CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

When Crash Fault Tolerance Meets Machine Learning

Master's thesis in Computer science and engineering

GUSTAV HÄGER
JONATHAN KÖRE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

When Crash Fault Tolerance Meets Machine Learning

Gustav Häger
Jonathan Köre

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

When Crash Fault Tolerance Meets Machine Learning

GUSTAV HÄGER
JONATHAN KÖRE

Supervisor: Elad Michael Schiller, Department of Computer Science and Engineering
Examiner: Morteza Haghir Chehreghani, Department of Computer Science and Engineering

Master's Thesis 2022
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 2022

When Crash Fault Tolerance Meets Machine Learning

GUSTAV HÄGER
JONATHAN KÖRE
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Fault tolerance is vital for distributed systems as it allows them to operate in networks where nodes may experience failures. Several properties are of interest when considering fault tolerance, but we focus on safety *i.e.,* system consistency, and liveness *i.e.,* algorithmic progression. Many different distributed applications can be realized through various algorithms, some of which provide safety, but not necessarily liveness. In an algorithm that only provides safety, it is possible that nodes may stop responding, *e.g.,* if they experience crash failures. If this happens in an asynchronous system, it is impossible to know, by the well-known FLP impossibility result, if the node is crashed or if it is simply abnormally slow. Because of this, mechanisms are needed to circumvent the FLP impossibility in such systems. We study one such mechanism, the unreliable Failure Detector, which is an augmentation of the asynchronous model. For this, we consider systems in which it is costly to make mistakes *i.e.,* faulty suspicions. Particularly, we model failure detection as a binary classification problem through our simple and generic parameter model that utilizes both timed and time-free parameters, *i.e.,* those calculated by using clocks (or timers), and those calculated by counting round-trip completions in the system. By this, we answer our research question *can Machine Learning-based Failure Detectors balance the trade-off between the detection time, the probability of a faulty suspicion and the cost of a faulty suspicion better than existing solutions?* with an affirmative through a broad range of Machine Learning-based Failure Detectors, for which many classifiers serve as the basis. We also present a method to lower the probability of a false suspicion by analyzing the precision of the classifiers. Our results show that the learning task is suitable in a Federated Learning setting, where we use FedDyn, which is promising as it inherently implies scalability. We find that our best Failure Detector, which uses Random Forest, is able to lower the detection time by up to 86.6% in comparison to an existing solution whilst not making more mistakes.

Keywords: Unreliable Failure Detectors, Machine Learning, Distributed Systems, Federated Learning, Fault Tolerance, Asynchronous Systems, Consensus.

Acknowledgements

We would like to thank our supervisor, Elad Michael Schiller, for his ideas, words of encouragement, and tireless support throughout this project.
We would also like to thank our examiner, Morteza Haghir Chehreghani, for his helpful input and suggestions.

Gustav Häger, Jonathan Köre, Gothenburg, August 2022

Contents

Acronyms

AUC Area Under Curve. 11, 34, 38, 46

BDBD Blanchard, Dolev, Beauquier, Delaët. 3, 5, 6, 23, 29, 37, 39–45, 47–49, 51, 52, 55, 56

CDF Cumulative Distribution Function. 21, 23, 24, 34

EWMA Exponentially Weighted Moving Average. 14

FD Failure Detector. 1–6, 13–15, 18, 28, 31–35, 37–53, 55, 56
FL Federated Learning. 9, 10, 55

LSTM Long Short-Term Memory. 9, 33, 35, 43, 46, 52

ML Machine Learning. 2–5, 7, 9, 10, 17, 20–23, 31, 33–35, 37–43, 46–49, 52, 53, 55, 56

NN Neural Network. 8, 9, 33, 35, 43, 46, 47

PD Precision-Distribution. 3, 4, 23, 27–29, 35, 39–41, 43, 46–48, 52, 55, 56
PR Precision-Recall. 11, 34, 43, 47, 49, 52, 55

QoS FD Quality of Service. 6, 37–39, 41, 43, 45

RNN Recurrent Neural Network. 9
ROC Receiver Operating Characteristic. 11, 34, 38, 46
RTC Round-Trip Count. 5, 6, 18, 27–29, 40, 44, 45, 52
RTT Round-Trip Time. 19–23, 32, 33

SMR State Machine Replication. 1–3, 56
SVM Support Vector Machine. 8

TSH Time Since (last) Heard. 6, 18, 27–29, 40, 43–45, 52
TTF Timed and Time-Free. 3, 6, 23, 28, 29, 39–41, 43, 45, 51, 55

List of Figures

List of Tables

1

Introduction

In the world of distributed computing, it is important that messages sent from a node arrive correctly at its peers. Broadcast Protocols have been developed to ensure this. The weakest protocol that is used for fault-tolerant applications is a broadcasting primitive called Reliable Broadcast [1, 2]. As defined by Hadzilacos and Toueg:

> Reliable Broadcast requires that all correct [nodes] deliver the same set of messages (Agreement), and that this set includes all the messages broadcast by correct [nodes] (Validity) but no spurious messages (Integrity). [1]

Informally, fault-tolerant broadcast algorithms such as Reliable Broadcast are achieved by re-transmitting messages to correct nodes that still have not delivered the message. One problem that naturally arises from this is that it may be difficult to know whether a node has crashed or if it is just abnormally slow. In other words, ensuring liveness, *i.e.,* that the algorithm does not get stuck indefinitely, in the presence of crash failures may be non-trivial. This is however mainly a problem in asynchronous models as these models make no timing assumptions on the system, *e.g.,* message delays and relative speeds of nodes. In fact, this problem is well-known and was first presented as an impossibility for the consensus problem by Fischer et al. [3]. Typically, it is referred to as the FLP impossibility result.

In order to circumvent this impossibility, Chandra and Toueg propose unreliable Failure Detectors (FDs) as an augmentation of the asynchronous model [4]. An FD maintains a set of nodes that it suspects to be faulty and may add and remove nodes from this set as it sees fit. FDs as an abstraction are provided in different classes, which are characterized based on their properties. The classes presented by Chandra and Toueg considered crash failures as well as the properties *accuracy* and *completeness*. Accuracy states the leniency towards errors, and completeness that faulty nodes will eventually be suspected.

1.1 Problem Statement

In distributed systems, many different applications, such as State Machine Replication (SMR), can be realized through various algorithms, some of which provide safety *i.e.,* system consistency, but not necessarily liveness [2]. In such an algorithm

that only provides safety, it is possible that certain nodes may stop responding, *e.g.,* if they experience *crash failures*. If this happens in an asynchronous setting, it may lead to an unbounded memory requirement if one needs to save all the messages which must be sent to the node if it starts responding again so that it can 'catch up'. Because of this, a method is needed to detect the faulty nodes *i.e.,* the ones which the algorithm should not wait for.

In this work, we focus on FDs to accomplish this. Specifically, we model failure detection as a binary classification problem and implement FDs that use Machine Learning (ML) as the mechanism for failure detection.

If a node is incorrectly suspected, it will fall behind the rest of the system as it is believed to be crashed, and will thus not be sent any messages. This can incur a large cost in terms of time (and resources) when the mistake is to be corrected. Without a loss of generality, we consider SMR, in which a 'state transfer' must be made to correct the mistake. A state transfer essentially involves transferring the current state of the system to the incorrectly suspected node in order for it to 'catch up'. Thus, it is not desirable to raise faulty suspicions. However, it is also unfeasible to wait too long before suspecting a node as it halts progression. Therefore, there is a trade-off between the detection time, the probability of a faulty suspicion and the cost of a state transfer (in terms of time). By this, we ask the research question, *can ML-based FDs balance the trade-off between the detection time, the probability of a faulty suspicion and the cost of a state transfer better than existing solutions?*

1.2 System and Fault Model

For our evaluation, we consider systems of n nodes p_i where $i \in \{1, ...n\}$. We refer to this set of nodes as $\mathcal{P}$. In all systems, p_1 is assumed to never experience failures, but we assume that up to $f < n - 1$ of the other nodes can. Further, we assume that all FDs were deployed on p_1 to monitor the other nodes $p_j \in \mathcal{P} \setminus \{p_1\}$

We assume the presence of both omission and crash failures [2], which occur without any prior indication. However, we only consider the detection of crash failures.

1.3 Motivation and Related Work

The motivation behind this work is grounded in the fact that very limited research has been conducted on the topic of using ML to improve an FD's detection time.

The only example that we are aware of is a project report by Sozinov and Hammar in 2017 [5], where they attempted to use Linear Regression on RTT data to make a failure detector correctly suspect nodes quicker. Their results showed that their proposed failure detector, called MLFD, gave shorter detection times but had a higher mistake rate when suspecting nodes. We believe that we can undercover new results by modeling the problem as a binary classification problem, rather than a regression problem, and by investigating a large number of ML models.

The background to the motivation for this work includes a number of recent advances in the intersection between distributed computing and machine learning, for example [6, 7, 8]. The selection of some of the algorithmic components, such as the studied failure detector, was inspired by recent advances in the area of self-stabilization [9, 10, 11, 12, 13] and self-stabilizing Byzantine-tolerant systems [14, 15, 16, 17, 18].

1.4 The Contribution of this Work

We are studying an important part of distributed systems, which is the integration of ML into algorithms for unreliable FDs. In fully asynchronous systems where (crash) faults may occur, a strategy is required to circumvent the FLP impossibility [3] if one is to solve problems such as consensus (and, by extension, SMR) whilst maintaining both safety and liveness. This can *e.g.,* be accomplished by randomization [19] or through the use of an unreliable FD [4].

Throughout this work, we compare our results against an unreliable FD proposed by Blanchard *et al.* [20], which makes its suspicions based on round-trip completions. We refer to this as the Blanchard, Dolev, Beauquier, Delaët (BDBD) solution, after its authors. Our first contribution is a simple extension to the BDBD solution, called the Timed and Time-Free (TTF) model, which uses an additional temporal dimension when making suspicions. Upon evaluating the FDs based on their detection time and mistake rate, we find that the TTF model is consistently at least 19.5% faster than the BDBD solution at various mistake rates.

Further, we present a simple and generally applicable method of converting message-passing system data to parameters, which can be used for training ML models to solve the problem of failure detection through binary classification. We call this our parameter model, and it consists of both timed and time-free parameters, where a timed parameter is calculated by using clocks (or timers), and a time-free (round-trip-based) parameter is instead calculated through the number of round-trips completed in the system, thus not requiring the use of clocks (or timers), not even implicitly. It is shown that our best classifier, Random Forest, solves the problem with an accuracy of 0.999973 and a specificity of 0.999980.

In order to evaluate the detection time at various mistake rates for the ML-based FDs, we introduce a new module, called the Precision-Distribution (PD) module, which is used to further lower the mistake rate for an FD. Essentially, the PD module limits when an FD is allowed to raise a suspicion, based on mistakes it made during training. Through the use of the PD module, our results show that our best ML-based FD, the one based on the Random Forest classifier, is able to yield a detection time that is up to 86.3% lower than that of the BDBD solution, at a mistake rate of 1×10^{-6}.

Finally, we propose a solution in which the ML-based FDs not only utilize the PD module, but also the BDBD solution as a fallback, which ensures that crashed nodes are eventually suspected, regardless of what the ML classifier outputs.

We hope that our contributions will enable more efficient distributed systems, in which crashed nodes are detected faster and more accurately than they are with today's solutions.

1.5 Limitations

We now list important aspects to consider when interpreting our results.

Firstly, there exists a potential data leak upon the creation of data points for the ML models, meaning that the classifiers might have access to data that they would not have in production. This results in that the ML classifiers might appear to be better than they would be in production. Further detail can be found in Section 6.1.3.

Secondly, there are too few data points used for evaluation ($15.8{\times}10^6$) to be confident in the approximation of the observed distributions of results obtained for low mistake rates at, for instance, 1×10^{-5} or 1×10^{-6}. Also, relating to this, the results are specific to data collected from eight highly stable nodes during a relatively short period of time of a few weeks.

1.6 Structure

This work is structured as follows. In Chapter 2, we give relevant background on FDs and ML. Chapter 3 describes how our data was collected. Then, Chapter 4 explains our parameter model and the concepts relevant to it. Following this, Chapter 5 introduces our PD module, which is used for lowering the probability of falsely suspecting nodes. Chapter 6 then explains implementation details, such as how we construct our data set, and information relevant to the training, testing, and configuration of ML models. Finally, Chapter 7 explains the experiments and configurations set up for evaluating FDs, the results are shown in Chapter 8 and discussed in Chapter 9.

2

Methods

This chapter introduces and explains the methods used in this work. Particularly, Section 2.1 explains FDs along with their evaluation metrics. Section 2.2 briefly explains ML, the ML models used in this work, the classification metrics, as well as an optimization method for classifiers.

2.1 Failure Detectors

The unreliable Failure Detector (FD) abstraction was introduced by Chandra and Toueg [4] as an augmentation of the asynchronous model, in order to circumvent the FLP impossibility result for the consensus problem presented by Fischer *et al.* [3]. An FD either *trusts* a node (to be correct), or it *suspects* the node (to be faulty), at any given time. Chandra and Toueg propose different FD classes, characterized by the *accuracy* and *completeness* properties. Here, the accuracy is related to the mistakes that an FD is allowed to make, and the completeness that an FD must eventually suspect faulty nodes. We consider traditional FDs of class $\Diamond P$ which have *strong completeness* and *strong eventual accuracy*. This means that FDs of class $\Diamond P$ eventually suspect all faulty nodes and that they also eventually never suspect any correct nodes.

2.1.1 The BDBD Solution

We take interest in the FD of class $\Diamond P$ proposed by Blanchard *et al.* [20], which we refer to as the Blanchard, Dolev, Beauquier, Delaët (BDBD) solution (after its authors). The BDBD solution can be explained as follows. Consider a node $p_i \in \mathcal{P}$ that uses the BDBD solution as an FD to detect if any of the nodes that it is communicating with are faulty. For each node $p_j \in \mathcal{P} \setminus \{p_i\}$ that p_i is communicating with, the FD keeps a count of how many round-trips other nodes $p_k \in \mathcal{P} \setminus \{p_i, p_j\}$ have completed with p_i since p_j last completed a round-trip with p_i. In this project, we call this count the Round-Trip Count (RTC). If the RTC of node p_j reaches a pre-defined constant Θ, then it is suspected by the FD. Similarly, Beauquier and Kekkonen-Moneta [21] proposed FDs of class $\Diamond P$ which suspects a node if it fails to respond in the same time it takes another node to respond m times.

