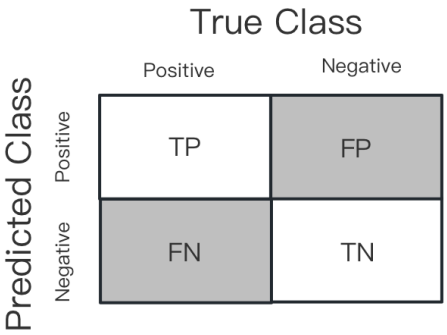


Figure 4.2: A confusion matrix.

Metric Name	**Derivation**	**Description**
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$	The ratio of data points that were correctly classified.
Sensitivity	$\frac{TP}{TP+FN}$	The ratio of positive data points that were correctly classified.
Specificity	$\frac{TN}{TN+FP}$	The ratio of negative data points that were correctly classified.
Precision	$\frac{TP}{TP+FP}$	The ratio of positive classifications that turned out to be correct.
Fall-Out	$\frac{FP}{FP+TN}$	The ratio of negative data points that were incorrectly classified. Can also be written as 1-Specificity.
False Discovery Rate (FDR)	$\frac{FP}{FP+TP}$	The ratio of the number of false positive classifications (false discoveries) to the total number of positive classifications (rejections of the null).
False Omission Rate (FOR)	$\frac{FN}{FN+TN}$	The ratio of actual positive instances that are incorrectly predicted as negative.

Table 4.1: Metrics derived from a confusion matrix.

of classifiers by combining the derived metrics in Table 4.1 and this discrimination threshold.

Our study focuses on two methods for the present approach, specifically the analysis of the Receiver Operating Characteristic (ROC) curve [26], and the Precision-Recall (PR) curve [27]. We acknowledge that Recall is synonymous with Sensitivity. Both curves are generated by adjusting the threshold from 0 (predicting all as positive) to 1 (predicting all as negative). In the ROC curve, *Sensitivity* is plotted on the y-axis and Fall-Out on the x-axis. The PR curve, on the other hand, employs *Precision* on its y-axis, and *Recall* on its x-axis. Once these curves are established, a threshold can be selected from them in order to optimize for desired metrics. We signify that

a classifier has been optimized in this way by appending its name with either PR or ROC.

After the curves are constructed, one can determine the Area Under Curve (AUC), which ranges from 0 to 1. For AUC ROC, a higher score signifies that the classifier can efficiently differentiate between the two classes. The AUC PR gauges the classifier's ability to accurately classify the positive data points without considering accurately classified negative data points. Nonetheless, we use only the AUC ROC as an evaluation metric, rather than the AUC PR, as it is more commonly employed.

4.2.2 Time and Accuracy Metrics

Similar to the FD Quality of Service (QoS) metrics defined by Chen *et al.* [25], we establish the time and accuracy metrics for evaluating FDs. These metrics are derived from the FDs' outputs. Specifically, the time metrics measure the duration during which the FD either trusts or suspects a node. The duration is measured using S- and T-transitions, where an S-transition signifies a transition from trust to suspicion, and a T-transition indicates a transition from suspicion to trust. Note that the First Detection Time (T_{D1}) is introduced by Häger and Köre in [9]. The detailed time metrics is shown below:

- **Time Metrics**
  - **Detection Time (T_D):** The duration from when a node crashes to when it is permanently suspected, denoting the time until the last S-transition, after which no further T-transitions occur.
  - **First Suspected Time (T_{D1}):** The duration from when a node crashes to when it is initially suspected, representing the time until the first S-transition takes place.

The accuracy metrics indicate how effectively an FD avoids making mistakes. They are shown below:

- **Accuracy Metrics**
  - **Average Mistake Rate (λ_M):** The probability of making a mistake per time unit, representing the average number of S-transitions per time unit.
  - **Query Accuracy Probability (P_{QA}):** The probability that the FD's output is correct at any given time.

We use two main metrics, Detection Time (T_D) and Average Mistake Rate (λ_M), to evaluate the performance of the ML-based FDs and investigate whether they can achieve a better balance between the detection time and the probability of raising a false suspicion. To clarify, we use T_D to represent the detection time in the trade-off, as it indicates the convergence of the failure detectors' suspicions, and λ_M to approximate $\Pr(FS)$, which represents the probability of raising a false suspicion. We choose λ_M as it assumes that making a mistake leads to a state transfer, and the duration of the error is irrelevant. In our system, each FD is queried at most once every 500ms, and we take this as the unit of time for λ_M.

4.3 Experiment Plan

4.3.1 Experiment suites

We set two experiment suites to evaluate the time and accuracy metrics. The detection time experiment suite, denoted by E_T, is conducted when one node, p_e, fails, *i.e.,*, eventually crashes. In this case, T_D is computed by measuring the period from the time of the crash to the last S-transition (Section 4.2.2), which is the point where no further T-transitions occur thereafter.

The accuracy experiment suite, denoted by E_A, is executed during periods where p_e does not fail. To evaluate the accuracy metrics, the number of mistakes made by the FD during these periods is counted. Any suspicion made for p_e in these periods is a mistake, and by definition, λ_M is the number of S-transitions per time unit.

As mentioned in Section 4.2, there is a trade-off between detection time and accuracy for an FD. We explore the performance of FDs by considering this trade-off between the time and accuracy metrics, following the approach of Chen *et al.* [25] and Häger-Köre [9].

4.3.2 Experiment thresholds

We demonstrate the properties of the trade-off for each FD, by executing the experiment in E_T and E_A with three types of thresholds. These thresholds include the θ parameter for the BDBD solution (Section 2.3.1), the 2D probability threshold for the TTF solution (Section 2.3.2), and the histogram threshold for the PD module (Section 2.7). We run both E_A and E_T experiments for each set of thresholds in the assessment. This approach facilitates establishing a correlation between specific detection time and its corresponding accuracy. Our exploration is based on the assumption that increasing the values of thresholds will result in a decrease in the Average Mistake Rate (λ_M) and an increase in Detection Time (T_D).

Exploring a wide range of threshold values provides a deeper comprehension of the trade-off between the time and accuracy metrics for FDs. However, due to the time-consuming nature of running E_T and E_A experiments, we select a limited set of threshold values, which is shown in Table 4.2.

These threshold values are chosen after analyzing the levels of Specificity that ML-based FDs achieve within the training set. For instance, to achieve a Specificity of five nines for the Random Forest FD with the PD module, the histogram threshold in the PD module (Section 2.7) should be set to 0.99. While achieving a Specificity of six nines, a threshold of 0.99998 is required. It's important to note that the threshold value required for each Specificity level may vary across ML-based FDs, as it depends on the performance of each FD on the training set.

0.990	0.9990	0.99990	0.999990	0.9999990
0.991	0.9991	0.99991	0.999991	0.9999991
0.992	0.9992	0.99992	0.999992	0.9999992
0.993	0.9993	0.99993	0.999993	0.9999993
0.994	0.9994	0.99994	0.999994	0.9999994
0.995	0.9995	0.99995	0.999995	0.9999995
0.996	0.9996	0.99996	0.999996	0.9999996
0.997	0.9997	0.99997	0.999997	0.9999997
0.998	0.9998	0.99998	0.999998	0.9999998

Table 4.2: Various values of thresholds

4.3.3 Experiment configurations

We run the experiment suites, E_T and E_A, with various threshold values under three experiment configurations: $C_{analytical}$, C_{PD} and $C_{fallback}$.

The $C_{analytical}$ configuration is to establish a baseline for the time and accuracy metrics of analytical FDs (the BDBD FD and the TTF FD) at varying thresholds, offering a benchmark and a basis for comparison with ML-based FDs through the BDBD solution.

The C_{PD} configuration is to analyze the time and accuracy metrics for ML-based FDs with the PD module. We aim to decrease the probability of falsely suspecting a non-faulty node for ML-based FDs by leveraging the PD module. In this configuration, we expect the ML-based FDs to achieve a shorter detection time (T_D) while maintaining the same level of average mistake rate (λ_M) as analytical FDs. Besides, an increase in the PD module threshold is expected to lead to a decline in T_D and an improvement in λ_M for FDs.

The $C_{fallback}$ configuration enables us to evaluate a solution wherein ML-based FDs not only incorporate the PD module but also integrate the BDBD solution as a fallback, ensuring the eventual detection of crashed nodes regardless of the ML classifier output. BDBD solution working as a fallback means that the suspicion made by BDBD will overrule the output of the ML-based FDs. This configuration includes three sub-configurations: $C_{fallback,higher}$, $C_{fallback,equal}$ and $C_{fallback,lower}$ based on whether the threshold value of BDBD is higher, equal to or lower than the threshold value of PD module. Specifically, when a PD module threshold is set to achieve a specificity of X number of 9s in the train set, for $C_{fallback,higher}$, $C_{fallback,equal}$ and $C_{fallback,lower}$, the threshold of BDBD is set based on a specificity level of $X + 1$ number of 9s, X number of 9s and $X - 1$ number of 9s, respectively. Our expectation for $C_{fallback}$ is that BDBD solution may have more influence on the final output in the $C_{fallback,lower}$ sub-configuration as BDBD overrules the output of the ML-based FDs once its threshold is met. If the T_D of BDBD is longer than that of ML-based FDs, as is likely, the BDBD solution's impact on the final results will be negligible in $C_{fallback,higher}$ and $C_{fallback,equal}$. However, if the T_D of BDBD is shorter than that of ML-based FDs, the BDBD solution will impact the final results in all configurations of $C_{fallback}$, with $C_{fallback,lower}$ being the one with the greatest impact.