2.1.2 TTF Model

We created a model that can be seen as an extension of the BDBD solution which was explained in Section 2.1.1. This solution, called the TTF model, uses a timer for measuring how long ago it was since a node $p_j \neq p_i$ last completed a round trip with p_i, where $\{p_i, p_j\} \in \mathcal{P}$. We refer to the value of the timer as Time Since (last) Heard (TSH). The TTF model then makes suspicions based on both the RTC and TSH of a given node. Specifically, it uses a monotonically increasing function $f(\texttt{RTC}, \texttt{TSH}) \rightarrow \{trust, suspect\}$. It is monotonically increasing in the sense that once the FD starts suspecting, it will do so forever, or until the RTC and TSH are reset again as a result of the suspected node responding. In this project, the function was based on statistics of the collected data, and this is further explained in Section 7.3.1.3.

2.1.3 Quality of Service Metrics

The FD Quality of Service (QoS) metrics, introduced by Chen *et al.* [22], are used to measure the *speed* of an FD, *i.e* how fast it detects failures, as well as its *accuracy*, *i.e* how well it avoids mistakes. The QoS metrics are derived from the durations of an FD's outputs, *i.e.,* the periods when the FD either suspects or trusts a node. These durations are measured with S- and T-transitions, where an S-transition is performed when the FD changes its output from `trust` to `suspect`, and a T-transition when the output is changed from `suspect` to `trust`. In the coming description of the used QoS metrics, we define a mistake, like Chen *et al.* [22], as an incorrect suspicion, as we also consider crash-free executions when measuring the accuracy metrics.

The following QoS metrics were considered for this work. We note that the First Detection Time (T_{D1}) is a metric provided by us, and is thus not found in Chen *et al.*'s work.

Speed Metrics

Detection Time (T_D): The time from when a node crashes to when it is permanently suspected, *i.e*, the time until the last S-transition, where after no more T-transitions occur.

First Detection Time (T_{D1}): The time from when a node crashes to when it is first suspected, *i.e*, the time until the first S-transition occurs.

Accuracy Metrics

Average Mistake Rate (λ_M): The probability of making a mistake per time unit, *i.e*, the average number of S-transitions per time unit.

Query Accuracy Probability (P_A): The probability that the failure detector's output is correct at any given time.

2.2 Machine Learning

This section briefly describes the concept of ML, as well as the classifiers used in this work. Since, to the best of our knowledge, fault detection as a binary classification problem is novel, we opt to investigate multiple off-the-shelf classifiers instead of focusing on optimizing a select few. We also describe metrics used for evaluating a classifier, as well as an optimization technique applicable to most classifiers.

2.2.1 The Concept of ML

ML algorithms are used to approximate or "learn" a mathematical function. The learned function is known as the ML model and is in the context of this project learned through supervised learning. In supervised learning, the model is trained with an algorithm by subjecting it to input data or "data points", along with a label indicating what the data represents. The aim of the algorithm is therefore to learn how to map the data points to the labels.

2.2.2 Decision Trees and Random Forest

The Decision Tree is a classifier that reaches its decision through the use of a tree, which is strategically constructed upon training. Starting at the root, the tree is traversed by asking yes and no questions about the parameter values of a data point until a leaf node is hit. By looking at the classes of the data points that ended in that leaf node during training, a decision can be made based on the majority class.

The Random Forest classifier constructs multiple Decision Trees during training, and then makes its decision based on the trees' majority vote. For more information about Decision Trees and Random Forest, we refer to chapters 4 and 5 of [23].

2.2.3 AdaBoost

Boosting is a strategy in which *weak learners* are combined into a *strong learner*, where a weak learner is one that is necessarily only slightly better than random guessing and a strong learner one that is arbitrarily accurate. AdaBoost, an adaptive boosting strategy, was introduced by Freund and Schapire [24]. In AdaBoost, T weak learners are trained into classifiers, one at a time. Based on the performance of classifier t, the weights of the training data are adjusted *s.t.* greater emphasis is placed on points that t miss-classified before the training data is used to train classifier $t + 1$. Intuitively, this means that each weak learner is trained to correctly classify data points that their predecessor performed poorly on. A classifier is yielded by weighting the predictions from the T weak learners based on their performance.

2.2.4 Logistic Regression

In Logistic Regression, a logistic function is fitted to a data set through the use of Stochastic Gradient Descent [25, 26], which is an algorithm commonly used in deep learning. The output is bound between 0 and 1 and represents the predicted probability that a data point belongs to either class A or B. For more information on Logistic Regression, we refer to chapter 3 of [23].

2.2.5 Naive Bayes

The Naive Bayes classifier makes its decisions based on the premise that all parameters of a data point are independent. Based on this independence, the classifier uses Bayes' Theorem to decide what class y a new data point $\mathbf{x}$ is most likely to belong to, based on observations made during training. Specifically, a binary classification is made as in Equation 2.1. For more information on Naive Bayes, we refer to the overview by Vikramkumar *et al.* [27].

$$\begin{aligned} \texttt{Predicted Class} &= \underset{y \in \{0,1\}}{\arg\max}\ P(y \mid \mathbf{x}) \\ P(y \mid \mathbf{x}) &= \frac{P(\mathbf{x} \mid y)P(y)}{P(\mathbf{x})} = \frac{P(x_1 \mid y)P(x_2 \mid y)...P(x_n \mid y)P(y)}{P(x_1)P(x_2)...P(x_n)} \end{aligned} \tag{2.1}$$

2.2.6 Support Vector Machine

An Support Vector Machine (SVM) maps input vectors (data points) of two classes to points in space and creates a multi-dimensional plane, called a hyperplane, between these points, which serves as the decision boundary. The concept was introduced by Cortes and Vapnik, originally under the name 'Support Vector Networks' [28]. The hyperplane is created in such a way that it maximizes the distance from it to the closest points of each class. As such, the SVM does not save all vectors used for training, but only the ones relevant for the construction of the hyperplane. These points are referred to as 'support vectors'. During classification, input is mapped to a point in space, and the point's relation to the hyperplane decides which class it belongs to, where each side of the hyperplane corresponds to one of the two classes.

2.2.7 Artificial Neural Network

Artificial Neural Networks are the basis of deep learning and are inspired by how learning occurs in the human brain. The main building block of the Neural Network (NN) is the neuron, which is used to represent a value. Neurons are arranged in layers and a neuron is linked to other neurons in the previous and next layer through weights. Figure 2.1 shows a simple NN, with one input layer, one hidden layer, one output layer, and respective weights connecting the layers. An input vector $\mathbf{x}$ is fed to the neurons in the first layer and is then propagated through the network by applying a mathematical function for each neuron. The value of each neuron is

calculated based on the weights $w_{i,j}^{l}$ and values of neurons in the previous layer, plus an additional bias, b_j, associated with the neuron. An additional activation function g, is also applied to this value. Equation 2.2 shows an example of how neuron h_j in the hidden layer is calculated. When the input vector has propagated through the network, a classification can be made by looking at the values of the neurons in the output layer $\mathbf{y}$. In the context of a network used for binary classification, the output layer can consist of a single neuron, representing the probability that the input vector belongs to class A or B. For more information on Artificial Neural Networks, we refer to chapter 7 of [23].

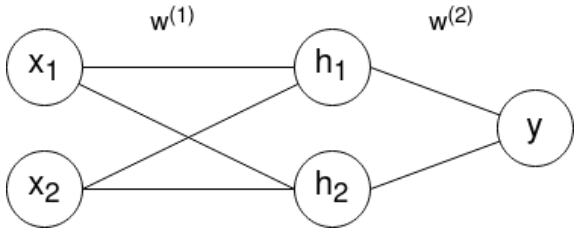

Figure 2.1: A simple neural network with three layers.

$$h_j = g(\sum_{i=1}^{2} w_{(i,j)}^{(1)} x_i + b_j) \tag{2.2}$$

2.2.8 Long Short-Term Memory

The Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN), which is an NN architecture in which the output of one prediction, is fed back into the network when making the next prediction. The LSTM architecture is specially designed towards retaining relevant information in memory for long periods of time, which is something that earlier RNN architectures have had difficulty with [29]. This makes the LSTM well suited for temporal applications, and LSTMs have shown to be successful in for instance time series prediction and natural language processing [30].

2.2.9 Federated Learning by FedDyn

In traditional ML, the models are trained on one machine, with a centralized data set. Federated Learning (FL) is instead used to train ML models on multiple nodes, called workers, in a distributed fashion, with local, non-shared, data sources [31]. FL can significantly speed up the training for large data sets, as multiple machines are used for computation. FL also enables the training to be conducted without directly needing to share data, thus eliminating the need for a centralized data set. This is typically done in an iterative process, in which the distributed nodes first train a model locally, to then send the learned model parameters to a centralized server where the parameters from the nodes are combined into a global model, which is then sent back to the distributed nodes. Sending only the parameters also saves

a considerable amount of bandwidth compared to sending all the collected data to a centralized server.

We select to use the state-of-the-art FL model FedDyn [32] in this project. FedDyn is strong in the sense that it does not make any assumptions about the heterogeneity of the data between nodes, can handle unbalanced data, a large number of nodes, partial participation of nodes, and has overall low transmission costs of data from the distributed nodes to the centralized server.

2.2.10 Classification Metrics

ML classifiers are often evaluated based on metrics derived from a confusion matrix. A confusion matrix, shown in Figure 2.2, is used for categorizing the predictions that a classifier makes, based on whether they were correct or wrong.

From the confusion metric results, other metrics can be derived. We take particular interest in the derived metrics shown in Table 2.1.

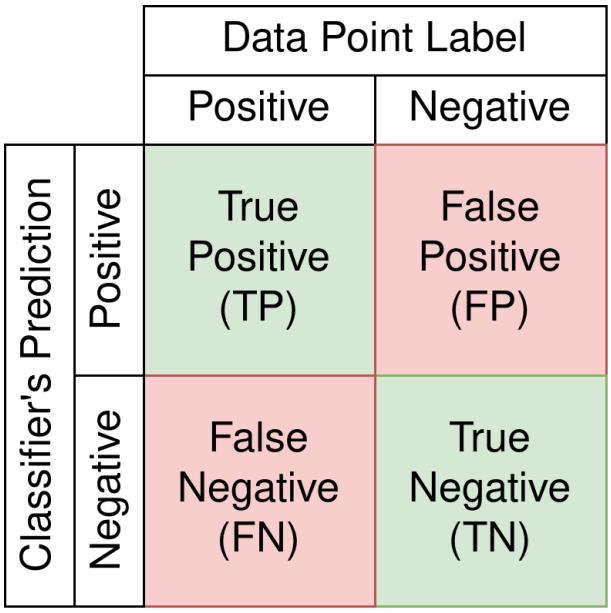

Figure 2.2: A confusion matrix.

Metric Name	Derivation	Description
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$	The fraction of data points that were correctly classified.
Sensitivity	$\frac{TP}{TP+FN}$	The fraction of positive data points that were correctly classified.
Specificity	$\frac{TN}{TN+FP}$	The fraction of negative data points that were correctly classified.
Precision	$\frac{TP}{TP+FP}$	The fraction of positive classifications that turned out to be correct.
Fall-Out	$\frac{FP}{FP+TN}$	The fraction of negative data points that were incorrectly classified. Can also be written as 1 - Specificity.

Table 2.1: Metrics derived from a confusion matrix.

2.2.11 The Discrimination Threshold and Curve Analysis

Most binary classifiers use a threshold, known as the *discrimination threshold*, when deciding what to classify a data point as. An intuitive threshold to use for a classifier that outputs a probability is 0.5, meaning that a data point X will be classified as class A if $P(A \mid X) \geq 0.5$, and class B otherwise. By lowering or increasing this threshold, the classifier's bias towards each of the two classes is changed, where 0.5 indicates no bias.

By analyzing how the metrics described in Section 2.2.10 change as the threshold is varied, one can gain further insights into the classifier's performance. Specifically, these insights regard a classifier's ability to predict probabilities of what class a data point belongs to, in contrast to the metrics in Section 2.2.10 which only considers the final, binary, class predictions.

We take interest in two methods for this approach, namely by analysis of the Receiver Operating Characteristic (ROC) curve [33], and the Precision-Recall (PR) curve [34]. We note that Recall is another name for Sensitivity. Both curves are created by varying the threshold from 0 (predicts all as positive) to 1 (predicts all as negative). For the ROC curve, the y-axis represents Sensitivity and the x-axis Fall-Out. The PR curve instead uses Precision as its y-axis, and Recall as its x-axis. Once these curves are constructed, a threshold can be selected from them in order to optimize towards desired metrics. We indicate that a classifier has been optimized in this fashion by suffixing its name with either PR or ROC.

After the curves are constructed, one can calculate the Area Under Curve (AUC) where the area of any curve is between 0 and 1. For the AUC ROC, a higher score means that the classifier is perfectly skilled at separating the two classes. The AUC PR reflects the classifier's ability to correctly classify the positive data points, without taking correctly classified negative data points into account. However, we only use the AUC ROC as an evaluation metric, and not the AUC PR, as it is more

widely adopted.

3

Data Collection

In this chapter, we describe the system from which the data was collected, in Section 3.1. We provide information about the relevant log events in Section 3.2. Finally, we describe the simulator used for "playing back" the collected logs in Python in Section 3.3, and synthetic crash failures in Section 3.4.

3.1 The System

In order to increase the applicability of the results of this project, we want the system, from which the data is collected, to be similar to a system found in a real distributed application setting. Therefore, we implemented and deployed a best-effort broadcasting algorithm that utilizes an FD for detecting faulty nodes.

For this system, it is assumed that a more general fault than crash-failures can occur, namely muteness failures. Specifically, muteness failures depict any type of lack of response from faulty nodes. For example, an adversary might decide not to respond to specific messages sent by correct nodes. This can cause a correct node to wait in vain for responses. The behavior of the aforementioned faulty nodes is often modeled as Byzantine failures [35], which are manifested by an arbitrary deviation from the algorithm code. The challenge of defending against Byzantine failures is often addressed by assuming that there are at most t Byzantine nodes. The best-effort broadcasting algorithm [14] and FD of class $\Diamond P_{mute}$ by Duvignau *et al.* [14] will now be briefly explained alongside the testbed used for the deployment of the application.

3.1.1 Broadcasting Algorithm

The best-effort broadcasting algorithm works on the basis that a broadcasting node p_i attempts to send a message containing a value to other nodes in the network. When a message has been transmitted to a node, the algorithm expects an acknowledgment, confirming the transmission. Due to the assumption that there can be nodes that are mute to p_i, this confirmation can not be guaranteed. The algorithm, therefore, asks $\Diamond P_{mute}$ which nodes are suspected to be mute to p_i, and which are to be trusted. The algorithm then attempts to re-transmit the message as long as there is a node p_j that is trusted and still has not acknowledged once. This re-transmission is made to all nodes, and not just to the ones that have not yet ac-

knowledged. The reason for this is that $\diamond P_{mute}$ requires this functionality in order to make decisions. We used the Exponentially Weighted Moving Average (EWMA) of observed RTTs for deciding the message re-transmission frequency. This method allows the re-transmission timeout for a node to match the expected RTT for that specific node. In other words, the broadcaster does not re-transmit a message before a node is expected to reply. EWMA is different compared to normal averaging in the fact that more recently observed RTTs are weighted heavier towards the average, which allows for a quicker adaptation to changes in RTTs.

3.1.2 Failure Detector $\diamond P_{mute}$

$\diamond P_{mute}$ works by suspecting nodes p_j which have not completed one single round trip with p_i in the same time that p_i has completed Θ round trips in total with the $n - t$ nodes that have completed the fewest round trips with p_i. The top t most responsive nodes are not included towards the sum of the round trips as they are considered possibly Byzantine and could have sent preemptive messages to p_i in an attempt to cause $\diamond P_{mute}$ to raise suspicion of correct nodes. A detailed explanation of $\diamond P_{mute}$ is provided by Duvignau *et al.* [14]

3.1.3 Broadcasting Rounds

As previously explained, a broadcaster broadcasts a message with value v_i to all nodes trusted by the FD. In order to keep the system running for an indefinite amount of time, the broadcaster is queried once every second in an attempt to broadcast a message with a new, unique, value $v_{i+1} \neq v_i$ to its trusted nodes. If the previous value v_i has not been delivered by the broadcaster, and thus also all trusted nodes, upon this invocation, the broadcaster responds by stating it is busy and another query with v_{i+1} is performed a second later. However, once a value v_i has been delivered by all trusted nodes and the broadcaster, we say that the broadcasting task is complete. Each period of time between a successful query to the broadcaster, and the completion of the corresponding broadcasting task, is referred to as a broadcasting round.

We note that the concept of a broadcasting round is only used in this context *i.e.,* the broadcasting application, and that our FDs do not take them into account, nor is the time between them considered.

3.1.4 Deployment

We deployed the system on PlanetLab Europe, a distributed systems testbed that is available for researchers [36]. Each academic institution or industrial research lab connected to PlanetLab Europe must provide a node on which other PlanetLab users can connect and deploy their software. At the time of the deployment of our system, only 9 nodes were available to deploy on, of which we used 8. These nodes were hosted at universities located in Göteborg (SWE), Linköping (SWE), Cambridge (GBR), Vancouver (CAN), Prague (CZE), and Modena (ITA). The universities in Cambridge and Prague provided two nodes each which we used.

In order to collect as much data as possible from the limited set of nodes, we made 8 separate deployments. Each node acted as a broadcaster in one of the deployments, and as a client in the remaining ones.

In the deployments where a node in either Cambridge or Prague was used as a broadcaster, the second node at the same university was excluded from being a client, since internal communication within the same university was extremely quick, which caused the collected log data to grow in size at a rate which was too high to be supported by our system.

3.2 Collection of Logs

All nodes continuously log their events to file, however, the logs of interest are those of the broadcasting node in each deployment. The logged events of relevance to this project are the sending and receiving of messages and the beginning and ends of broadcasting rounds. Each event is attached with a timestamp of when it occurred, and message events are attached with a unique message id so that the RTT of each message can be calculated from the logs.

3.3 Simulator

In addition to the deployed system, we also created a simulator in Python which is used to "play back" the collected logs. The simulator is used for the creation of data points (described in Section 4), and for evaluation so that our proposed FDs can be evaluated without having to be implemented on the deployed systems.

3.4 Synthetic Crash Failures

It turned out that the nodes on PlanetLab were highly stable, and crashed very rarely (if ever). Therefore, we instead opted to simulate the crashes through synthetic crash failures. These crashes are referred to as crash intervals, and they were introduced into the simulator after the logs had been collected.

Upon "play-back" of the collected logs, any messages received from node p_i during a period of time in which p_i is synthetically crashed are ignored by the broadcasting algorithm. This results in the broadcaster experiencing a lack of response from the currently 'crashed' node. Each crash interval lasted at least 3 minutes and was triggered by sampling from a Poisson distribution which had an expectation of a crash once every ten minutes.

4

Parameter Model

In distributed systems, there is typically a plethora of information that one may acquire. Depending on the selected communication protocol, and an application's access to a system, this information can fluctuate. We are interested in modeling the available information to construct parameters that may be used as input for ML and analytical models. It is possible to create a very complex parameter model, *e.g.,* by acquiring information from a transport-layer protocol such as TCP, or from an internet-layer protocol such as ICMP. However, by doing this, one highly restricts the applicability of the parameter model as a system must be able to accommodate it. Because of this, we take the approach of constructing our parameter model based on information that should be available to most message-passing applications: messages sent and received, along with the corresponding time of each action.

This chapter describes the input parameters used for the ML and analytical models. First, the categories of parameters are described in Section 4.1. Then, the concept of windows, which are used for the computation of the parameter values, is introduced in Section 4.2. Methods of data normalization are explained in Section 4.4. Finally, a list of our parameters is presented in Section 4.5.

4.1 Parameter Categories

Every parameter belongs to a set of categories. We will here define these categories. In order to see examples of the different parameters, refer to Section 4.5 where we have tagged each parameter with the categories they belong to.

4.1.1 Individual or Grouped

A parameter is individual if it can be constructed by using data relating to one node, and one node only. For instance, how long ago was it since we heard from node p_i? The opposite of this would be a grouped parameter, which is based on data of multiple nodes. For example, how many round trips has the broadcaster been able to complete with others since p_i was last heard from?

4.1.2 Timed or Time-free

A parameter is either timed or time-free (round-trip based). A parameter is said to be timed if clocks (or timers) are used for calculating the parameter value (even

if it is just indirectly). Time-free parameters do not use clocks or timers (not even indirectly) and are instead calculated by counting round-trips completed in the system. Examples of timed and time-free parameters are the Time Since (last) Heard (TSH) and Round-Trip Count (RTC) parameters, respectively. We note that the time-free parameters are based on the works of Duvignau *et al.* [14], Beauquier and Kekkonen-Moneta [21], and Blanchard *et al.* [20], who used this parameter type in perfect failure detectors.

4.1.3 Raw or Normalized

A parameter is said to be normalized if the parameter value is guaranteed to be in the range of $[-1, 1]$. The normalization methods used are described in Section 4.4. The parameter is instead said to be raw if it is not normalized. We note that TSH and RTC are the only raw parameters, and the remaining parameters are normalized. In the tables of Section 4.5, we indicate that a parameter is either raw or normalized through the 'Normalization' column. If 'None' is given, then the parameter is raw, otherwise, it is normalized with the given method.

4.2 Timed and Time-free Windows

For the sake of a simpler analysis, we assume that during the execution of the studied protocol, the failure detector is queried at a set time interval, say, every δ time units. From the classifier's point of view, each such query is a point in time in which the system state is sampled and recorded. We call such a sample a *timed sample*. We define the fixed, non-sliding (timed) window, W_{timed}, to be a time-series of $|W_{timed}| = \Lambda$ timed samples, which the Failure Detector (FD) uses either for training or for replying to the user queries. We note that during the bootstrapping phase of the mechanism for fault detection, the mechanism cannot report suspicions before it has processed at least a complete window W_{timed} of Λ samples.

We will now introduce time-free windows. To understand the contents of time-free windows, we must introduce round-trip counters, which are based on the definition in Algorithm 4 in [14] and the BDBD solution [20]. A round-trip counter is a vector, specific to node $p_i \in \mathcal{P}$, in which each entry corresponds to the number of round-trips that nodes $p_j \in \mathcal{P} \setminus \{p_i\}$ have completed since p_i last completed a round-trip. For each node $p_i \in \mathcal{P}$, the system maintains a round-trip counter, which is reset every time p_i completes a round-trip. We define a *complete* round-trip counter as the state that the round-trip counter was in just before it was reset, *i.e* the number of round-trips that other nodes $p_j \in \mathcal{P} \setminus \{p_i\}$ completed while p_i completed one round-trip. We now define the fixed, non-sliding, time-free window $W_{timefree,i}$ to be a sequence of the last observed $|W_{timefree,i}| = \Gamma - 1$ complete round-trip counters of p_i, with the addition of p_i's current round-trip counter which is being maintained by the system (and is thus not complete yet). Similarly to W_{timed}, we note that during the bootstrapping phase of the mechanism for fault detection, the mechanism cannot report suspicions for p_i before it has processed at least a window $W_{timefree,i}$ of $\Gamma - 1$ complete round-trip counters.

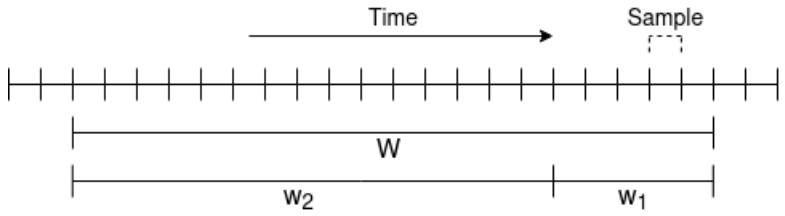

Figure 4.1: A window W with a size of 20 samples, along with two nested windows w_1 and w_2 which cover the most recent 5 and 15 samples, respectively. Note that the system does not know about the samples outside of W. This is added for the illustrative purpose of how W relates to time.

4.2.1 Nested Windows

Sometimes statistical calculations need to use window nesting. We define the fixed, non-sliding, timed nested window w_{timed} of $\lambda < \Lambda$ samples to be a series of λ consecutive samples in W_{timed}. We also define, for a given node $p_i \in \mathcal{P}$, the fixed, non-sliding, time-free nested window $w_{timefree,i}$ of $\gamma < \Gamma$ round-trip counters to be a series of γ consecutive round-trip counters in $W_{timefree,i}$. When two nested windows share at least one sample or round-trip counter, we say that they are overlapping. Similarly, if they share no samples or round-trip counters, we say that they are non-overlapping.

Further, we say that a window is uniformly partitioned if all samples (or round-trip counters) contained in it are evenly distributed in non-overlapping nested windows. We define $w_{timed,part}$ as a partition of the uniformly partitioned W_{timed}, and $w_{timefree,i,part}$ as a partition of the uniformly partitioned $W_{timefree,i}$. Figure 4.1 shows an example window W with a size of 20 samples, and two nested, non-overlapping, windows w_1 and w_2. Note that window W is not uniformly partitioned in this example as the nested windows have different sizes.

4.2.2 Samples

This section explains what is sampled in the system. We note that the concept of a sample only exists for W_{timed}, but one could regard a round-trip counter as a sample in $W_{timefree,i}$.

We begin by listing the contents of one timed sample. We refer to each piece of content as a parameter. A sample contains system state data for all nodes $p_i \in \mathcal{P}$. The parameters specific to p_i are the following:

- A sequence of observed Round-Trip Times (RTTs) for p_i since the previous sample was taken.
- The number of sent algorithm messages to p_i since the previous sample was taken.
- The number of received algorithm messages from p_i since the previous sample was taken.

- The time since we last heard from p_i.

We say that, in a timed sample, the RTT is the time, in milliseconds, that it takes for a node $p_i \in \mathcal{P}$ to respond, according to the studied protocol, to a message sent to it. Each sample contains a sequence of all RTTs recorded since the system was last sampled.

We say that the *number of sends* and *number of receives* in a timed sample are the numbers of sent and received messages, according to the studied protocol, of a node $p_i \in \mathcal{P}$ since the system was last sampled.

At the time of sampling, we say that the *time since last heard* is the time, in milliseconds, since node $p_i \in \mathcal{P}$ last completed a round-trip, according to the studied protocol. This parameter does not reset when the system is sampled.

4.3 Value Aggregation

Sometimes, we need to perform operations on a grouping of values in order to calculate the parameter values, *e.g.,* when we want the average or variance of observed RTTs. We call these operations aggregation functions. This section explains the different aggregation functions that we use for data in windows. Each function takes a vector `V` and returns an aggregate of the vector. We use the notation `m(V) → aggregate` to describe that a function `m` takes vector `V` and returns the aggregate described by `aggregate`. Note that it can occur that `V` is empty, in which case the aggregation function is undefined. This case only affects the RTT-Based Parameters in Section 4.5.1 however, and the mitigation strategy is described there.

`avg(V) → the average of V`

`median(V) → the median of V`

`min(V) → the minimum of V`

`variance(V) → the variance of V`

`max(V) → the maximum of V`

`skew(V) → the skew of V`

`sum(V) → the sum of V`

`ratio_above(V, t) → The ratio of elements in V which are above a threshold t`

`vector_sum(V, d) → The sum of a multidimensional vector V along dimension d`

`vector_avg(V, d) → The average of a multidimensional vector V along dimension d`

`vector_avg(V, d) → The average of a multidimensional vector V along dimension d`

4.4 Normalization

This section describes the methods used for the normalization of the parameters that are used in the ML models.

The normalization techniques used in this project ensure that the parameter values are between $[-1, 1]$. We have also aimed at achieving a mean of zero for these

parameters, but we can not guarantee that this is the case. Cumulative Distribution Function (CDF) and Trend describe the methods that we use to achieve the normalization. We also note that the parameter named `Receive Rate` does not use these normalization techniques, but is already inherently normalized between $[0, 1]$ as a direct result of its aggregation function.