The permissible threshold values for the $C_{analytical}$ and C_{PD} configurations are the values listed in the second to fifth columns of Table 4.2.

In each run of E_T and E_A within the $C_{fallback}$ configuration, one threshold for BDBD and one threshold for the PD module are required. The permissible combinations of threshold values are the adjacent values in adjacent columns in Table 4.2. For example, in the sub-configuration $C_{fallback,higher}$, potential threshold values for the PD module and BDBD include {0.990, 0.9990}, {0.991, 0.9991} and so forth. While in the sub-configuration $C_{fallback,lower}$, the feasible threshold values for PD module and BDBD are {0.9990, 0.990}, {0.9991, 0.991}, and similar combinations. For the sub-configuration $C_{fallback,equal}$, the possible threshold values for PD module and BDBD are {0.9990, 0.9990}, {0.9991, 0.9991}, and onwards.

4.4 Clustering and Ranking ML-based FDs

To investigate the structural patterns in the input/output behaviors of the ML-based FDs proposed by Häger and Köre in [9], the values of False Positive (FP) and False Negative (FN) of each ML classifier were compiled for each data point. Specifically, we use the results of FP/FN to cluster these ML classifiers using K-means algorithm [28] at the crash intervals where any ML classifiers report FN and the crash-free intervals where any ML classifier reports FP.

We rank the ML-based FDs in terms of specificity and sensitivity to find the top-performing FDs. To facilitate further comparison and enable the selection of the best two or three FDs, a pairwise comparative analysis of the candidate FDs will be conducted using the following criteria. Let us consider the pairwise comparison between FDs A and B:

- A & B: both FDs A and B make correct predictions.
- ¬A & B: FD A makes an incorrect prediction while FD B makes a correct prediction.
- A & ¬B: FD A makes a correct prediction while FD B makes an incorrect prediction.
- ¬A & ¬B: both FDs A and B make incorrect predictions.

By examining and comparing these distinct scenarios, we aim to determine the most reliable FDs.

4.5 Li-Marin LSTM FD Evaluation

The Li-Marin LSTM FD [10] gives the status of a node by comparing the actual arrival time of the heartbeat message and the predicted time, which makes its evaluation different from the ML-based FDs by Häger and Köre. The Li-Marin LSTM FD may produce two error types: falsely identifying non-faulty nodes as faulty nodes (FP errors) and falsely identifying faulty nodes as non-faulty nodes (FN

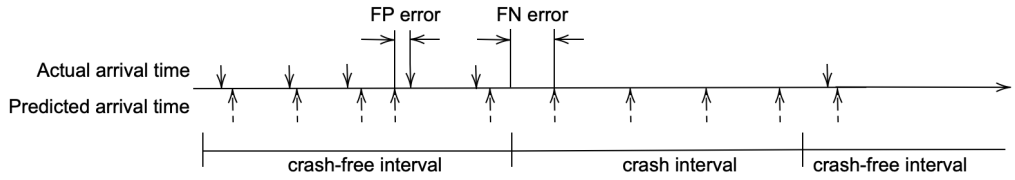


Figure 4.3: The places where FP/FN errors may occur in Li-Marin LSTM FD.

errors). The places where these two types of errors may occur are shown in Figure 4.3.

Recall that we want to control this FD in a manner that inclines it towards over-estimation of the anticipated arrival time of the heartbeat message. However, we should recognize that the extent of this overestimation impacts the occurrences of FP errors and FN errors. In the scenario where the overestimation is substantial, the FP errors diminish while the FN errors escalate. Conversely, if the overestimation is small, the FP errors increase while the FN errors decrease. Balancing these two types of errors is important for the effective functioning of the failure detector. To address this, we introduce a variable to control the extent of over-estimation.

In addition to the metrics that we have listed previously, we introduce a new metric called *Probability of Availability* (P_A) (following the approach of Li and Marin), which is the ratio of the safe predictions (actual arrival time is ahead of the predicted time) over the total predictions. This metric indicates the ability of this FD to avoid wrong suspects.

5

Results

Our results show in Section 5.1 that classifiers such as *Random Forest*, *SVM*, and *Logistic Regression* outperform other classifiers in general. We also compare, in Section 5.2, the QoS metrics of ML-based FDs with PD module and fallback to the BDBD solution (Section 2.3.1). This compartment shows that *Random Forest*, *SVM*, and *Logistic Regression* can balance well the trade-off between detection time and accuracy, but only when aiming at short detection time, ca 1 second. When the detection time is more than 2 seconds is considered, BDBD and TTF can offer better accuracy.

In section 5.3, we cluster the ML-based FDs at both the crash intervals where any ML-classifiers report FN and the data points where any FD reports FP. Based on these results, the FDs are ranked based on their specificity and sensitivity. Additionally, a pairwise comparison of the top-performing FDs is conducted toward the conclusion of this section. Further results of all clustering outputs for all FDs over various datasets are shown in Appendix A.

Subsequently, Section 5.4 presents the evaluation of the LSTM-based heartbeat FD, focusing on the probability of availability (P_A) and the classification metrics.

Finally, we explain our results and discuss their limitations in Section 5.5.

5.1 ML-based FDs without PD Module

We present the results of ML-based binary classification FDs without the use of the Precision Distribution (PD) module (refer to Section 2.7). Our results were collected after resolving the data leakage problem in Häger-Köre's implementation. The confusion matrix results for the FDs over 15.8×10^6 data points are presented in Table 5.1, while Table 5.2 displays the ratios of all FDs concerning the best one, in terms of both false positives (FP) and false negatives (FN). Figure 5.1 provides an intuitive understanding of FP and FN errors across all ML-based FDs. For detailed derived metrics data and their respective ratios concerning all FDs relative to the best one, see Table 5.3 and Table 5.4.

These results were collected for both individual ML-based FDs and ensemble FDs, namely *Stacking*, *BlendStacking* and *Ensemble*. *Stacking* FD incorporates *SVM* and *Random Forest*, leveraging *SVM*'s superior performance in minimizing FP errors and

Random Forest's effectiveness in reducing FN errors. The aim is to capitalize on the strengths of both *SVM* and *Random Forest* within the *Stacking* FD.

Additionally, *BlendStacking* FD integrates *SVM*, *Random Forest*, and analytical FDs: BDBD and TTF (Section 2.3). As per the findings in Section 5.2, while ML-based FDs demonstrate fewer prediction errors with detection time under two seconds, analytical FDs exhibit a lower average mistake rate with longer detection time. Through the integration of the outputs from both ML-based FDs and analytical FDs, we hope the *BlendStacking* FD can deliver robust performance across all detection time intervals. The *Ensemble* FD integrates all individual ML-based FDs.

From the analysis of Figure 5.1, it shows that the FDs *SVM* and *Naive Bayes* tend to produce more FN errors. Conversely, the other FDs (*AdaBoost*, *Neural Network*, *Logistics Regression*, *Random Forest* and *Decision Tree*) tilt to more FP errors. Notably, *SVM* produces the minimum number of FP errors, followed by *Random Forest* and *Logistics Regression*. Although *Neural Network* produced the maximum number of FP errors, it has the fewest FN errors. Note that *Neural Network*, *Naive Bayes* and *Decision Tree* are the most polarized FDs concerning FP and FN errors whereas *SVM* and *Random Forest* can harmonize these types of errors. In summary, when considering both FP and FN errors, the individual ML-based FDs *SVM* and *Random Forest* are the best, whereas *Naive Bayes* ranks the least favorable classifier. The *Stacking* FD and *Ensemble* FD strike a good balance between FP and FN errors, while the performance of *BlendStacking* FD is not eye-catching.

According to the data in Table 5.2, although the ensemble FDs, namely *Stacking* and *BlendStacking*, may not perform the best in terms of FP errors, they are not far from the best-performing classifier, specifically *SVM*. The individual ML-based FDs, such as *Random Forest* and *Naive Bayes* are also performing well. Moreover, The individual ML-based FDs, including *Random Forest*, *AdaBoost* and *Decision Tree*, along with the ensemble FD *Stacking* show strong performance in terms of FN errors, closely trailing the top performer (*Neural Network*).

Figure 5.2 demonstrates the comprehensive comparison of classifier performance in terms of the various metrics encompassing accuracy, sensitivity, specificity, precision, and AUC ROC. In general, *SVM*, *Random Forest* and *Logistics Regression* perform well across all metrics. While *AdaBoost* and *Decision Tree* show notable performance across several metrics, their behaviors in terms of specificity and precision are not satisfying. *Neural Network* and *Naive Bayes* present good performance in one or two metrics, albeit manifesting poor results in others.

5.2 ML-based FDs with PD Module and Fallback

We now show the performance of ML-based FDs across various levels of the average mistake rate (λ_M) (Section 4.2.2) with PD module enabled and BDBD as the fallback. Note that as the **Naive Bayes** FD produces significantly more prediction errors than other FDs, we exclude it from subsequent evaluations. Following the footsteps of Häger and Köre [9], we optimized the training of ML-based FDs using the Precision-Recall (PR) curve, where Precision values on the y-axis and Recall values on the

ML-Classifiers	FP	TP	FDR	FN	TN	FOR
AdaBoost	4952	3683719	0.1342%	2990	12153543	0.0246%
Neural Network	8225	3684027	0.2228%	2682	12150270	0.0221%
Logistics Regression	4150	3680967	0.1126%	5742	12154345	0.0472%
Naive Bayes	3217	3666098	0.0877%	20611	12155278	0.1693%
Random Forest	2868	3683815	0.0778%	2894	12155627	0.0238%
Decision Tree	6486	3683599	0.1758%	3110	12152009	0.0256%
SVM	1630	3679244	0.0443%	7465	12156865	0.0614%
Stacking	2288	3683619	0.0621%	3090	12156207	0.0254%
BlendStacking	2463	3680042	0.0669%	6667	12156032	0.0548%
Ensemble	2248	3682871	0.0610%	3838	12156247	0.0316%

Table 5.1: The confusion matrix data results from evaluating the classifiers on the test set. Refer to Section 4.2.1 for the definition of these metrics.

	AB	NN$\diamond$	LR	NB	RF	DT	SVM$*$	Stacking	BlendStacking	Ensemble
FP Ratios	2.038	4.046	1.546	0.974	0.760	2.979	0	0.4037	0.5110	0.3791
FN Ratios	0.115	0	1.141	6.685	0.079	0.160	1.783	0.1521	1.4858	0.4310

Table 5.2: The ratio $x/X - 1$ is evaluated for each FD, where x is the number of FP or FN, and X is the best FP, and respectively, FN value among all FDs. The best FP value is the SVM FD (marked with $*$), and the best FN value is the Neural Network FD (marked with $\diamond$).

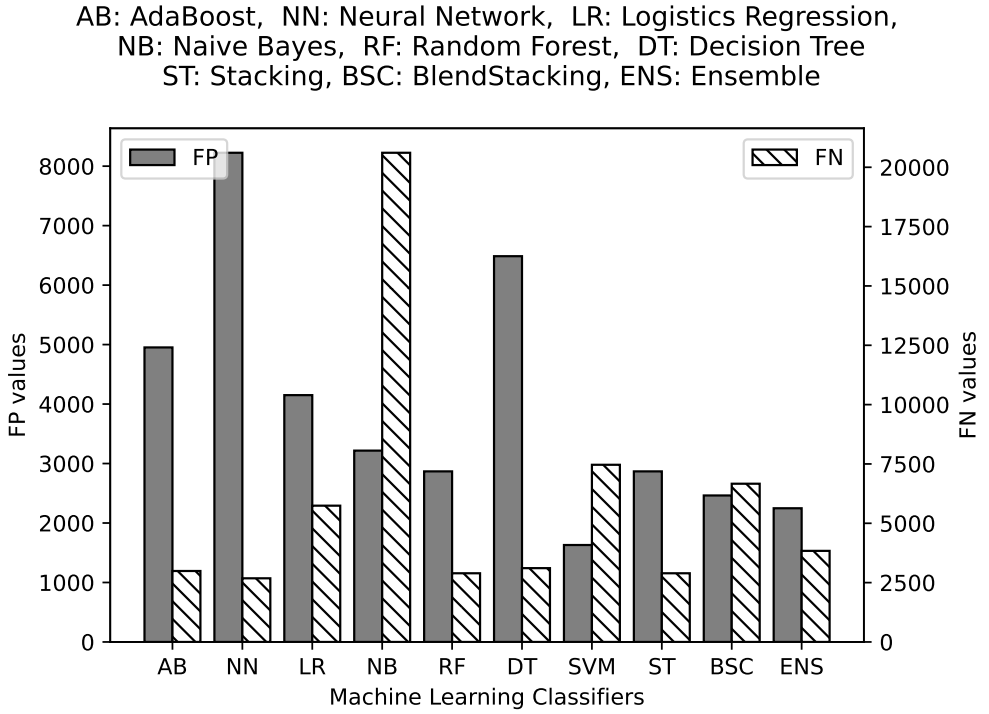


Figure 5.1: Visualization of the FP and FN values for different ML classifiers.

ML-Classifers	Accuracy	Sensitivity	Specificity	Precision	Fall-Out	AUC ROC
AdaBoost	0.999499	0.999189	0.999593	0.99866	0.000407	0.999925358
Neural Network	0.999312	0.999273	0.999324	0.997772	0.000676	0.999298020
Logistics Regression	0.999376	0.998443	0.999659	0.998874	0.000341	0.999879967
Naive Bayes	0.998496	0.994409	0.999735	0.999123	0.000265	0.999045223
Random Forest	0.999636	0.999215	0.999764	0.999222	0.000236	0.999828069
Decision Tree	0.999394	0.999156	0.999467	0.998242	0.000533	0.999820163
SVM	0.999426	0.997975	0.999866	0.999557	0.000134	0.999876591
Stacking	0.999636	0.999215	0.999764	0.999557	0.000236	0.999603392
BlendStacking	0.999424	0.998192	0.999797	0.999331	0.000203	0.999632772
Ensemble	0.999616	0.998959	0.999815	0.999390	0.000185	

Table 5.3: The classification matrics derived from the confusion matrix data.

ML-Classifers	Accuracy Ratio	Sensitivity Ratio	Specificity Ratio	Precision Ratio	Fall-Out Ratio	AUC ROC Ratio
AdaBoost	0.376374	0.115543	2.037313	2.024831	-0.000273	0
Neural Network	0.890110	0	4.044776	4.029345	-0.000542	8.404625
Logistics Regression	0.714286	1.141678	1.544776	1.541761	-0.000207	0.608116
Naive Bayes	3.131868	6.690509	0.977612	0.979684	-0.000131	11.791418
Random Forest	0	0.079780	0.761194	0.756208	-0.000102	1.303408
Decision Tree	0.664835	0.160935	2.977612	2.968397	-0.000399	1.409327
SVM	0.576923	1.785420	0	0	0	0.653345
Stacking	0	0.079780	0.761194	0	-0.000102	4.313470
BlendStacking	0.582418	1.486933	0.514925	0.510158	-0.000069	3.919857
Ensemble	0.054945	0.431912	0.380597	0.376975	-0.000051	

Table 5.4: The ratio $(1-x)/(1-X)-1$ is evaluated for each classification metric, where X represents the best value within a specific column. Note that the best Fall-Out value corresponds to the smallest value, whereas for other metrics, the best value is the largest one.

x-axis. The threshold values that we used here are defined in Table 4.2.

Please note that the thresholds were selected based on the level of Specificity the FDs achieved in the training set. We want to know the Specificity they attained in the test set. The Specificity of FDs at various thresholds are presented in Table 5.5. In Table 5.6, we provides the ratio of the detection time (T_D) of both individual ML-based FDs and the ensemble FDs to that of BDBD across configurations, including C_{PD}, $C_{fallback,higher}$, $C_{fallback,equal}$, and $C_{fallback,lower}$. For a comprehensive overview, Figure 5.3 shows the trade-off between T_D and λ_M for the configuration of C_{PD}. The evaluation results for the configuration $C_{fallback}$ are shown in Figures 5.4, 5.5, and 5.6. These figures correspond to the results of $C_{fallback,higher}$, $C_{fallback,equal}$ and $C_{fallback,lower}$, respectively.

Table 5.6 shows that the T_D for ML-based FDs is generally better than that for BDBD solution across various error rate levels when $\lambda_M \geq 1 \times 10^{-5}$. However, for error rates below 1×10^{-5}, some ML-based FDs show a higher detection time compared to BDBD (indicated by values greater than 1 in the corresponding cells). Note that lower λ_M requires longer T_D, the data in the table implies that some ML-based FDs might perform worse than BDBD when we allow longer T_D.