We will now explain how a histogram is constructed for a window, and consecutively how the CDF of the histogram is used to produce parameters for the ML model. For a given window W, timed or time-free, a histogram is created from data contained in it. The histograms used in this project are all created by partitioning W into $m \geq 2$ partitions, where each partition contains $\frac{|W|}{m}$ samples (or round-trip counters). Note that m can be different for different histograms. Each partition then contributes with one entry to the histogram, where the entry is created from the values of the samples (or round-trip counters) contained in the partition. When a parameter for the ML model is to be constructed, the CDF of the histogram is queried with a value x, which has been extracted from the most recent partition in the window. To ensure that the parameter value v is in $[-1, 1]$, it is set by $v = 2 \cdot CDF(x) - 1$.

We will now present an example of how a histogram can be produced from a timed window W. In the example, we produce a histogram of average RTTs for node p_i. Window W and its partitions w_i are shown in Figure 4.2. W has a size of 20 samples and each partition covers two of these samples. Figure 4.3 then shows how the histogram is built based on the data contained in the samples that each partition covers.

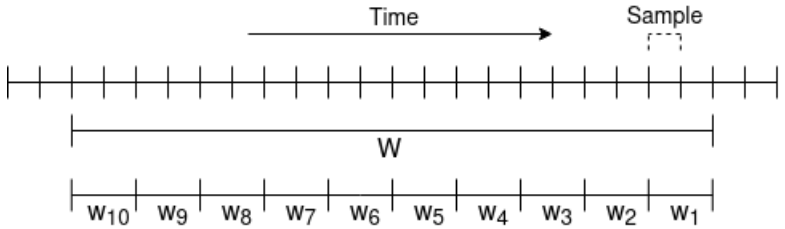

Figure 4.2: A window W with a size of 20 samples, partitioned to consist of ten partitions w_i, which cover two samples each.

Avg RTT per partition	
Partition	**Avg RTT**
1	22.4
2	26.6
3	24.5
4	23.0
5	23.2
6	24.1
7	24.9
8	24.4
9	25.1
10	25.7

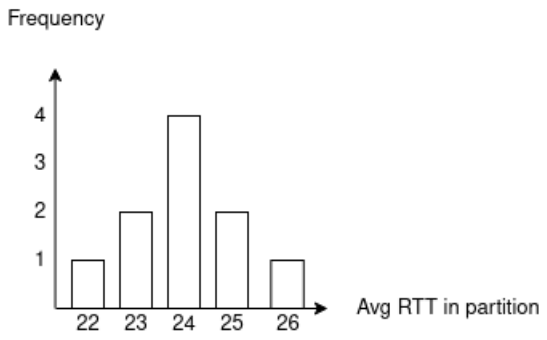

Figure 4.3: An example illustrating how a histogram is produced from the configuration shown in Figure 4.2, given that the samples contain data on RTTs.

A Trend is a measurement of how data differs between nested windows w. As an illustrative example, assume, without a loss of generality, a window W of Λ samples. Now assume s nested windows, each covering $|w_i| = \lambda_i < \Lambda$ values in W, where $0 < i \leq s$. Now, we arrange these windows such that w_i covers λ_i values in W, which are more recent than at least some of the values in w_j, if $i < j$. This allows for the construction of nested windows which may *e.g.* represent the most recent 10 and 100 samples in W, or the most recent 10 samples and the 10 samples preceding those. Comparisons are performed by passing the relevant data from each nested window w_i and each nested window w_j where $i < j$ to an activation function.

The activation function is based on a *tanh* function and guarantees that a value in the interval $[-1, 1]$ is returned. Equation 4.1 shows the activation function and how the Trend value is calculated from the aggregates of windows w_i, w_j. In the equation, m refers to an arbitrary aggregation function. The activation function has the property that values in the interval $(0, 1]$ represent an increase between windows, where a larger increase is indicated by a higher value. Similarly, values in the interval $[-1, 0)$ represent a decrease between windows, where a larger decrease is indicated by a lower value. A value of 0 signifies no change between windows. The output of this activation function is then used as the parameter value.

$$f(x = m(w_i), y = m(w_j)) = \begin{cases} 1 & \text{if } y = 0 \\ -1 & \text{if } x = 0 \text{ and } y \neq 0 \\ \tanh(\log(x/y)) & \text{otherwise} \end{cases} \tag{4.1}$$

As an example, consider once again the window configuration shown in Figure 4.1. A Trend, in this case, would be to pass an aggregation from w_1 and w_2 to the activation function in order to get an indication of how the values change between the nested windows. One could for instance look at the averages or the minimum values of RTTs.

It should be noted that all parameters that will be used directly in the training of the ML models will be normalized.

Raw parameters are instead used for the PD module described in Chapter 5, in order to further help the ML models avoid falsely suspecting nodes. The raw parameters are also used for the BDBD solution and the TTF model.

4.5 Parameter Definitions

This section defines the parameters for the data points by using the categories defined in Section 4.1, the windows defined in Section 4.2, the aggregation functions described in Section 4.3, and the normalization methods defined in Section 4.4.

We have grouped the parameters based on their input. As a recap from Section 4.4, the histogram used for each CDF parameter is constructed from one value per partition, given that a window W has been uniformly partitioned. If a parameter is a CDF parameter, then the CDF of the value from the most recent partition (chronologically), is set as the parameter value. Note that each CDF parameter uses a different histogram, specifically created to be used for that parameter. For Trends, partitions are not required, as a Trend compares nested windows of possibly varying sizes. If a parameter is a Trend parameter, the most recent nested window (time-wise) is compared to older windows.

Parameters are always specific to one node. As an example, the parameter *Average RTT CDF* is based on RTTs for one node only, which means that a parameter value is created for each node.

We now want to give the reader an idea of what we believe are reasonable sizes for windows, nested windows, and partitions. We propose window sizes of $|W_{timed}| = \Lambda = 100$ samples and $|W_{timefree,i}| = \Gamma = 1000$ round-trip counters. For W_{timed}, we consider three non-overlapping nested windows $w_{timed,1}$, $w_{timed,2}$, and $w_{timed,3}$ of $\lambda_1 = 10$, $\lambda_2 = 40$, and $\lambda_3 = 50$ samples, which cover all Λ samples of W_{timed}. For W_{timed}, we consider a partition size of $\lambda_{part} = 1$ samples, yielding a total of 100 partitions $w_{timed,part}$. Similarly, for $W_{timefree,i}$, we consider three non-overlapping nested windows $w_{timefree,i,1}$, $w_{timefree,i,2}$, and $w_{timefree,i,3}$ of $\gamma_1 = 10$, $\gamma_2 = 90$, $\gamma_3 = 900$ round-trip counters, which cover all Γ round-trip counters of $W_{timefree,i}$. Further, for $W_{timefree,i}$, we consider a partition size of $\gamma_{part} = 10$ round-trip counters, yielding a total of 100 partitions $w_{timefree,i,part}$

4.5.1 Parameters Based on RTTs

The parameters in this section are based on the observed RTTs within W_{timed}, and they are listed in Table 4.1. The parameter values are calculated based on the observed RTTs in either the most recent partition or nested window, depending on whether the parameter is normalized through CDF or Trend, respectively. We denote the value of relevance, in this case, the observed RTTs, as $\mathtt{V}$, which we say is a multi-dimensional vector, where each coordinate i corresponds to the RTTs observed for node $p_i \in \mathcal{P}$. The aggregation functions, shown in column *Aggregation*, were defined in Section 4.4. We note that our method of applying different aggregation functions on a sequence of observed RTTs is inspired by the project report of Sozinov

and Hammar [5].

As mentioned in Section 4.3, V_i can be empty if no completed round-trips are observed within a given window w_{timed} or $w_{timed,part}$. The default values that were used for the aggregation functions, in this case, were 0 for variance and skew. For median, avg, min, and max, it was supposed to be set to $\delta \cdot \lambda_{part}$, but was instead set to $\delta \cdot \lambda_1$ by mistake. Setting it to this value might have a negative impact on the affected parameters, but changing it was not possible as it was discovered too late into the project.

Parameter Name	Aggregation	Normalization	Categories
Average RTT CDF	`avg(V_i)`	CDF	Timed Individual
Minimum RTT CDF	`min(V_i)`	CDF	
Maximum RTT CDF	`max(V_i)`	CDF	
Median RTT CDF	`median(V_i)`	CDF	
RTT Variance CDF	`variance(V_i)`	CDF	
RTT Skew CDF	`skew(V_i)`	CDF	
Average RTT Trend	`avg(V_i)`	Trend	
Minimum RTT Trend	`min(V_i)`	Trend	
Maximum RTT Trend	`max(V_i)`	Trend	
Median RTT Trend	`median(V_i)`	Trend	
RTT Variance Trend	`variance(V_i)`	Trend	
RTT Skew Trend	`skew(V_i)`	Trend	

Table 4.1: Parameters based on RTTs.

4.5.2 Parameters Based on Sent and Received Messages

The parameters in this section are based on the observed numbers of sent and received algorithm messages within W_{timed}, and they are listed in Table 4.2. The normalized parameter values are calculated based on the number of sent or received algorithm messages in the most recent partition or nested window, depending on whether the parameter is normalized through CDF or Trend, respectively. The Receive Rate is calculated from a nested window w_{timed}, and as such, there is one Receive Rate parameter corresponding to each nested window w_{timed}. They do not need further normalization as their aggregation function inherently normalizes the value.

Parameter Name	Aggregation	Normalization	Categories
Sent Messages CDF	None	CDF	Timed Individual
Sent Messages Trend	None	Trend	
Received Messages CDF	None	CDF	
Received Messages Trend	None	Trend	
Receive Rate	`sum_above(V, 0)`	Inherent	

Table 4.2: Parameters based on sent and received algorithm messages.

4.5.3 Parameters Based on Time Since Last Heard

The parameters in this section are based on the *Time Since Last Heard* observations within W_{timed}, and they are listed in Table 4.3. For these parameters, the partitions are defined to have a size of one sample. This size decision is motivated by our belief that the most recent sample, *i.e* how long ago we heard from node p_i, is the most relevant sample when making a decision on whether a node is crashed. In this case, the value of relevance is denoted `V`, where $\texttt{V}_i$, is the *Time since last heard* value of the sample in the partition for node p_i. $\texttt{V}_i'$ is a copy of `V`, except that $\texttt{V}_i$ has been removed.

Parameter Name	Aggregation	Normalization	Categories
Time since last heard (TSH)	None	None	Timed Individual
Time since last heard CDF	None	CDF	
Time last heard grouped CDF	$\texttt{avg}(\texttt{V}_i')$	CDF	Timed Grouped

Table 4.3: Parameters based on the time since last heard.

4.5.4 Parameters Based on Round-Trip Counters

The parameters in this section are based on the *round-trip counter* observations within $W_{timefree,i}$, and they are listed in Table 4.3. For the parameters starting with the name **Completed round-trips since last heard**, the partitions are defined to have a sample of one. This decision is motivated by the same reason as for the parameters in Section 4.5.3. In this case, the value of relevance `V`, is the single observed *round-trip counter* in the partition. For the remaining parameters, larger partition sizes can be used, and `V` is in this case the sequence of observed *round-trip counters* in the partition.

Parameter Name	Aggregation	Normalization	Categories
Completed round-trips since last heard CDF	`sum(V)`	CDF	Time-free Grouped
Completed round-trips in Window CDF	`sum(vector_sum(V, 0))`	CDF	
Completed round-trips since last heard (RTC)	`sum(V)`	None	

Table 4.4: Parameters based on round-trip counters.

5

Precision-Distribution Module

In order to further lower the probability of falsely suspecting a non-faulty node, the Precision-Distribution (PD) module was developed which could be used together with any classifier. Given that a suspicion is raised by a classifier, the PD module can be queried in order to get an (approximated) probability that the suspicion is correct, based on previous suspicions. The PD module is in reality just a 2D-Histogram which consists of two-dimensional buckets, namely the dimensions Round-Trip Count (RTC) and Time Since (last) Heard (TSH). The buckets are populated by making a classifier predict the classes of the train set, and storing frequencies of suspicions which turned out to be FPs and TPs, given that the suspicion was raised at the RTC and TSH that each bucket represents. Each bucket can then represent the precision ($\frac{TP}{TP+FP}$), *i.e.,* the probability that a suspicion is correct. For buckets that had had no TP or FP entries during training, we took an optimistic approach and set their precision value to 1.0. Figure 5.1 shows an excerpt of such a histogram for a PR optimized Random Forest classifier. To use the PD module, a threshold is set for what the precision needs to be in order for the raised suspicion to be accepted as a suspicion. If the precision is below the threshold, then the raised suspicion is ignored.

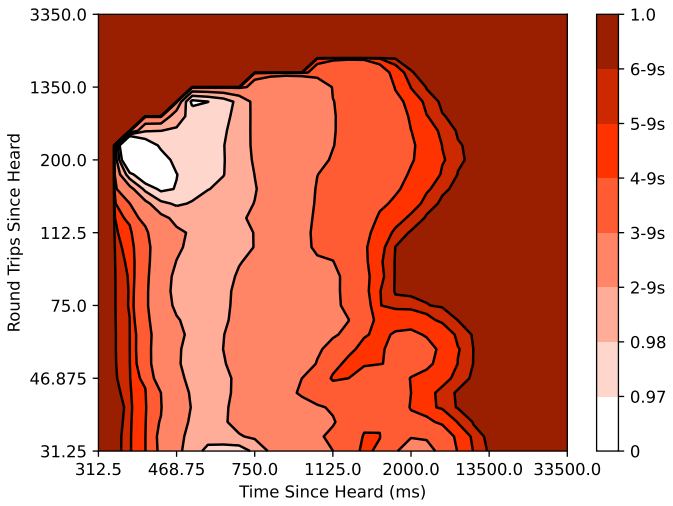

Figure 5.1: Excerpt of the PD module's underlying 2D-Histogram for the PR optimized Random Forest classifier. The frequencies indicate the precision.

5.1 Notation

The concept of using a matrix with the dimensions RTC and TSH, such as the PD module's 2D-Histogram, is also used for the TTF module, and therefore we now define notation to help when discussing such matrices. We say that $R2I(x)$ and $T2I(x)$ map a value x to the index of the corresponding RTC and TSH matrix cell, respectively. The function names stand for RTC and TSH to index, respectively. We say that $B_{x,y}$ is the value of the cell found at location (x, y) in the matrix.