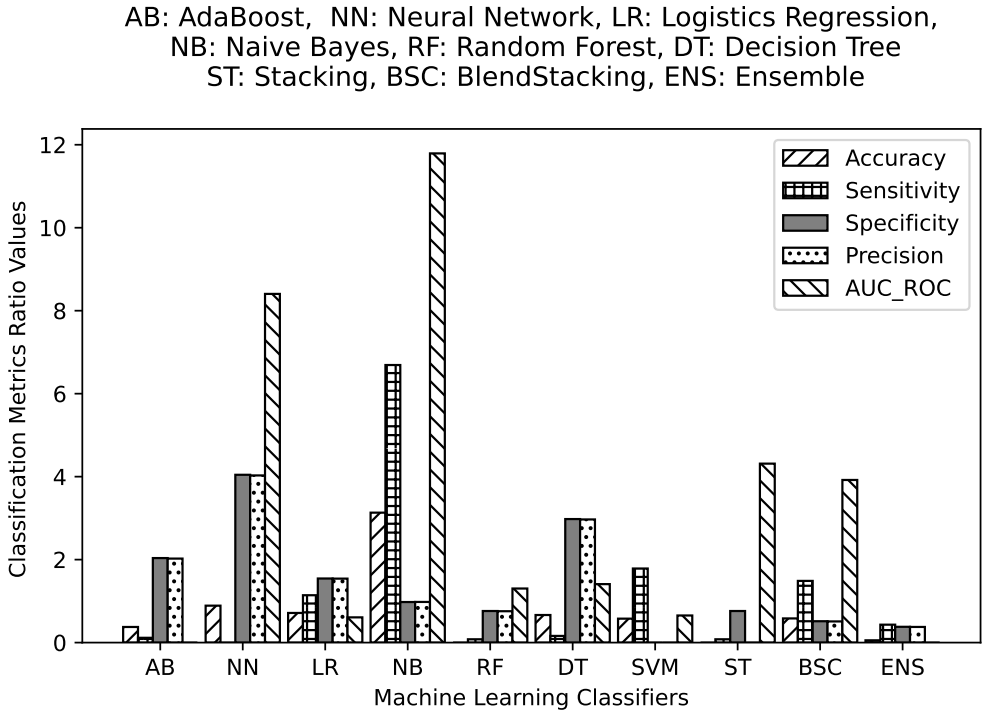


Figure 5.2: Visualization of the classification metrics values for different ML classifiers.

Figure 5.3 shows a similar trend from another perspective. The left-hand figure, a zoomed-in view, shows that when $\lambda_M \geq 10^{-5}$ and the detection time is less than 2 seconds, all individual ML-based FDs can achieve lower error rates than analytical FDs (TTF and BDBD). The right-hand figure (zoomed-out view) shows that individual ML-based FDs perform worse than analytical FDs when the detection time is longer than 5 seconds. In the context of individual ML-based FDs, it is obvious that *Random Forest* generally performs better than the other classifiers, and *Nerual Network* has the highest error rate but the shortest detection time. This is consistent with the results in Section 5.1.

Furthermore, when looking at the ensemble FDs (*Stacking*, *BlendStacking*, and *Ensemble*), we see that they consistently outperform analytical FDs when the detection time is under 2 seconds, with the *Stacking* FD consistently exhibiting superior performance across all configurations. However, as the detection time increases, the *Stacking* FD quickly becomes worse than analytical FDs, then the *Ensemble* FD also becomes worse than analytical FDs. Only *BlendStacking* FD is always better than analytical FDs. Despite variations in details across all configurations, Figures 5.4, 5.5, and 5.6 illustrate similar trends.

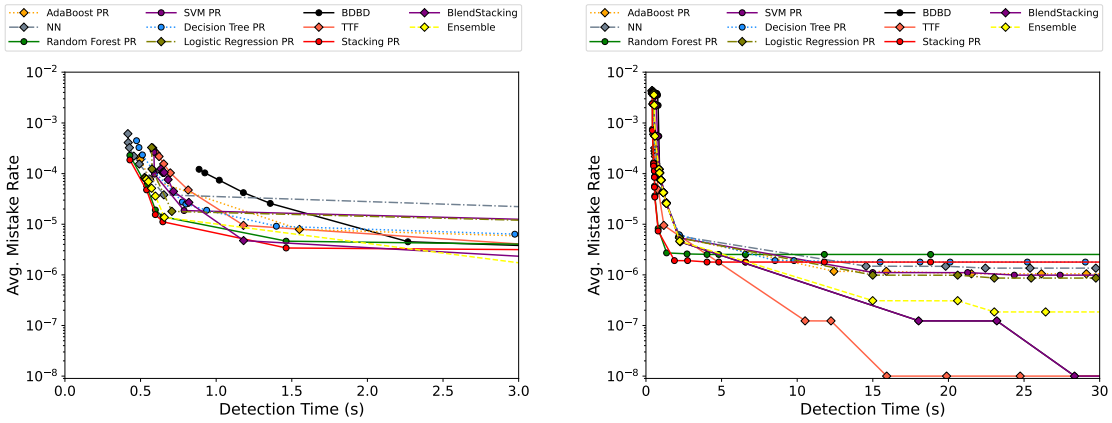
FD (config)	Specificity
AdaBoost PR ($t = 0.999$)	0.99978088
AdaBoost PR ($t = 0.9999$)	0.99998784
AdaBoost PR ($t = 0.99999$)	0.99999880
AdaBoost PR ($t = 0.9999999$)	0.99999880
Decision Tree PR ($t = 0.999$)	0.99952300
Decision Tree PR ($t = 0.9999$)	0.99995721
Decision Tree PR ($t = 0.99999$)	0.99999811
Decision Tree PR ($t = 0.9999999$)	0.99999811
Logistic Regression PR ($t = 0.999$)	0.99965037
Logistic Regression PR ($t = 0.9999$)	0.99999105
Logistic Regression PR ($t = 0.99999$)	0.99999905
Logistic Regression PR ($t = 0.9999999$)	0.99999905
NN ($t = 0.999$)	0.99935398
NN ($t = 0.9999$)	0.99998885
NN ($t = 0.99999$)	0.99999842
NN ($t = 0.999999$)	0.99999842
Random Forest PR ($t = 0.999$)	0.99976380
Random Forest PR ($t = 0.9999$)	0.99993937
Random Forest PR ($t = 0.99999$)	0.99999634
Random Forest PR ($t = 0.999999$)	0.99999710

FD (config)	Specificity
SVM PR ($t = 0.999$)	0.99970973
SVM PR ($t = 0.9999$)	0.99999351
SVM PR ($t = 0.99999$)	0.99999899
SVM PR ($t = 0.999992$)	0.99999899
TTF ($t = 0.999$)	0.99956485
TTF ($t = 0.9999$)	0.99999950
TTF ($t = 0.99992$)	1.0
BDBD ($t = 0.999$)	0.98968365
BDBD ($t = 0.9999$)	0.99983936
BDBD ($t = 0.99999$)	0.99999950
BDBD ($t = 0.999992$)	1.0
Ensemble ($t = 0.999$)	0.99982487
Ensemble ($t = 0.9999$)	0.99999338
Ensemble ($t = 0.99999$)	0.99999924
Ensemble ($t = 0.999999$)	0.99999924
Stacking ($t = 0.999$)	0.99980930
Stacking ($t = 0.9999$)	0.99995066
Stacking ($t = 0.99999$)	0.99999779
Stacking ($t = 0.999999$)	0.99999792
BlendStacking ($t = 0.999$)	0.99983703
BlendStacking ($t = 0.9999$)	0.99999950
BlendStacking ($t = 0.99999$)	1.0

Table 5.5: The specificity attained on the test set when employing a configuration with a threshold that yielded a specificity of t in the training set.

λ_M	**Configuration**	**AB PR**	**NN**	**RF PR**	**SVM PR**	**DT PR**	**LR PR**	**Stacking**	**BlendStacking**	**Ensemble**
$\lambda_M \leq 1 \times 10^{-2}$	C_{PD}	0.560	0.469	0.484	0.664	0.534	0.646	0.484	0.715	0.594
	$C_{fallback,higher}$	0.020	0.018	0.018	0.025	0.020	0.025	0.018	0.030	0.023
	$C_{fallback,equal}$	0.513	0.463	0.478	0.614	0.507	0.599	0.478	0.715	0.588
	$C_{fallback,lower}$	0.645	0.588	0.611	0.732	0.639	0.718	0.611	0.832	0.723
$\lambda_M \leq 1 \times 10^{-3}$	C_{PD}	0.560	0.469	0.484	0.664	0.534	0.646	0.484	0.715	0.594
	$C_{fallback,higher}$	0.020	0.018	0.018	0.025	0.020	0.025	0.018	0.030	0.023
	$C_{fallback,equal}$	0.513	0.463	0.478	0.614	0.507	0.599	0.478	0.715	0.588
	$C_{fallback,lower}$	0.598	0.670	0.500	0.723	0.638	0.691	0.500	0.911	0.703
$\lambda_M \leq 1 \times 10^{-4}$	C_{PD}	0.549	0.641	0.529	0.580	0.761	0.693	0.529	0.669	0.516
	$C_{fallback,higher}$	0.023	0.028	0.023	0.026	0.033	0.030	0.023	0.030	0.023
	$C_{fallback,equal}$	0.506	0.572	0.527	0.675	0.721	0.624	0.527	0.669	0.560
	$C_{fallback,lower}$	0.968	0.959	0.550	0.967	0.903	0.960	0.550	1.0	0.979
$\lambda_M \leq 1 \times 10^{-5}$	C_{PD}	0.684	4.905	0.645	5.510	0.616	5.198	0.645	0.521	1.445
	$C_{fallback,higher}$	0.066	0.479	0.063	0.539	0.060	0.508	0.063	0.030	0.508
	$C_{fallback,equal}$	0.655	1.114	0.644	1.743	0.598	1.425	0.644	0.521	1.4300
	$C_{fallback,lower}$	0.976	0.983	0.356	0.982	0.972	0.975	0.356	1.0	0.985

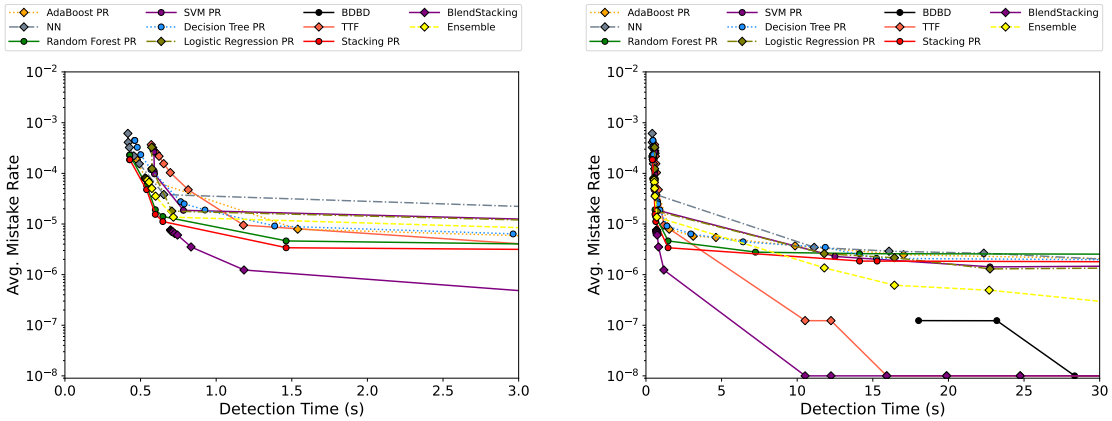
Table 5.6: The detection time (T_D) of each FD relative to the T_D of the BDBD solution for attaining specific values of the average mistake rate (λ_M) in the configurations: C_{PD}, $C_{fallback,higher}$, $C_{fallback,equal}$, and $C_{fallback,lower}$. AB, NN, RF, LR and DT are the abbreviations of Adaboost, Neural Network, Random Forest, Logistics Regression and Decision Tree respectively.



(a) Result for the zoomed-in time range 0 to 3 seconds.

(b) Result over the time range 0 to 30 seconds.

Figure 5.3: The trade-off between T_D and λ_M for the results of C_{PD} without fallback.



(a) Result for the zoomed-in time range 0 to 3 seconds.

(b) Result over the time range 0 to 30 seconds.

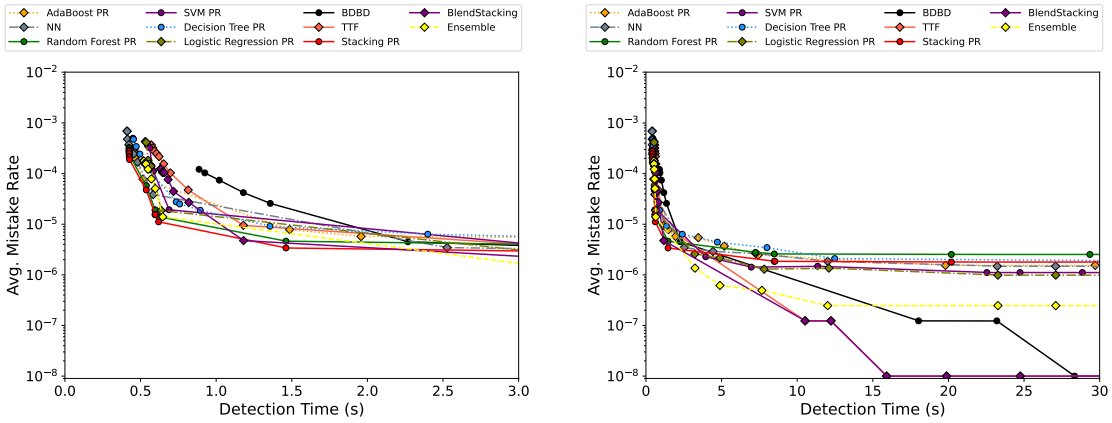
Figure 5.4: The trade-off between T_D and λ_M for the results of $C_{fallback,higher}$ and C_{PD}.

5.3 Clustering ML-Based Binary Classifiers

5.3.1 Crash Intervals

The ML-Classifiers were grouped into clusters during the crash intervals in which at least one ML-Classifier reported FN using K-means algorithm [28]. The entire test dataset consisted of 10,573 crash intervals, and among these, a total of 10,537 crash intervals were identified wherein at least one ML-Classifier reported FN. For the crash intervals where no ML-Classifier reported FN errors, clustering the ML-Classifiers was unnecessary.

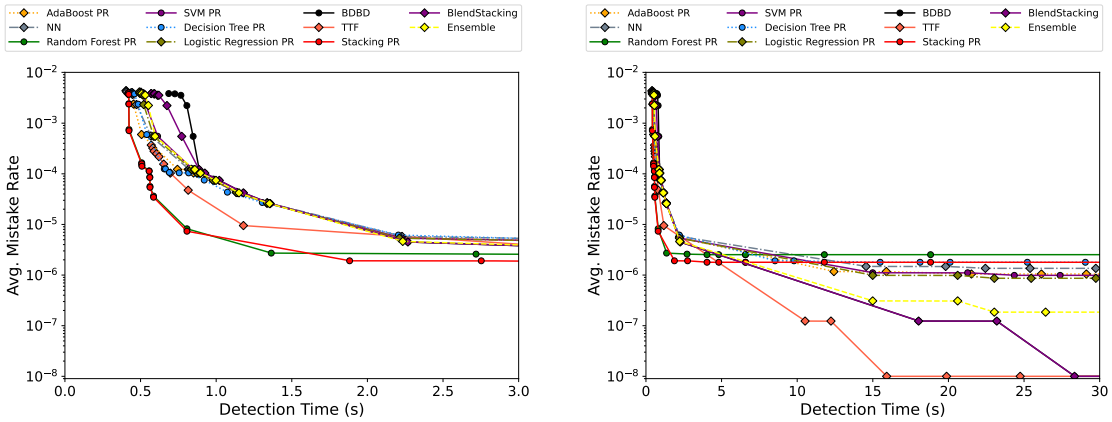
From the Table 5.1 and the clustering results (table A.2) from the K-means algorithm, we notice that:



(a) Result for the zoomed-in time range 0 to 3 seconds.

(b) Result over the time range 0 to 30 seconds.

Figure 5.5: The trade-off between T_D and λ_M for the results of $C_{fallback,equal}$ and C_{PD}.



(a) Result for the zoomed-in time range 0 to 3 seconds.

(b) Result over the time range 0 to 30 seconds.

Figure 5.6: The trade-off between T_D and λ_M for the results of $C_{fallback,lower}$ and C_{PD}.

- The Naive Bayes classifier contributes to the majority of FNs, whereas the remaining classifiers exhibit a relatively small number of FNs. The occurrence of crash intervals wherein more than three ML-Classifiers report FN is infrequent. Based on this observation, it is appropriate to set the number of clusters utilized by the K-means algorithm to two for the majority of crash intervals.

- Throughout the majority of crash intervals, the occurrence of FNs is typically reported by each classifier only once. In instances where a classifier does report FNs multiple times within a single crash interval, it is observed that these instances of FN reports from the classifier are predominantly contiguous. This finding implies that situations akin to the "Cried Wolf" phenomenon, where a classifier frequently oscillates between FN and True Positive (TP) classifications,

are very uncommon.

An attempt was also made to assign weights to various samples using a decreasing arithmetic progression. However, the introduction of weights yielded negligible impact for the majority of crash periods.

5.3.2 Crash-free Intervals

The ML-Classifiers were also grouped into clusters during the crash-free intervals in which at least one ML-Classifier reported FP using K-means algorithm [28]. Again, clustering of ML-Classifiers was unnecessary during crash-free intervals where no ML-Classifier reported FP. The entire test dataset consisted of 3,569 crash-free intervals, of which instances were observed where only one classifier reported FP at some data points, while at other data points, two or more classifiers reported FPs. The Logistics Regression classifier and the Naive Bayes classifier generated the most FPs, whereas the remaining classifiers showed relatively consistent numbers of FPs generated.