5.2 Bucket Sizes

The bucket sizes were selected to be small near origo and larger further out. In particular, the bucket sizes follow the following progression. First, we set a value x, which is used as a basis when calculating the bucket sizes. This value was set to 500ms for the TSH dimension (since the FD sample interval δ is 500ms) and 50 for the RTC dimension. The first 0 to x on an axis is then split up into 16 buckets of equal size. The next x to $2x$ is split up into 8 buckets, then 4 buckets, and so on. Figure 5.2 shows the sizes of the first 31 buckets. When the bucket size reaches x, the bucket size starts to double for each new bucket, and the maximum bucket size is capped at $8x$.

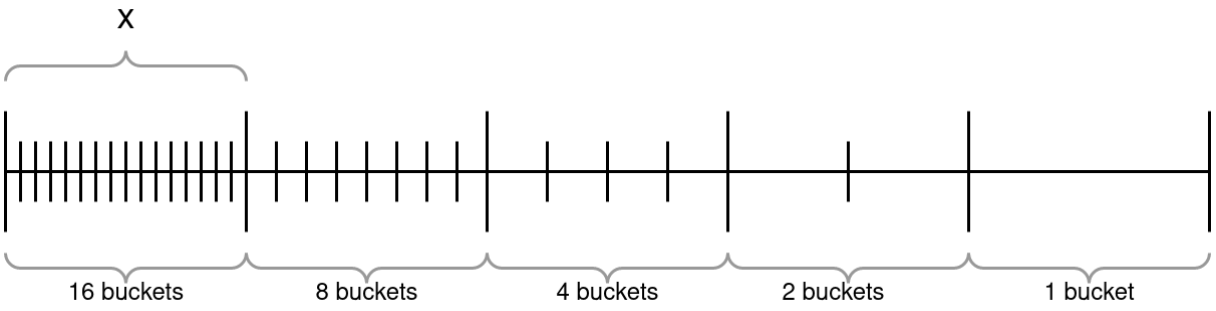

Figure 5.2: The sizes in one dimension of the first 31 buckets of the histogram.

We will now explain the motivation behind this selection of bucket sizes. Theoretically, the smaller the bucket sizes are, the more precisely the distribution that the histogram represents can be approximated. In practice, however, we only have access to a limited amount of data points, which means that bucket sizes should not be made too small, or else they might not capture any data points at all. With this in mind, we picked smaller bucket sizes close to origo (`T2I(TSH)` = 0, `R2I(RTC)` = 0) since a vast majority of data points will have low TSH and RTC values, as communication between two nodes is generally quick. Secondly, we picked larger bucket sizes for values further from origo, which is partly due to the lower frequency of data points present there, but also due to the fact that data points associated with crashed nodes data points tend to spread out as RTC and TSH increase, which can be seen in Figure 5.3. Note that the histogram in Figure 5.3 has uniform bucket sizes and that this histogram is not related to the 2D-Histogram used for the PD

module. It should be noted that this bucket size progression has other use cases than just the PD module, for instance, when picking RTC and TSH thresholds for the BDBD solution and the TTF model.

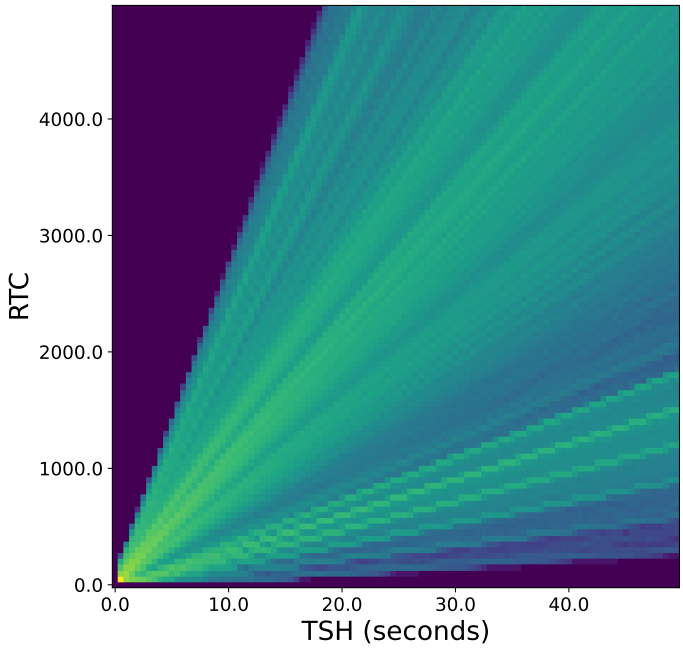

Figure 5.3: Histogram of the RTCs and TSHs observed together with data points associated with crashed nodes.

5.3 Data Smoothing

The data contained in the populated 2D-histogram was smoothed in order to avoid being over-fitted on the training data, thus increasing its generality. This was done by using a Gaussian Filter provided by SciPy, with the settings shown in Table 5.1. The only parameter that was experimented with for the settings was σ, and it was set to 1 during the final evaluation.

Setting	Value
σ	1
order	0
mode	reflect
cval	0.0
truncate	4.0

Table 5.1: Settings used for scipy.ndimage.gaussian_filter

6

Implementation

In this chapter, we present some implementation details for our data set construction in Section 6.1, the training and testing of the ML classifiers in Section 6.2, and finally the ML classifiers in Section 6.3.

6.1 Data Set Construction

In this section, we describe how the data sets were created from the collected logs. Given a deployment, we create data points for each client by using the logs of the broadcaster. Table 6.1 lists the number of data points in each set, as well as the frequency of labels. Throughout this section, we will refer to the clients as nodes and any of our FDs as *the FD*.

Data set	Number of data points	Positives (crashed)	Negatives (correct)
Train	65 928 164	15 364 165	50 564 000
Test	15 847 091	3 728 722	12 118 369

Table 6.1: The number of data points in each data set, along with the frequency of labels.

6.1.1 Collected Logs

For each of our eight deployments, the corresponding log file is split into two (chronological) partitions *s.t.* the first partition contains 80% of the logs, and the second partition 20%. We refer to these partitions as the *train* and *test* partitions, respectively.

6.1.2 Creation of Data Points

The simulator, introduced in Section 3.3, iterates through a log file in chronological order and populates the timed and time-free windows described in Section 4. The timed window W_{timed} was previously described to contain samples of system state data for all nodes. In practice, we instead implement one timed window for each node, in which the samples only contain system state data necessary for creating that node's timed parameters. We note that we implement the windows as sliding, and they can intuitively be considered queues of samples. This means that for the

timed window, every time the system is sampled, the sample is added to the window, but if the window is already fully populated we remove the oldest sample from it. Similarly, the time-free window is updated with a round-trip counter every time its corresponding node responds, and not when the system state is sampled.

The sizes used for windows, nested windows, and partitions are those described in Section 4.5. We use the timestamps of the log entries for measuring time, and the FD is queried every $\delta = 0.5$ seconds, which means that a data point is created based on the contents of the windows every δ seconds. However, if there is no log entry exactly δ seconds after the previous query, the next query will instead be performed once the next log entry is reached.

To label the data points, the synthetic crash intervals, described in Section 3.4 serve as the ground truth.

6.1.3 Bootstrapping

As mentioned in Section 4.2, the FD requires the windows to be fully populated before it can report suspicions. We say that when the windows are fully populated, the FD is bootstrapped. Thus, data points are only created when the FD is bootstrapped.

Recall that our fault model for evaluation assumes that once a node crashes, it is crashed forever. However, as we want to evaluate and train the FD on a large number of crashes, we find it infeasible to restart the system after each synthetic crash interval. Therefore, we say that each crash (and crash-free) interval for a node is preceded by an arbitrary bootstrapping period wherein the node does not experience faults. We refer to this as a *clean* bootstrap. In contrast, a bootstrap is *dirty* if it contains information about a crash interval that has been exited, since our fault model does not allow for node recovery.

We require a clean bootstrap when exiting an interval and subsequently entering a new interval, *i.e.,* moving from a crash to a crash-free interval or vice versa. It is possible to ensure that the bootstrap is clean in different ways. One can allow for a grace period after each crash interval, which has to be sufficiently long *s.t.* all data relating to the crash is emptied from the windows. However, we opted for another approach, where we simply reset the bootstrap to the most recently observed clean bootstrap after each crash interval is exited. This is achieved by saving a snapshot of the windows and the system state just before the node enters a crash interval. This essentially means that only events from crash-free intervals are considered for the bootstrap when entering a new interval.

One complication that arises with the snapshot approach when moving from a crash interval to a crash-free interval is that the system state is separated in time. If, for instance, the FD was to sample the system state, then the first half of the sample could have been from before the crash, and the second half after the crash. This results in problems since RTTs can not be correctly calculated, as the messages sent before the crash interval are not the same as the messages received after the crash. To circumvent this, we reset the system state and instruct the FD to wait δ seconds

before generating a new data point (and sampling the system state). Thus, the RTTs can start being accurately calculated again.

We also encountered a possible problem when moving from a crash-free interval to a crash interval, which was that a taken sample could contain data both from when the node was crashed and from when it was correct. To understand why this is could be seen as a problem, consider a node that crashes 1ms before the FD is queried and the data point is generated. The most recent sample was almost completely collected at a time when the node was not crashed, yet the label says that the data point belongs to a crashed node. We saw this as a problem since a data point with such a sample as its most recent sample would be indistinguishable from a data point associated with a correct node. Therefore, we cleared the system state in the same way as when moving from a crash interval to a crash-free interval. Unfortunately, due to an error in our code, the data point was still generated δ seconds after the last data point, and not δ seconds after the reset was performed. This means that the first sample taken for each crash has no observed RTTs or receives, and a time since heard $\leq \delta$. These properties can only be achieved if the sample is manipulated in this way, as a time since heard $\leq \delta$ would imply that the node responded during the time that the sample was collected, which would result in at least one entry in the list of observed RTTs in the sample. As this type of sample will only be generated once a crash is about to occur, this may be referred to as a data leak [37], which essentially means that the model is supplied with information that it will not have in production.

Even though our parameter model fully relies on normalized values, it is still highly possible that the ML models explored in this project managed to take advantage of this property to make more accurate classifications. As this was discovered too late, we were not able to correct the mistake. We believe it would have been better to simply let the system state remain untouched, rather than resetting it.

6.2 Training and Testing

The data points used for training the models were selected by a method known as random undersampling. This means that if there is a class that has a majority, a random subset of that class is selected. Specifically, we did this *s.t.* the subset of the majority class was of the same size as the set of the minority class. The only exception to this was the LSTM model, as it requires a chronological ordering of the data points to work. Therefore, the LSTM used all available training data. For testing, the entire test set was used for all classifiers.

6.3 Model Construction

This section lists the implementations used for the classifiers utilized in this work. We list relevant hyperparameters, with a particular focus on those differing from the implementations' default values. The classifiers based on NNs were created by using PyTorch [38] version `1.11.0`, and the rest were taken from Scikit-Learn [39]

version `1.0.2`. Lastly, in Section 6.3.3, we introduce a new FD, called the Ensemble FD which uses the output of all other FDs when making a prediction.

As a reminder, recall that our research question is to investigate whether ML can be used to balance the trade-off between detection time, the probability of making a false suspicion, and the time it takes to perform a state transfer better than existing solutions. Therefore, we aim at investigating a large number of ML methods off-the-shelf, instead of optimizing a select few. As such, optimizing the hyperparameters has not been the top priority, and thus only limited time has been devoted to this.

6.3.1 Scikit-Learn

Recall from Section 4.4 that our parameter model normalizes the CDF and Trend parameters such that they take values between [-1, 1], and that the mean of these parameters should be close to zero. However, since we did not present a proof that the actual mean is zero, and since the *receive rate* parameters are between [0, 1], we opted to use Scikit-Learn's `StandardScaler`, which ensures that the parameters have a mean of zero and unit variance. The `StandardScaler` was only used for the Scikit-Learn classifiers. We list the classifier implementations from Scikit-Learn in Table 6.2.

Model Name	**Implementation**	**Hyperparameters**
AdaBoost	AdaBoostClassifier	`n_estimators=100` (default: 50)
Decision Tree	DecisionTreeClassifier	`max_depth=4` (default: None)
Logistic Regression	LogisticRegressionCV	`cv=k, max_iter=1e4, tol=1e-1, solver='sag'`
Naive Bayes	GuassianNB	No changes
Random Forest	RandomForestClassifier	No changes
SVM	LinearSVC	`loss='hinge', tol=1e-5, max_iter=1e9`

Table 6.2: The implementations used from Scikit-Learn, along with hyperparameters differing from default values. The same random state was set for all implementations.

For each of the Scikit-Learn classifiers, we created two additional classifiers, namely an AUC PR- and an AUC ROC-optimized classifier. This was performed by further splitting the train set 80% - 20% into a train and validation set. First, each classifier was trained on the train subset and evaluated on the validation set. The result from the validation set was then used to construct both the PR and ROC curves. From the PR curve, the decision threshold was selected *s.t.* it minimized the difference between precision and recall. From the ROC curve, we instead selected the decision threshold that maximized the difference between sensitivity and fall-out. Finally, the thresholds were used on the corresponding classifiers trained on the entirety of the train set.

6.3.2 PyTorch Models

The hyperparameters and network architectures for the PyTorch models are shown in Table 6.3.

We note that data points were sampled differently when training the LSTM in comparison to the other classifiers. In order to ensure a chronological ordering of data points, we construct sub-batches of 100 consecutive data points each and make up each batch of these sub-batches. Thus, each batch contains 6400 data points in total. Further, as a side-effect of this, we note that we did not perform random undersampling when training the LSTM, as we were unsure of what the proper approach would be.

We also note that we typically refer to the vanilla NN simply as the 'NN model'.

Model Name	**Architecture**	**Hyperparameters**
LSTM	Input Layer (size=36) ReLU activation LSTM layer (size=8) Output Layer (size=2) Softmax activation	epochs: 100 batch size: 64 learning rate: 5×10^{-3}
NN	Input Layer (size=36) ReLU activation Linear Layer (size=8) ReLU activation Output Layer (size=1) Sigmoid activation	epochs: 5 batch size: 2048 learning rate: 5×10^{-3}
FedDyn	Same as NN model	Worker Parameters Same as NN model Server Parameters Communication rounds: 100 Device Participation: 0.1

Table 6.3: The architectures and hyperparameters for the PyTorch models. We note that FedDyn uses the same model as the NN.