From the Table 5.1 and the clustering results (table A.2) from the K-means algorithm, we notice that:

- Instances wherein more than two classifiers simultaneously report FPs at the same data point are rare. Therefore, we explored the experiments with two and three clusters when using the K-means algorithm. Setting the number of clusters to a higher value did not yield substantial or meaningful outcomes.
- In most cases, the Logistics Regression classifier and the Naive Bayes classifier were consistently clustered together as one group, while the remaining classifiers were grouped separately as another distinct cluster.

5.3.3 Ranking of ML-Based Binary Classifiers

To determine the presence of structural patterns within these classifiers, a ranking in terms of specificity and sensitivity was conducted. The result of this ranking is presented in Table 5.4. The top-performing classifiers based on this ranking include *Random Forest, AdaBoost, SVM*, and *Decision Tree*.

5.3.3.1 Pairwise Comparison

The comparison results of *Random Forest, SVM* and *Decision Tree* are presented in Table 5.7. This table shows that the closest two classifiers are *Random Forest* and *SVM*. Furthermore, the comparison of *Random Forest, SVM* and *AdaBoost* is presented in Table 5.8, which shows that *AdaBoost* is closer to *SVM*.

5.4 Li-Marin LSTM FD

In this section, we present the result of Li-Marin LSTM FD for lower and higher over-estimation. The confusion matrix data for LSTM FD is presented in Table

	SVM vs Decision Tree	**Random Forest vs Decision Tree**	**Random Forest vs SVM**
FP	(SVM & Decision Tree): 3727753 (¬SVM & Decision Tree): 104 (SVM & ¬Decision Tree): 171 (¬SVM & ¬Decision Tree): 694	(Random Forest & Decision Tree): 3727851 (¬Random Forest & Decision Tree): 6 (Random Forest & ¬Decision Tree): 315 (¬Random Forest & ¬Decision Tree): 550	(Random Forest & SVM): 3727890 (¬Random Forest & SVM): 34 (Random Forest & ¬SVM): 276 (¬Random Forest & ¬SVM): 522
FN	(SVM & Decision Tree): 12118132 (¬SVM & Decision Tree): 8 (SVM & ¬Decision Tree): 77 (¬SVM & ¬Decision Tree): 152	(Random Forest & Decision Tree): 12118135 (¬Random Forest & Decision Tree|: 5 (Random Forest & ¬Decision Tree): 20 (¬Random Forest & ¬Decision Tree): 209	(Random Forest & SVM): 12118143 (¬Random Forest & SVM): 66 (Random Forest & ¬SVM): 12 (¬Random Forest & ¬SVM): 148

Table 5.7: The pairwise comparison of classifiers *Random Forest, SVM* and *Decision Tree*.

	SVM vs Adaboost	**Random Forest vs Adaboost**
FP	(SVM & Adaboost): 3727876 (¬SVM & Adaboost): 48 (SVM & ¬Adaboost): 276 (¬SVM & ¬Adaboost): 522	(Random Forest & Adaboost): 3728055 (¬Random Forest & Adaboost): 111 (Random Forest & ¬Adaboost): 97 (¬Random Forest & ¬Adaboost): 459
FN	(SVM & Adaboost): 12118142 (¬SVM & Adaboost): 67 (SVM & ¬Adaboost): 117 (¬SVM & ¬Adaboost): 43	(Random Forest & Adaboost): 12118142 (¬Random Forest & Adaboost): 13 (Random Forest & ¬Adaboost): 117 (¬Random Forest & ¬Adaboost): 97

Table 5.8: The comparisons of classifiers *SVM* vs *Adaboost* and *Random Forest* vs *Adaboost* .

5.9, along with the result of LSTM binary classifier FD implemented by Häger and Köre in [9]. Table 5.10 shows the classification metrics that are derived from Table 5.9. Note that the evaluation of the LSTM FD involves assessing its probability of availability (P_A) and the classification metrics including Accuracy, Specificity, Precision, Sensitivity and Fall-Out.

As expected, the Li-Marin LSTM model with higher over-estimation tends to generate more FN errors, incorrectly identifying faulty nodes as non-faulty. Conversely, when exhibiting lower over-estimation, it tends to generate more FP errors, wrongly

LSTM FD	FP	TP	FDR	FN	TN	FOR
LSTM Binary Classifier FD	33829	3684769	0.9097%	1872	12116930	0.0154%
Li-Marin LSTM with lower over-estimation	24122	387254201	0.0062%	7551	107519792	0.0070%
Li-Marin LSTM with higher over-estimation	6818	386813574	0.0018%	12473	107530313	0.0160%

Table 5.9: The confusion matrix data results of LSTM classifier and LSTM FD.

LSTM FD	P_A	Accuracy	Sensitivity	Specificity	Precision	Fall-Out	AUC ROC
LSTM Binary Classifier FD		0.997746	0.999492	0.997216	0.990903	0.002784	0.998354
Li-Marin LSTM with lower Over-estimation	0.998643	0.999537	0.999972	0.999291	0.997179	0.000087	
Li-Marin LSTM with higher Over-estimation	0.998547	0.999101	0.999867	0.999941	0.999788	0.000059	

Table 5.10: The P_A and classification metrics results of LSTM classifier FD and the Li-Marin LSTM FD at lower and higher degree of over-estimation.

identifying non-faulty nodes as faulty. Table 5.9 shows that the Li-Marin LSTM FD outperforms the LSTM classifier FD in terms of FP errors, regardless of whether it has a lower or higher over-estimation. However, concerning FN errors, their performances are comparable. Besides, as shown in Table 5.10, the Li-Marin LSTM FD performs better than the LSTM classifier FD across all metrics.

5.5 Discussion of Results

5.5.1 Interpretation

We evaluated the performance of seven individual ML-based FDs, As detailed in Section 5.1, the evaluation on our dataset reveals different levels of FP and FN across these FDs. Among the individual ML-based FDs, three FDs tend to produce more FP errors, and four FDs lean towards generating more FN errors. In Section 5.3, we show that they have a certain structural pattern. The Random Forest, SVM, and AdaBoost FDs are close to each other in terms of FP and FN levels. Except for Naive Bayes FD, the accuracy of all individual ML-based FDs is three nines, which indicates that ML-based FDs are effective for the failure nodes detection problem. Moreover, all ML-based FDs more or less produced FP and FN errors, which may indicate that some data points are difficult to classify.

The results of the configuration $C_{fallback}$ in Section 5.2 show that using BDBD as a fallback can enhance the performance of ML-based FDs in certain scenarios while diminishing it in others. Most ML-based FDs remained unaffected by the fallback in $C_{fallback,higher}$. This could be attributed to the FDs having already determined their final suspicions before the fallback was triggered. The ML-based FDs in $C_{fallback,equal}$ demonstrated improvements in detection time compared to C_{PD} with the help of fallback. Conversely, in $C_{fallback,lower}$, the fallback caused longer detection time for the ML-based FDs. This might be because the fallback threshold is lower than the threshold of the PD module, leading to the suspicions made by ML-based FDs being

overridden by the fallback.

Section 5.2 reveals that ML-based FDs exhibit a shorter detection time for failure nodes compared to analytical FDs, namely BDBD and TTF. However, analytical FDs can achieve a higher accuracy when the detection time is longer. The ensemble FDs combine the advantages of both ML-based FDs and analytical FDs to a certain extent. Specifically, the *Stacking* and *Ensemble* FDs can achieve a higher accuracy than individual ML-based FDs. Nevertheless, since they are built on top of part or all of ML-based FDs, they are still subject to the characteristics of ML-based FDs. Consequently, their accuracies still fall short of analytical FDs when the detection time becomes longer. The *BlendStacking* FD overcomes the shortcomings of ML-based FDs and the analytical FDs. That is to say, it can identify failure nodes more rapidly than analytical FDs, and when the detection time becomes longer, it can achieve the same accuracy as analytical FDs.

5.5.2 Limitations

Since our experiments were conducted on the "simulative" environment as described in Section 4.1, the results only reflect the performance of FDs in this environment. It does not represent the performance of FDs in a real environment, where some non-faulty nodes may be slow to reply. Furthermore, to deploy these FDs in a real-world setting, they need to be retrained using the data in the real environment.

Given that we use the dataset from Häger and Köre, the limitations associated with their dataset also apply to our study. The test dataset has only 15.8×10^6 data points, and the number of data points corresponding to results with $\lambda_M \leq 10^{-5}$ is even smaller. This is because such data points are generated based on specific conditions. One should be cautious when interpreting our results. Furthermore, this concern is exacerbated by the specificity of the results, which stem from data collected from only eight highly stable nodes within a relatively short period spanning a few weeks.

The pairwise comparison in Section 5.3 is incomplete. We mainly compared six different ML-based FDs, a complete set of pairwise comparisons entails $\binom{6}{2}$ individual comparisons. Each of these comparisons demands a considerable amount of time, rendering the total time required for this task unmanageable. To mitigate this time constraint, we have chosen to limit the pairwise comparison to FDs that have better performance and similar behavior. It is important to note that this approach may miss the potential to get better FDs by combining the two ML-based FDs because the pairwise comparison is not conducted on these two FDs.