6.3.3 Ensemble FD

We implement a majority-voting-based Ensemble FD by simply letting each ML-based FD cast a vote by their output. The Ensemble's output is then decided by the majority vote. We note that the Ensemble will thus be influenced by the PD module for each of the ML-based FDs. As such, we decided not to provide the Ensemble with its own PD module where it was applicable for the other FDs.

7

Evaluation

In this chapter, we describe the experiments that were set up in order to evaluate the FDs based on the QoS and classification metrics. We also describe the metrics' relevance to this project and what we expect to observe when executing the experiments.

Specifically, Section 7.1 refines our research question and motivates our choice of evaluation metrics. Section 7.2 then explains our experiment suites for QoS metric evaluation. Finally, Section 7.3 explains the execution of the experiment suites in terms of thresholds and FD configurations, as well as our expectations.

7.1 Evaluation Criteria

As mentioned in Section 1.1, our research question is whether ML-based FDs can balance the trade-off between the detection time, the probability of raising a false suspicion and the time it takes to perform a state transfer better than existing solutions. For the sake of simplicity, we refer to the probability of raising a false suspicion as $P(\texttt{FS})$. A better balance of the trade-off means either achieving shorter detection times than existing solutions whilst not increasing $P(\texttt{FS})$, or decreasing $P(\texttt{FS})$ without incurring any cost in terms of detection time. As the state transfer time is application-specific, we do not have a concrete expression to minimize, and therefore there might not be a single answer to the question of what FD is the "best". With this in mind, we aim to answer our research question by evaluating and comparing our proposed FDs to the BDBD solution at varying levels of $P(\texttt{FS})$.

We use the QoS metrics, described in Section 2.1.3, to evaluate the performance of our FDs. To answer our research question, we use the speed metric *Detection Time* (T_D), and the accuracy metric *Average Mistake Rate* (λ_M) as our main metrics. We motivate the selection of T_D over T_{D1} as it represents convergence in the FDs' suspicions, and we say that T_D corresponds to the detection time in the trade-off for our research question. We motivate the selection of λ_M by assuming that simply making a mistake leads to a state transfer, and thus the duration of the mistake itself is irrelevant. Since each FD is queried at most once every 500ms in our system, and we say that the unit of time of λ_M is 500ms, we say that λ_M approximates $P(\texttt{FS})$ in our research question. In addition to our main metrics, we also briefly compare T_D and T_{D1}, as the ratio between them provides an indication of the stability of each FD in terms of an FD's tendency to stick with a suspicion once it is raised. We

also compare P_A against λ_M to gain insights into how the FDs perform when the durations of mistakes are taken into account.

Whilst not the primary focus, we also evaluate the classifiers, which serve as the basis for our ML-based FDs, based on traditional ML evaluation techniques, such as the metrics derived from a confusion matrix. The ones considered in this project are *Accuracy*, *Sensitivity*, *Specificity*, *Precision*, *Fall-Out*, and *AUC ROC*.

7.2 Our Experiment Suites

To evaluate our FDs based on the QoS metrics we considered two different experiment 'suites', similarly to Chen *et al.* [22].

For both of our experiment suites we considered systems of n nodes p_i where $i \in \{1, ...n\}$. In all systems, p_1 was assumed to never experience failures, but we considered it possible that up to $f < n - 1$ of the other nodes would. The FDs were deployed on p_1, and their performance was evaluated on a per-node basis for p_e, where $e \in \{2, ..., n\}$. As such it was possible that p_j could experience omission failures whilst the performance of detecting crash failures of p_e was being evaluated, where $j \neq e$ and $j \in \{2, ..., n\}$.

We called the first suite of experiments the 'speed' suite, for which we evaluated the *speed* metrics. The second suite of experiments we instead dubbed the 'accuracy' suite, for which we evaluated the *accuracy* metrics. The speed and accuracy suites are referred to as E_S and E_A, respectively.

Both suites were performed by using our simulator, mentioned in Section 3.3, on the test partition of the logs.

7.2.1 The 'Speed' Experiment Suite (E_S)

The speed metrics were evaluated on instances where p_e was crashed, according to the generated crash intervals described in Section 3.4. If we say that in one instance, p_e is crashed between the times τ_{start} and τ_{end}, then T_{D1} is measured as the time from τ_{start} until the first S-transition occurs in the crash interval. We say that p_e was suspected forever by an FD if the FD suspected it until τ_{end}. As such, T_D is measured as the time from τ_{start} until an S-transition occurred which was maintained until τ_{end}.

7.2.2 The 'Accuracy' Experiment Suite (E_A)

Unlike E_S, which consisted of multiple instances where p_e was crashed, the accuracy experiment suite consisted of one continuous instance in which p_e was not crashed. This instance was achieved by omitting the synthetic crash intervals for p_e, which means that p_e was never synthetically crashed during the complete duration of the test partition of the logs. Only a single, crash-free instance is required for evaluating the accuracy metrics, as all mistakes made by an FD can be contained in it.

As all suspicions are by definition mistakes in this suite, the accuracy metrics could be derived from the suspicions made for p_e. Here, λ_M is the number of S-transitions per time unit (500ms), and P_A is the fraction of time spent in the trust state compared to the total runtime of the experiment.

7.3 Experiment Plan

We will now describe how the E_S and E_A evaluation suites were executed, in terms of what FDs were evaluated and what the relevant thresholds were set to. These thresholds refer to the value of Θ for the BDBD solution, the threshold for the TTF model, and the threshold for the PD modules for the ML models. By evaluating the FDs for different thresholds, the trade-off between the QoS metrics can be analyzed for each FD. This is based on the presumption that increasing the thresholds for any model will lower λ_M, and increase T_D.

For each set of thresholds used for evaluation, we performed both E_S and E_A evaluations. This enables a correlation between a certain detection time and its corresponding accuracy metrics.

7.3.1 Threshold Selection

Since running E_S and E_A turned out to be a CPU-heavy task, we could only perform evaluations for a limited number of thresholds. This subsection describes the methods used for picking the thresholds.

7.3.1.1 ML-based FDs

For the ML-based FDs, an analysis was performed to measure what threshold would be needed to reach a certain level of specificity in the train set. For instance, a PD module with a threshold of 0.99 for the Random Forest classifier was needed to reach a specificity of 5-9s, while a threshold of 0.99998 was needed to reach a specificity of 6-9s. Note that the resulting threshold needed for each specificity is not necessarily the same for each classifier, as the threshold is based on how each classifier performed on the train set. We based these calculations on specificity and not P_A or λ_M, as the latter metrics can only be acquired by running the E_A suite over the train partition of the logs, which is a very time-consuming task. Specificity, on the other hand, is quick to calculate for a given threshold, as it can efficiently be derived from confusion matrix data relating to how the classifier performed without using the PD module. Specificity also has the property that it is closely related to P_A and λ_M as it refers to the probability of trusting a correct node. The levels of specificity which were used for deriving the thresholds are shown in Table 7.1.

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 7.1: Levels of specificity used to derive thresholds.

7.3.1.2 The BDBD solution

For the BDBD solution, a similar analysis to that of the ML-based FDs was performed by calculating what Θ was needed to achieve a certain specificity for the train-set, given that the BDBD solution suspects everything where $\texttt{RTC} > \Theta$.

To simplify the analysis, we do not consider every possible Θ, but only the Θs corresponding to the ends of the bucket intervals explained in Section 5.2, along the RTC axis.

7.3.1.3 The TTF Model

For the TTF model, we used the same bucket sizes as in the PD module, for both the TSH and RTC dimensions. Recall that the functions R2I and T2I, introduced in Section 5.1, are used to map RTC and TSH values to buckets in a matrix. Given that each data point has an associated RTC and TSH value, we went through each matrix location (x, y) and calculated the specificity that the TTF model would achieve given that it trusted everything where $(R2I(RTC) \leq x) \vee (T2I(TSH) \leq y)$, and suspected everything where $(R2I(RTC) > x) \wedge (T2I(TSH) > y)$. The specificity for bucket (x, y) was then stored as $B_{x,y}$. The classification function $f(\texttt{RTC}, \texttt{TSH}) \rightarrow \{trust, suspect\}$ was set to suspect if the specificity $B_{R2I(RTC),T2I(TSH)}$ was above a certain threshold. The threshold was finally set to the target specificity level.

7.3.2 FD Configurations

We performed E_S and E_A evaluations for the following FD configurations C:

$\mathbf{C_{analytical}}$: The BDBD solution FD, and the TTF model FD (with different thresholds).

$\mathbf{C_{ML}}$: ML-based FDs.

$\mathbf{C_{PD}}$: ML-based FDs with the use of the PD modules (with different thresholds).

$\mathbf{C_{fallback}}$: ML-based FDs with the PD modules (with different thresholds), and the BDBD solution as a fallback.

$C_{analytical}$ is used to gather a reference point for how simple analytical solutions perform at different thresholds. This configuration also provides a benchmark of sorts by the BDBD solution, to which we will compare the ML-based FDs. Increasing the threshold is expected to improve the accuracy metrics while worsening the speed metrics. We expect that the TTF model balances the trade-off better than the BDBD solution, as it uses an additional temporal dimension.

C_{ML} is used to get an understanding of how well the classifiers work as FDs when used 'off-the-shelf', *i.e.,* without the use of any interfering additional module. This configuration provides a baseline for the ML-based FDs. We hope that the FDs in this configuration achieve shorter T_D than the BDBD solution, at similar levels of λ_M. We also expect FedDyn to be worse than the NN model, as FedDyn is trained in an FL setting. We also expect that the LSTM model might be better than the remaining models as it is the only model that uses past predictions when making a new prediction.

C_{PD} is used to analyze the trade-offs between the QoS metrics as we attempt to lower $P(\texttt{FS})$ by using the PD modules. As the PD module thresholds are increased, we expect that the FDs in C_{PD} will experience a decline in terms of speed metrics and an improvement in terms of accuracy metrics. Explicitly, we expect that T_D and T_{D1} will increase alongside P_A, whilst λ_M will decrease.

$C_{fallback}$ is used to evaluate the BDBD solution as a fallback and corresponds to our proposed solution for the studied problem. In this configuration, the BDBD solution runs simultaneously as the ML-based FD which is being evaluated, and any suspicions made by the BDBD solution will overrule the ML-based FD's output. As such, the fallback can be seen as an upper threshold after which the ML-based FDs are forced to make suspicions. Using a fallback does not inherently improve the performance of an ML-based FD, as the underlying ML-based FD's decisions are overruled, but it guarantees that the ML-based FD converges to suspecting a node forever.

Selecting a reasonable threshold for the fallback is not trivial if the goal is to improve the ML-based FDs, as a low threshold might overrule the ML-based FDs completely, and a high threshold might not affect them at all. Therefore, we set the fallback's threshold (Θ) in three different ways in the configurations $C_{fallback,higher}$, $C_{fallback,equal}$ and $C_{fallback,lower}$. The thresholds used for the different configurations are selected relative to the threshold used in the PD modules for the ML-based FDs. In the following configuration descriptions, we assume that a PD module threshold has been set based on it yielding a specificity of X-9s in the train set. For $C_{fallback,higher}$, $C_{fallback,equal}$ and $C_{fallback,lower}$, we set Θ based on a specificity level of $(X+1)$-9s, X-9s, and $(X-1)$-9s respectively.

The expectation that we have for $C_{fallback}$ is that lower thresholds for the fallback will give the BDBD solution more influence on the output, as the BDBD solution overrides the ML-based FD once Θ has been met. The fallback's impact on an ML-based FD depends on how the ML-based FD performs in relation to the BDBD solution. If, say, the BDBD solution has a T_D of 5 seconds at a λ_M of 0.01, while an ML-based FD has a T_D of 1 second at the same λ_M, then we expect that $C_{fallback,higher}$ and

$C_{fallback,equal}$ will not impact the ML-based FD at all since the ML-based FD will already have made its suspicions once the fallback is hit. $C_{fallback,lower}$ might have an impact, however, as the BDBD solution's T_D is likely to be lower than the ML-based T_D when a lower threshold is used. On the other hand, if the BDBD solution has a shorter T_D than the ML-based FD at the same value of λ_M, then all configurations of $C_{fallback}$ might have some form of impact on the ML-based FD's performance, with $C_{fallback,lower}$ being the most significant one.

8

Results

We begin this chapter by showing the distributions of our collected data in Section 8.1. It is shown that the distributions have shifted slightly between the train and test set, and that observing TSHs above one second is quite rare, especially in the test set.

Following this, we show results from comparing the investigated FDs to the BDBD solution. Recall from Section 7.1 that our refined research question is whether ML-based FDs can balance the trade-off between the Detection Time (T_D) and the Average Mistake Rate (λ_M) better than existing solutions. The first results to be shown are those of the TTF model, and it is shown that its T_D is consistently at least 19.5% lower than that of the BDBD solution at various λ_M.

Following this, we investigate how well the classifiers, which serve as the basis for our ML-based FDs, perform 'out-of-the-box' in terms of classification metrics. Since a large number of classifiers were investigated in this work, we choose to show only a selection of them, along with their corresponding ML-based FDs, in this chapter. In doing this, we focus on those that are conceptually different and our best performers. Specifically, we show the ML-based FDs based on the PR curve optimized Random Forest and AdaBoost models, NN, LSTM, FedDyn, as well as our Ensemble FD. It is shown that the classifiers solve the binary classification problem with near-perfect classification metric scores. In particular, the PR optimized Random Forest classifier achieves an accuracy of 0.999973 and a specificity of 0.999980. Further results for all FDs and their corresponding classifiers are shown in Appendix A.

Then, in Section 8.4, we compare the QoS metrics of ML-based FDs to the BDBD solution, when the PD module was used to impact λ_M for the ML-based FDs. It is shown that the PD module can be used to successfully lower λ_M, whilst sacrificing T_D. Our best FD is shown to consistently balance the trade-off better than the BDBD solution while being able to achieve a T_D that is up to 86.3% lower.

Our last results, presented in Section 8.5, show that our proposed solution, $C_{fallback}$, affects the performance of the ML-based FDs and, whilst not the main goal, is also able to further lower T_D compared to C_{PD}.

We end this chapter by interpreting our results and discussing their limitations in Sections 8.7 and 8.8, respectively.