6

Conclusion

ML-based FDs are proven to be effective in detecting faulty nodes within a distributed system. We evaluated the performance of FDs by measuring Detection Time (T_D) and Average Mistake Rate (λ_M). While individual ML-based FDs exhibit different levels of performance, our analysis involving clustering and ranking identified the Random Forest FD and the SVM FD as the top performers. Nonetheless, although ML-based FDs reduce detection time, their error rate is higher than the analytical FD, especially when we allow longer detection time. To tackle this issue, we devised three ensemble FDs. Notably, one ensemble FD, which integrates the top-performing ML-based FDs with the analytical FD, successfully achieves a good balance between T_D and λ_M.

The PD module helps in enhancing the accuracy of ML-based FDs through the threshold mechanism. This threshold invalidates ML-based FD results when the precision level is too low. Despite the improvements in accuracy facilitated by the PD module, ML-based FDs with PD enabled cannot guarantee completeness property. Therefore, we adopt the BDBD as a fallback for ML-based FDs to address this limitation. However, while using BDBD as a fallback aids in addressing completeness concerns, it does not guarantee accuracy. Its impact on ML-based FDs is contingent upon the relationship between the BDBD threshold and the PD threshold. Specifically, when the BDBD threshold is larger than the PD threshold, it minimally affects ML-based FDs. Conversely, when the two thresholds are equal, it contributes to enhancing the accuracy of ML-based FDs. However, if the BDBD threshold is lower than the PD threshold, it prolongs the detection time for ML-based FDs.

We implemented the Li-Marin LSTM FD and evaluated its performance on our dataset. In comparison to Chen's heartbeat FD [25], this FD demonstrates significantly reduced detection time while maintaining a similar P_A. Note that this enhancement is achieved at the cost of a rise in computational overhead. When evaluating FDs based on ML classification, this FD does not exhibit a substantial advantage. Besides, fine-tuning the degree of overestimation remains a challenging task.

Bibliography

[1] C. Cachin, R. Guerraoui, and L. E. T. Rodrigues, *Introduction to Reliable and Secure Distributed Programming (2. ed.)* Springer, 2011, ISBN: 978-3-642-15259-7. DOI: 10.1007/978-3-642-15260-3. [Online]. Available: https://doi.org/10.1007/978-3-642-15260-3 (visited on 07/01/2022).

[2] M. J. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," in *Foundations of Computation Theory*, M. Karpinski, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 127–140, ISBN: 978-3-540-38682-7.

[3] M. J. Fischer, N. A. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985. DOI: 10.1145/3149.214121. [Online]. Available: https://doi.org/10.1145/3149.214121 (visited on 07/01/2022).

[4] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[5] P. Blanchard, S. Dolev, J. Beauquier, and S. Delaët, "Practically self-stabilizing paxos replicated state-machine," in *Networked Systems - Second International Conference, NETYS 2014, Marrakech, Morocco, May 15-17, 2014. Revised Selected Papers*, G. Noubir and M. Raynal, Eds., ser. Lecture Notes in Computer Science, vol. 8593, Springer, 2014, pp. 99–121. DOI: 10.1007/978-3-319-09581-3_8. [Online]. Available: https://doi.org/10.1007/978-3-319-09581-3%5C_8 (visited on 07/01/2022).

[6] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Communication-efficient leader election and consensus with limited link synchrony," in *PODC*, ACM, 2004, pp. 328–337.

[7] A. Mostefaoui, E. Mourgaya, and M. Raynal, "Asynchronous implementation of failure detectors," in *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.*, 2003, pp. 351–360. DOI: 10.1109/DSN.2003.1209946.

[8] M. Raynal, *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018, ch. 18, ISBN: 978-3-319-94140-0. DOI: 10.1007/978-3-319-94141-7. [Online]. Available: https://doi.org/10.1007/978-3-319-94141-7 (visited on 03/01/2022).

[9] G. Häger and J. Köre, "When crash fault tolerance meets machine learning," M.S. thesis, Department of Computer Science, Engineering, Chalmers University of Technology, and University of Gothenburg, 2022.

[10] X. Li and O. Marin, "Towards implementing ml-based failure detectors," *CoRR*, vol. abs/2210.00134, 2022.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] W. Chen, S. Toueg, and M. Aguilera, "On the quality of service of failure detectors," *IEEE Transactions on Computers*, vol. 51, no. 1, pp. 13–32, 2002. DOI: 10.1109/12.980014.

[13] M. Bertier, O. Marin, and P. Sens, "Implementation and performance evaluation of an adaptable failure detector," in *Proceedings International Conference on Dependable Systems and Networks*, 2002, pp. 354–363. DOI: 10.1109/DSN.2002.1028920.

[14] M. K. Aguilera, W. Chen, and S. Toueg, "Heartbeat: A timeout-free failure detector for quiescent reliable communication," in *WDAG*, ser. Lecture Notes in Computer Science, vol. 1320, Springer, 1997, pp. 126–140.

[15] V. K. Ayyadevara, *Pro Machine Learning Algorithms : A Hands-On Approach to Implementing Algorithms in Python and R*. Berkeley, CA: Apress, 2018, ISBN: 978-1-4842-3564-5. DOI: 10.1007/978-1-4842-3564-5. [Online]. Available: https://doi.org/10.1007/978-1-4842-3564-5 (visited on 07/01/2022).

[16] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[17] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951. DOI: 10.1214/aoms/1177729586. [Online]. Available: https://doi.org/10.1214/aoms/1177729586 (visited on 07/01/2022).

[18] J. Kiefer and J. Wolfowitz, "Stochastic Estimation of the Maximum of a Regression Function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952. DOI: 10.1214/aoms/1177729392. [Online]. Available: https://doi.org/10.1214/aoms/1177729392 (visited on 07/01/2022).

[19] Vikramkumar, B. Vijaykumar, and Trilochan, *Bayes and naive bayes classifier*, 2014. DOI: 10.48550/ARXIV.1404.0933. [Online]. Available: https://arxiv.org/abs/1404.0933.

[20] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995. DOI: 10.1007/BF00994018. [Online]. Available: https://doi.org/10.1007/BF00994018 (visited on 07/01/2022).

[21] D. A. E. Acar, Y. Zhao, R. M. Navarro, M. Mattina, P. N. Whatmough, and V. Saligrama, "Federated learning based on dynamic regularization," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=B7v4QMR6Z9w (visited on 07/01/2022).

[22] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 4, pp. 1–21, 2012.

[23] A. Paszke, S. Gross, F. Massa, *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, 2019. arXiv: 1912.01703 [cs.LG].

[24] PlanetLabEurope, *Home*. [Online]. Available: https://www.planet-lab.eu/Home (visited on 07/01/2022).

[25] W. Chen, S. Toueg, and M. K. Aguilera, "On the quality of service of failure detectors," *IEEE Trans. Computers*, vol. 51, no. 5, pp. 561–580, 2002. DOI: 10.1109/TC.2002.1004595. [Online]. Available: https://doi.org/10.1109/TC.2002.1004595 (visited on 07/01/2022).

[26] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[27] K. Boyd, K. H. Eng, and C. D. P. Jr., "Area under the precision-recall curve: Point estimates and confidence intervals," in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*, H. Blockeel, K. Kersting, S. Nijssen, and F. Zelezný, Eds., ser. Lecture Notes in Computer Science, vol. 8190, Springer, 2013, pp. 451–466. DOI: 10.1007/978-3-642-40994-3_29. [Online]. Available: https://doi.org/10.1007/978-3-642-40994-3%5C_29 (visited on 07/01/2022).

[28] K. Krishna and M. Narasimha Murty, "Genetic k-means algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 3, pp. 433–439, 1999. DOI: 10.1109/3477.764879.

A

Additional Results

A.1 Clustering results of K-means

Table A.2 displays the clustering outcomes for seven classifiers, namely *AdaBoost, Decision Tree, Logistic Regression, Naive Bayes, Random Forest, SVM* and *ANN*. The clustering analysis was conducted on the complete dataset as well as three subsets and five subsets derived from the entire dataset.

	Number of Clusters: 2	Number of Clusters: 3	Number of Clusters: 4
The whole set	Crash period(FN): {"1": ["ab", "nn", "lr_cv", "random_forest", "decision_tree", "svm"], "0": ["nb"]} non crash period(FP): {"0": ["ab", "nn", "nb", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"]}	Crash period(FN): {"1": ["ab", "nn", "random_forest", "decision_tree", "svm"], "2": ["lr_cv"], "0": ["nb"]} non crash period(FP): {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]}	Crash period(FN): {"0": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "1": ["nb"]} non crash period(FP): {"0": ["ab", "random_forest", "decision_tree"], "3": ["nn", "svm"], "1": ["lr_cv"], "2": ["nb"]}
3 subsets	Crash period(FN): {"0": ["ab", "nn", "lr_cv", "random_forest", "decision_tree", "svm"], "1": ["nb"]} {"0": ["ab", "nn", "lr_cv", "random_forest", "decision_tree", "svm"], "1": ["nb"]} {"0": ["ab", "nn", "lr_cv", "random_forest", "decision_tree", "svm"], "1": ["nb"]} non crash period(FP): {"0": ["ab", "nn", "nb", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"]} {"0": ["ab", "nn", "nb", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"]}	Crash period(FN): {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "2": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "2": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "2": ["lr_cv"], "1": ["nb"]} non crash period(FP): {"1": ["ab", "nn", "random_forest", "decision_tree", "svm"], "0": ["lr_cv"], "2": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]}	Crash period(FN): {"1": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "0": ["nb"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "1": ["nb"]} non crash period(FP): {"3": ["ab", "random_forest"], "0": ["nn", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]} {"3": ["ab", "random_forest", "decision_tree"], "0": ["nn", "svm"], "1": ["lr_cv"], "2": ["nb"]}

	{"0": ["ab", "nn", "nb", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"]}	{"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]}	{"3": ["ab", "random_forest"], "1": ["nn", "decision_tree", "svm"], "2": ["lr_cv"], "0": ["nb"]}
5 subsets	Crash period(FN): {"1": ["ab", "nn", "lr_cv", "random_forest", "decision_tree", "svm"], "0": ["nb"]} {"0": ["ab", "nn", "lr_cv", "random_forest", "decision_tree", "svm"], "1": ["nb"]} {"0": ["ab", "nn", "lr_cv", "random_forest", "decision_tree", "svm"], "1": ["nb"]} {"1": ["ab", "nn", "lr_cv", "random_forest", "decision_tree", "svm"], "0": ["nb"]} {"0": ["ab", "nn", "lr_cv", "random_forest", "decision_tree", "svm"], "1": ["nb"]} non crash period(FP): {"1": ["ab", "nn", "nb", "random_forest", "decision_tree", "svm"], "0": ["lr_cv"]} {"0": ["ab", "nn", "nb", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"]} {"0": ["ab", "nn", "nb", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"]}	Crash period(FN): {"2": ["ab", "nn", "random_forest", "decision_tree", "svm"], "0": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "2": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "2": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "2": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "2": ["lr_cv"], "1": ["nb"]} non crash period(FP): {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]}	Crash period(FN): {"1": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "0": ["nb"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "1": ["nb"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "1": ["nb"]} non crash period(FP): {"1": ["ab", "random_forest", "decision_tree"], "3": ["nn", "svm"], "2": ["lr_cv"], "0": ["nb"]} {"3": ["ab", "random_forest"], "1": ["nn", "decision_tree", "svm"], "0": ["lr_cv"], "2": ["nb"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "3": ["nn"], "2": ["lr_cv"], "1": ["nb"]}

	{"0": ["ab", "nn", "nb", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"]} {"0": ["ab", "nn", "nb", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"]}	{"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]} {"0": ["ab", "nn", "random_forest", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]}	{"2": ["ab", "random_forest"], "0": ["nn", "decision_tree", "svm"], "1": ["lr_cv"], "3": ["nb"]} {"0": ["ab", "random_forest"], "3": ["nn", "decision_tree", "svm"], "1": ["lr_cv"], "2": ["nb"]}

Table A.2: The clustering outcomes for seven classifiers.

Table A.4 displays the clustering outcomes for top-performing five classifiers, namely *AdaBoost, Decision Tree, Random Forest, SVM* and *ANN*. The clustering analysis was conducted on the complete dataset as well as three subsets and five subsets derived from the entire dataset.

	Number of Clusters: 2	Number of Clusters: 3	Number of Clusters: 4
The whole set	Crash period(FN): {"1": ["ab", "random_forest", "decision_tree", "svm"], "0": ["nn"]} non crash period(FP): {"1": ["ab", "random_forest", "decision_tree"], "0": ["nn", "svm"]}	Crash period(FN): {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree", "svm"]} non crash period(FP): {"2": ["ab", "random_forest"], "1": ["nn", "svm"], "0": ["decision_tree"]}	Crash period(FN): {"0": ["ab"], "1": ["nn"], "2": ["random_forest", "decision_tree"], "3": ["svm"]} non crash period(FP): {"3": ["ab"], "0": ["nn", "svm"], "1": ["random_forest"], "2": ["decision_tree"]}
3 subsets	Crash period(FN): {"1": ["ab", "random_forest", "decision_tree", "svm"], "0": ["nn"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "1": ["nn"]} {"1": ["ab", "random_forest", "decision_tree", "svm"], "0": ["nn"]} non crash period(FP): {"1": ["ab", "random_forest"], "0": ["nn", "decision_tree", "svm"]} {"1": ["ab", "random_forest", "decision_tree"], "0": ["nn", "svm"]}	Crash period(FN): {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree", "svm"]} {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree", "svm"]} {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree", "svm"]} non crash period(FP): {"0": ["ab", "random_forest"], "1": ["nn", "svm"], "2": ["decision_tree"]} {"0": ["ab", "random_forest"], "1": ["nn", "svm"], "2": ["decision_tree"]}	Crash period(FN): {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree"], "3": ["svm"]} {"2": ["ab"], "0": ["nn"], "1": ["random_forest", "decision_tree"], "3": ["svm"]} {"0": ["ab"], "1": ["nn"], "2": ["random_forest", "decision_tree"], "3": ["svm"]} non crash period(FP): {"1": ["ab"], "0": ["nn", "svm"], "3": ["random_forest"], "2": ["decision_tree"]} {"2": ["ab"], "0": ["nn", "svm"], "3": ["random_forest"], "1": ["decision_tree"]}

	{"1": ["ab", "random_forest"], "0": ["nn", "decision_tree", "svm"]}	{"0": ["ab", "random_forest"], "1": ["nn", "svm"], "2": ["decision_tree"]}	{"2": ["ab", "random_forest"], "3": ["nn"], "0": ["decision_tree"], "1": ["svm"]}
5 subsets	Crash period(FN): {"0": ["ab", "random_forest", "decision_tree", "svm"], "1": ["nn"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "1": ["nn"]} {"1": ["ab", "random_forest", "decision_tree", "svm"], "0": ["nn"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "1": ["nn"]} {"0": ["ab", "random_forest", "decision_tree", "svm"], "1": ["nn"]} non crash period(FP): {"0": ["ab", "random_forest", "decision_tree"], "1": ["nn", "svm"]} {"0": ["ab", "random_forest"], "1": ["nn", "decision_tree", "svm"]} {"0": ["ab", "random_forest", "decision_tree"], "1": ["nn", "svm"]}	Crash period(FN): {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree", "svm"]} {"0": ["ab"], "1": ["nn"], "2": ["random_forest", "decision_tree", "svm"]} {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree", "svm"]} {"2": ["ab", "random_forest", "decision_tree"], "1": ["nn"], "0": ["svm"]} {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree", "svm"]} non crash period(FP): {"1": ["ab", "random_forest"], "0": ["nn", "svm"], "2": ["decision_tree"]} {"1": ["ab", "random_forest"], "0": ["nn", "svm"], "2": ["decision_tree"]} {"1": ["ab", "random_forest"], "0": ["nn", "svm"], "2": ["decision_tree"]}	Crash period(FN): {"0": ["ab"], "1": ["nn"], "2": ["random_forest", "decision_tree", "svm"]} {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree", "svm"]} {"2": ["ab"], "1": ["nn"], "0": ["random_forest", "decision_tree", "svm"]} {"0": ["ab"], "1": ["nn"], "2": ["random_forest", "decision_tree", "svm"]} {"0": ["ab"], "1": ["nn"], "2": ["random_forest", "decision_tree", "svm"]} non crash period(FP): {"1": ["ab"], "0": ["nn", "svm"], "3": ["random_forest"], "2": ["decision_tree"]} {"0": ["ab"], "1": ["nn", "svm"], "3": ["random_forest"], "2": ["decision_tree"]} {"1": ["ab"], "0": ["nn", "svm"], "2": ["random_forest"], "3": ["decision_tree"]}

	{"1": ["ab", "random_forest", "decision_tree"], "0": ["nn", "svm"]} {"1": ["ab", "random_forest"], "0": ["nn", "decision_tree", "svm"]}	{"0": ["ab", "random_forest"], "2": ["nn", "svm"], "1": ["decision_tree"]} {"2": ["ab", "random_forest"], "1": ["nn", "svm"], "0": ["decision_tree"]}	{"3": ["ab"], "0": ["nn", "svm"], "1": ["random_forest"], "2": ["decision_tree"]} {"3": ["ab"], "1": ["nn", "svm"], "0": ["random_forest"], "2": ["decision_tree"]}

Table A.4: The clustering outcomes for seven classifiers.