To display the results of the trade-off between T_D and λ_M, we make use of trade-off

plots and tables. We will now explain how to interpret these, as they might not be completely intuitive. The following explanation regards Figures 8.1, 8.2, 8.3, 8.4, 8.5, and 8.6 as well as Tables 8.3, and 8.7. In the plots, each marker signifies a measured result from an evaluation, while the lines between the markers are the result of using linear interpolation for the relation between the logit-scaled λ_M, and T_D. The values in the aforementioned tables represent an FD's T_D relative to that of the BDBD solution. We use the notation $\lambda_M \leq$ `level` to indicate that the relation is based on measured results, explicitly those that are closest to `level`. This is done as we wanted to avoid extrapolation. The notation $\lambda_M =$ `level` instead indicates that interpolated values were used to calculate the relation. We used interpolation where measured results were sparse. Note that for all T_D and λ_M trade-off plots, a Y-axis value of 0 signifies that the FD made no mistakes.

Further, we note that we do not include plots for the Query Accuracy Probability (P_A), as we found its inverse almost exactly followed the same relation to T_D as that of λ_M.

8.1 Distributions of Collected Data

To provide some context for the results, we begin by presenting the distributions of RTC and TSH for the data points in Tables 8.1 and 8.2. #TSH and #RTC is the frequency of occurrences above or equal to the value corresponding to each specified percentile. We note that these distributions represent data points gathered in a crash-free context, *i.e.,* one without simulated crashes. It can be seen that the crash-free train set distribution is shifted towards larger RTCs and TSHs compared to the crash-free test set distribution.

Percentile	**RTC**	**#RTC**	**TSH**	**#TSH**
99	33	708253	183	660023
99.9	45	66465	384	66002
99.99	806	6602	13994	6601
99.999	12493	661	123033	661
99.9999	18641	67	195619	67
99.99999	20814	7	210134	7
100	21136	1	213135	1

Table 8.1: Distribution of the RTCs and TSHs for data points of the training set (generated without using synthetic crashes).

Percentile	RTC	#RTC	TSH	#TSH
99	32	163700	172	158681
99.9	42	19348	285	15869
99.99	57	1621	625	1587
99.999	244	159	2621	159
99.9999	1039	16	7270	16
99.99999	1722	2	9258	2
100	1807	1	9391	1

Table 8.2: Distribution of the RTCs and TSHs for data points of the test set (generated without using synthetic crashes).

8.2 Analytical FDs

We now present the QoS results of the two analytical FDs, namely the BDBD solution and the TTF model. These results were gathered from evaluating $C_{analytical}$ and they serve as an indication of what using simple statistics can accomplish. Recall that we expect to see that using both RTC and TSH, as in the TTF model, results in shorter detection times in comparison to the BDBD solution which only uses RTC. Figure 8.1 shows the trade-off between T_D and λ_M when the thresholds are varied. The figure shows that the TTF model generally balances the trade-off better than the BDBD solution, in terms of λ_M for lower T_D. In particular, for the levels shown in Table 8.3, the TTF model achieves a speed-up of at least 19.5%.

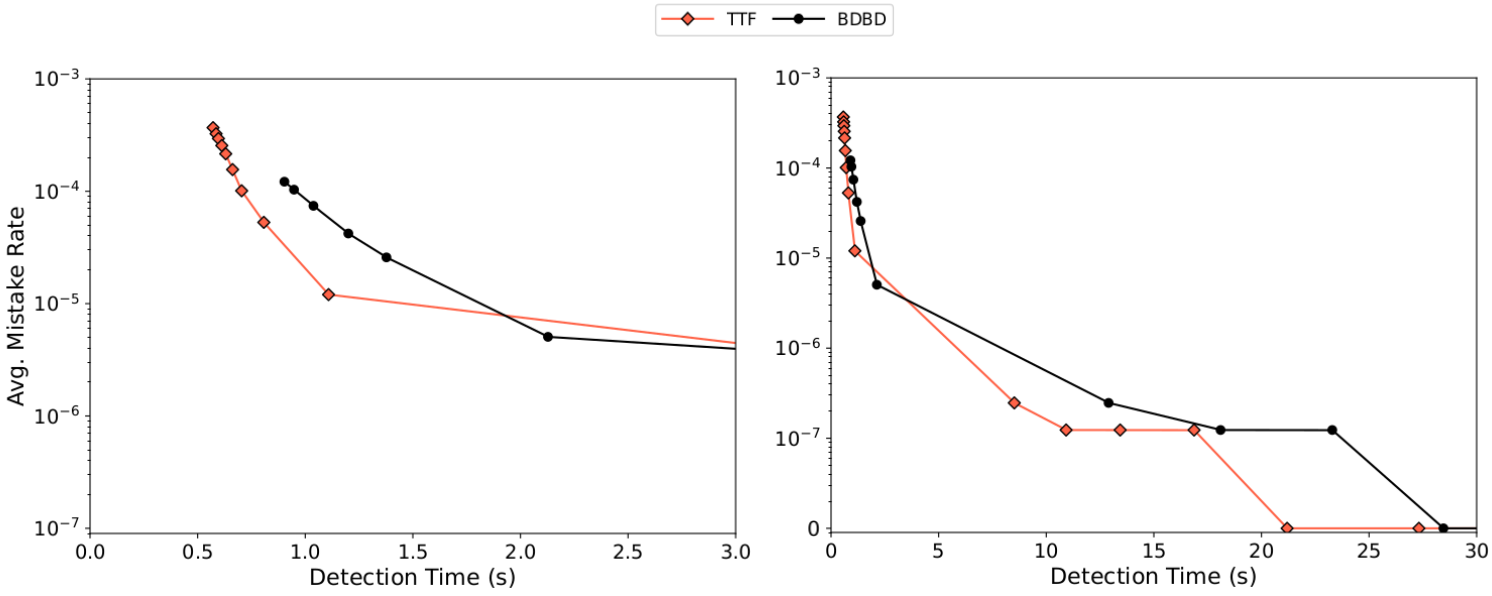

Figure 8.1: The trade-off between T_D and λ_M for the results of $C_{analytical}$.

λ_M	TTF
$\lambda_M \leq 1 \times 10^{-3}$	0.633
$\lambda_M = 1 \times 10^{-4}$	0.738
$\lambda_M = 1 \times 10^{-5}$	0.805
$\lambda_M = 1 \times 10^{-6}$	0.739

Table 8.3: The TTF FD's T_D relative to the BDBD solution's T_D for achieving certain values of λ_M, in $C_{analytical}$.

8.3 ML-based FDs

We will now show the results gathered from C_{ML}, *i.e.,* the ML-based FDs without the use of the PD module. Table 8.4 shows the confusion matrix results for the classifiers, and Table 8.5 shows the derived classification metrics. It should be noted that the AUC ROC scores for NN, LSTM, and FedDyn were not calculated based on the predicted probabilities of the classifiers, but only on the predicted classes themselves.

Classifier	**FP**	**TP**	**FN**	**TN**
Random Forest PR	240	3728537	185	12118129
AdaBoost PR	271	3726732	1990	12118098
NN	830	3728640	82	12117539
LSTM	899	3728573	26	12109602
FedDyn	10995	3737877	2632	12107374

Table 8.4: The confusion matrix data results from evaluating the classifiers on the test set.

Classifier	**Accuracy**	**Sensitivity**	**Specificity**	**Precision**	**Fall-Out**	**AUC ROC**
Random Forest PR	0.999973	0.999950	0.999980	0.999936	0.000020	0.99999817
AdaBoost PR	0.999857	0.999466	0.999978	0.999927	0.000022	0.99999996
NN	0.999942	0.999978	0.999932	0.999777	0.000068	0.99995476
LSTM	0.999942	0.999993	0.999926	0.999759	0.000074	0.99995940
FedDyn	0.999141	0.999296	0.999093	0.997067	0.000907	0.99919453

Table 8.5: The classification metrics derived from Table 8.4.

Figure 8.2 shows the trade-off between T_D and λ_M for the ML-based FDs of C_{ML}. As C_{ML} did not make use of the PD module, no thresholds were varied in C_{ML}, and thus the ML-based FDs' performance results are presented as points in the graph. For comparative purposes, we also show the results from $C_{analytical}$ in the graph.

We note that our hopes for the ML-based FDs held as they were mostly able to balance the trade-off better than the analytical FDs, however only for certain λ_M. We

can however not draw any such conclusions for FedDyn as we are missing analytical FD results at FedDyn's λ_M. However, our expectation on the relative performance of FedDyn FD held true, as its λ_M is over an order of magnitude higher than that of the NN FD, at a higher T_D. Our expectation for the LSTM FD did not hold, as it performed similarly to the other ML-based FDs and not better.

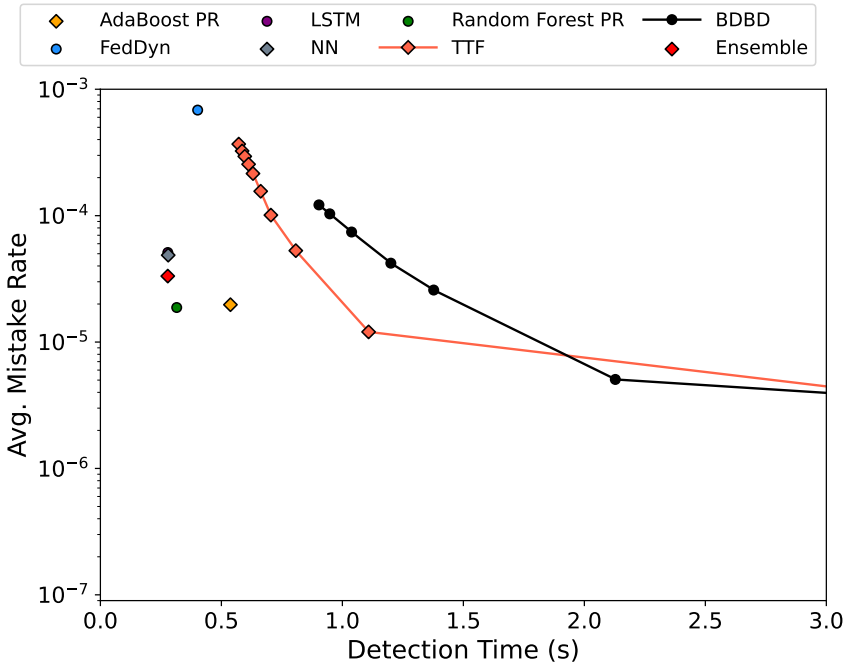

Figure 8.2: The trade-off between T_D and λ_M for the results of $C_{analytical}$, and C_{ML}. Note that the NN and LSTM markers are stacked on top of each other.

8.4 ML-based FDs with the PD Module

We are now interested in seeing how the ML-based FDs perform at multiple levels of λ_M. As expected, lower levels were achieved in the evaluation for C_{PD}, where the PD module thresholds were varied for the ML-based FDs. Since the thresholds were chosen according to what level of specificity they provided for the FDs in the train set, we now present what specificity they achieved in the test set. This is shown in Table 8.6 which also includes the specificity levels achieved for $C_{analytical}$ at certain thresholds. It can be seen that the performance on the test set was better in terms of specificity in comparison to what the thresholds achieved in the train set.

Figure 8.3 shows the trade-off between T_D and λ_M for C_{PD}, and for comparative purposes also the results from $C_{analytical}$. Table 8.7 shows that the ML-based FDs mostly perform well for C_{PD} in comparison to the BDBD solution for the given levels of λ_M. Whilst all ML-based FDs are able to achieve a speedup for several levels, only the PR curve optimized Random Forest, FedDyn, and Ensemble show T_D's consistently lower than the BDBD solution, where the PR curve optimized Random Forest is able to achieve a T_D that is up to 86.3% lower than that of the BDBD solution, at $\lambda_M = 1 \times 10^{-6}$.

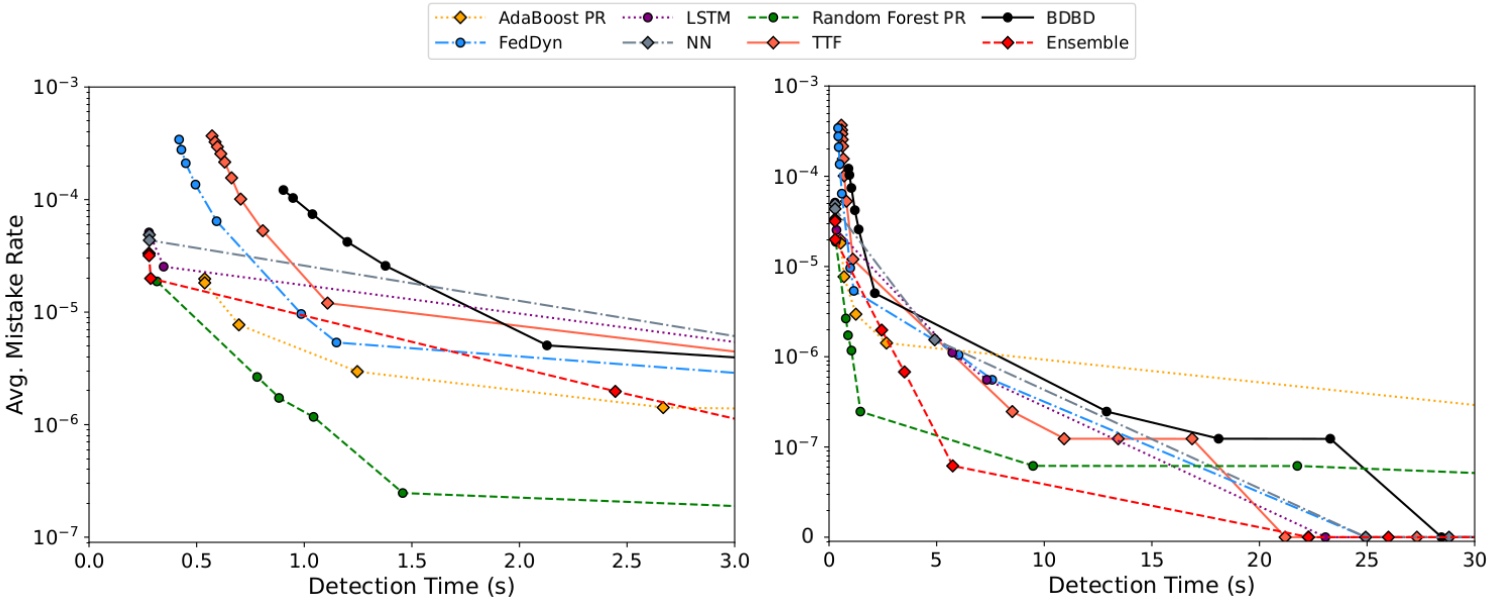

Figure 8.3: The trade-off between T_D and λ_M for the results of $C_{analytical}$, and C_{PD}.

FD (config)	Specificity	FD (config)	Specificity
AdaBoost PR (t = 0.999)	0.999 977 061	Random Forest PR (t = 0.999)	0.999 980 779
AdaBoost PR (t = 0.9999)	0.999 977 061	Random Forest PR (t = 0.9999)	0.999 980 779
AdaBoost PR (t = 0.99999)	1	Random Forest PR (t = 0.99999)	0.999 997 227
		Random Forest PR (t = 0.999999)	1
FedDyn (t = 0.999)	0.999 520 671	TTF (t = 0.999)	0.999 565 793
FedDyn (t = 0.9999)	0.999 995 147	TTF (t = 0.9999)	0.999 998 677
FedDyn (t = 0.99992)	1	TTF (t = 0.99994)	1
LSTM (t = 0.999)	0.999 918 893	BDBD (t = 0.999)	0.999 839 362
LSTM (t = 0.9999)	0.999 995 021	BDBD (t = 0.9999)	0.999 998 677
LSTM (t = 0.99992)	1	BDBD (t = 0.99994)	1
NN (t = 0.999)	0.999 925 384	Ensemble (t = 0.999)	0.999 945 992
NN (t = 0.9999)	0.999 992 186	Ensemble (t = 0.9999)	0.999 992 501
NN (t = 0.99991)	1	Ensemble (t = 0.99993)	1

Table 8.6: The specificity that was achieved on the test set when a configuration was run with a threshold that achieved a specificity of t in the train set.

8.5 ML-based FDs with the PD Module and Fallback

Finally, we present the evaluation results of $C_{fallback}$, in which the ML-Based FDs used the PD module as well as the BDBD solution as a fallback. Figures 8.4, 8.5, and 8.6 show $C_{fallback,higher}$, $C_{fallback,equal}$ and $C_{fallback,lower}$ respectively. Table 8.7 shows each FD's T_D in comparison to that of the BDBD solution. Our expectation

that the BDBD solution influences the output more as the fallback threshold is decreased seems to hold, as most FD results approach the BDBD solution's curve in the plots. We note that for $C_{fallback,lower}$, our best ML-based FD, the PR curve optimized Random Forest, is up to 86.6% faster than the BDBD solution, at $\lambda_M = 1 \times 10^{-6}$.

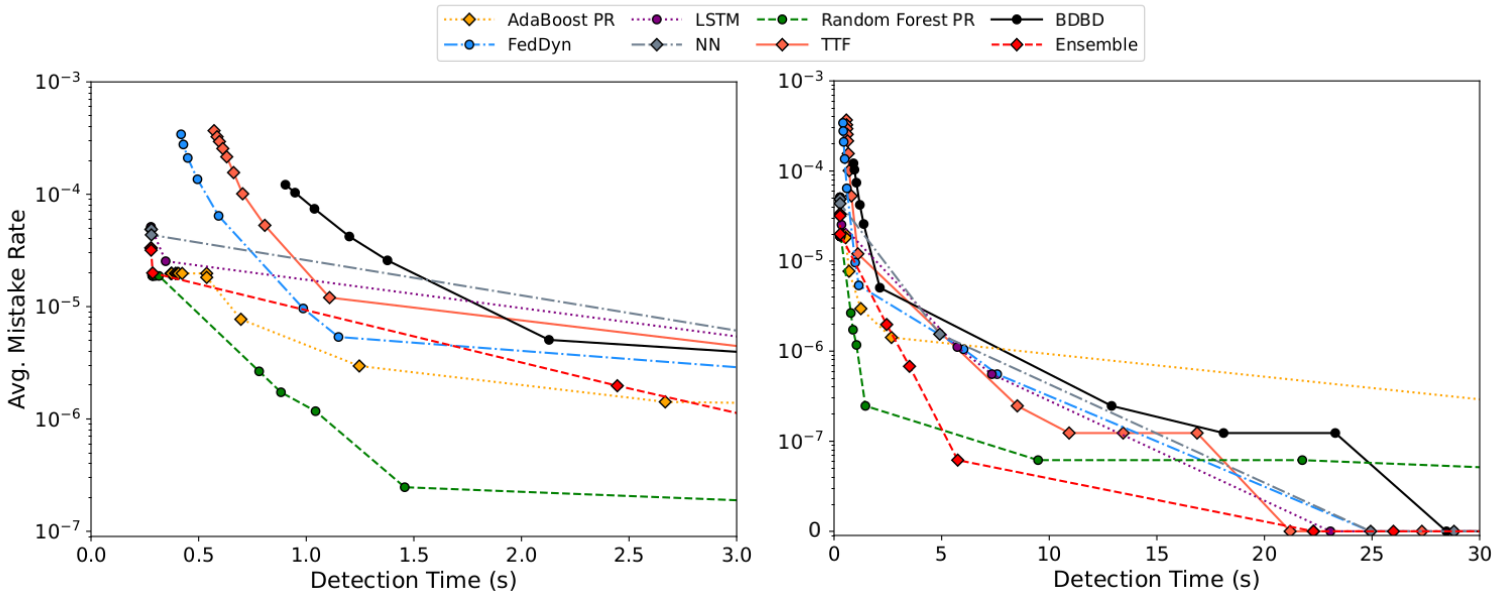

Figure 8.4: The trade-off between T_D and λ_M for the results of $C_{analytical}$, and the selection of ML-based FDs from $C_{fallback,higher}$.

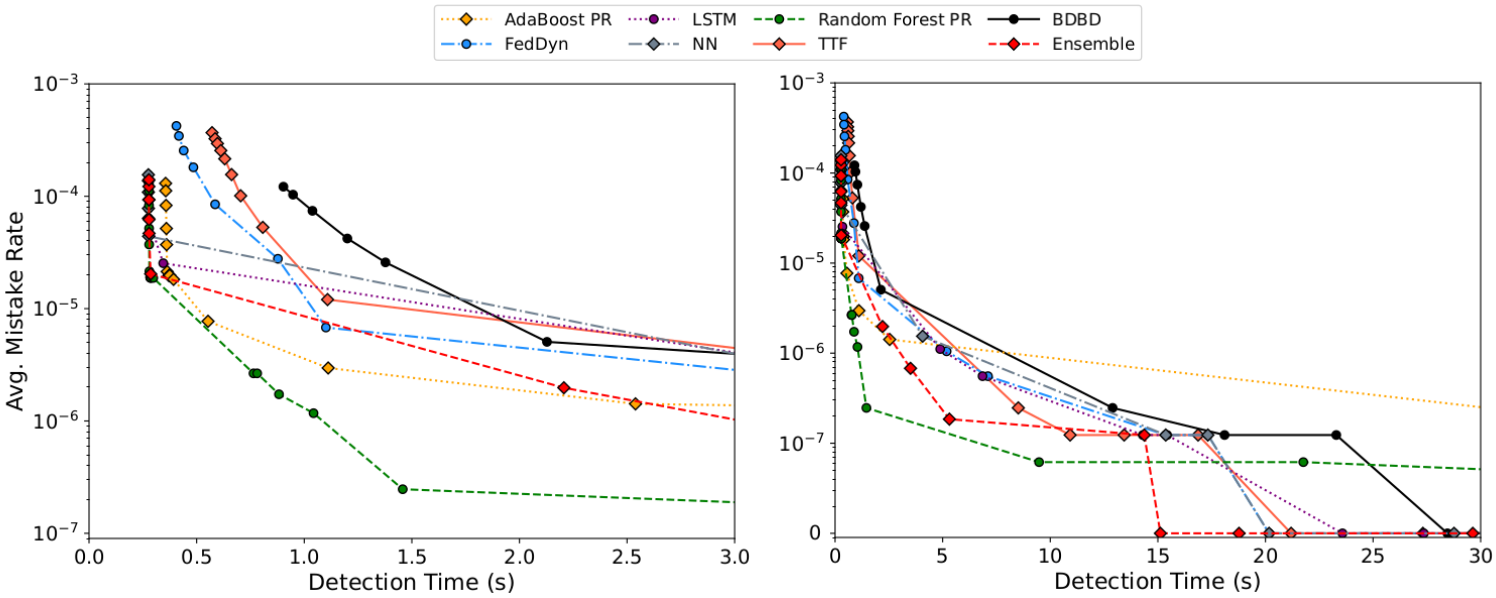

Figure 8.5: The trade-off between T_D and λ_M for the results of $C_{analytical}$, and the selection of ML-based FDs from $C_{fallback,equal}$.

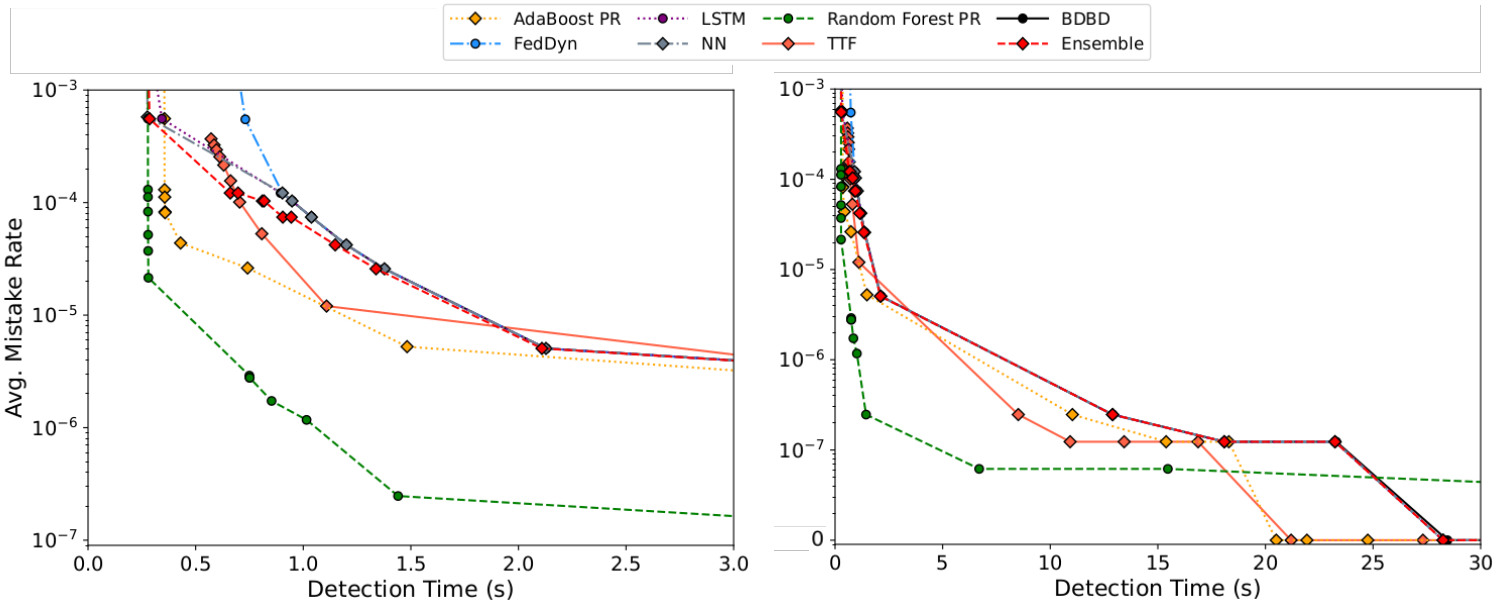

Figure 8.6: The trade-off between T_D and λ_M for the results of $C_{analytical}$, and the selection of ML-based FDs from $C_{fallback,lower}$.

λ_M	Configuration	Random Forest PR	AdaBoost PR	LSTM	NN	FedDyn	Ensemble
$\lambda_M \leq 1 \times 10^{-3}$	C_{2DHist}	0.349	0.595	0.308	0.310	0.463	0.308
	$C_{fallback,higher}$	0.314	0.414	0.308	0.308	0.463	0.308
	$C_{fallback,equal}$	0.308	0.394	0.305	0.305	0.449	0.308
	$C_{fallback,lower}$	0.308	0.393	0.380	0.305	0.809	0.316
$\lambda_M \leq 1 \times 10^{-4}$	C_{2DHist}	0.304	0.518	0.268	0.270	0.571	0.268
	$C_{fallback,higher}$	0.273	0.360	0.268	0.268	0.571	0.268
	$C_{fallback,equal}$	0.268	0.345	0.266	0.266	0.564	0.268
	$C_{fallback,lower}$	0.268	0.345	1.	1.	1.	0.872
$\lambda_M = 1 \times 10^{-5}$	C_{2DHist}	0.256	0.357	1.074	1.278	0.539	0.513
	$C_{fallback,higher}$	0.245	0.357	1.074	1.278	0.539	0.513
	$C_{fallback,equal}$	0.245	0.278	0.935	1.077	0.573	0.480
	$C_{fallback,lower}$	0.253	0.653	1.	1.	1.	0.985
$\lambda_M = 1 \times 10^{-6}$	C_{2DHist}	0.137	1.920	0.755	2.526	0.776	0.395
	$C_{fallback,higher}$	0.137	1.920	0.755	2.526	0.776	0.395
	$C_{fallback,equal}$	0.137	1.905	0.655	0.802	0.674	0.384
	$C_{fallback,lower}$	0.134	0.842	1.	1.	1.	0.999

Table 8.7: Each FD's T_D relative to the BDBD solution's T_D for achieving certain values of λ_M, in C_{PD}, $C_{fallback,higher}$, $C_{fallback,equal}$, and $C_{fallback,lower}$.

8.6 Stability of the ML-based FDs

We are also interested in how T_{D1} is correlated to T_D. An FD that is stable and sticks with its choice of suspecting a node should have an equality between T_D and

T_{D1}, while a more unstable FD will have a longer T_D than T_{D1}. Plot 8.7 shows the relation between T_D and T_{D1} for configurations C_{PD}, $C_{fallback,higher}$, $C_{fallback,equal}$, and $C_{fallback,lower}$. It can be seen that as the fallback threshold is lowered, T_D approaches T_{D1} for most FDs.

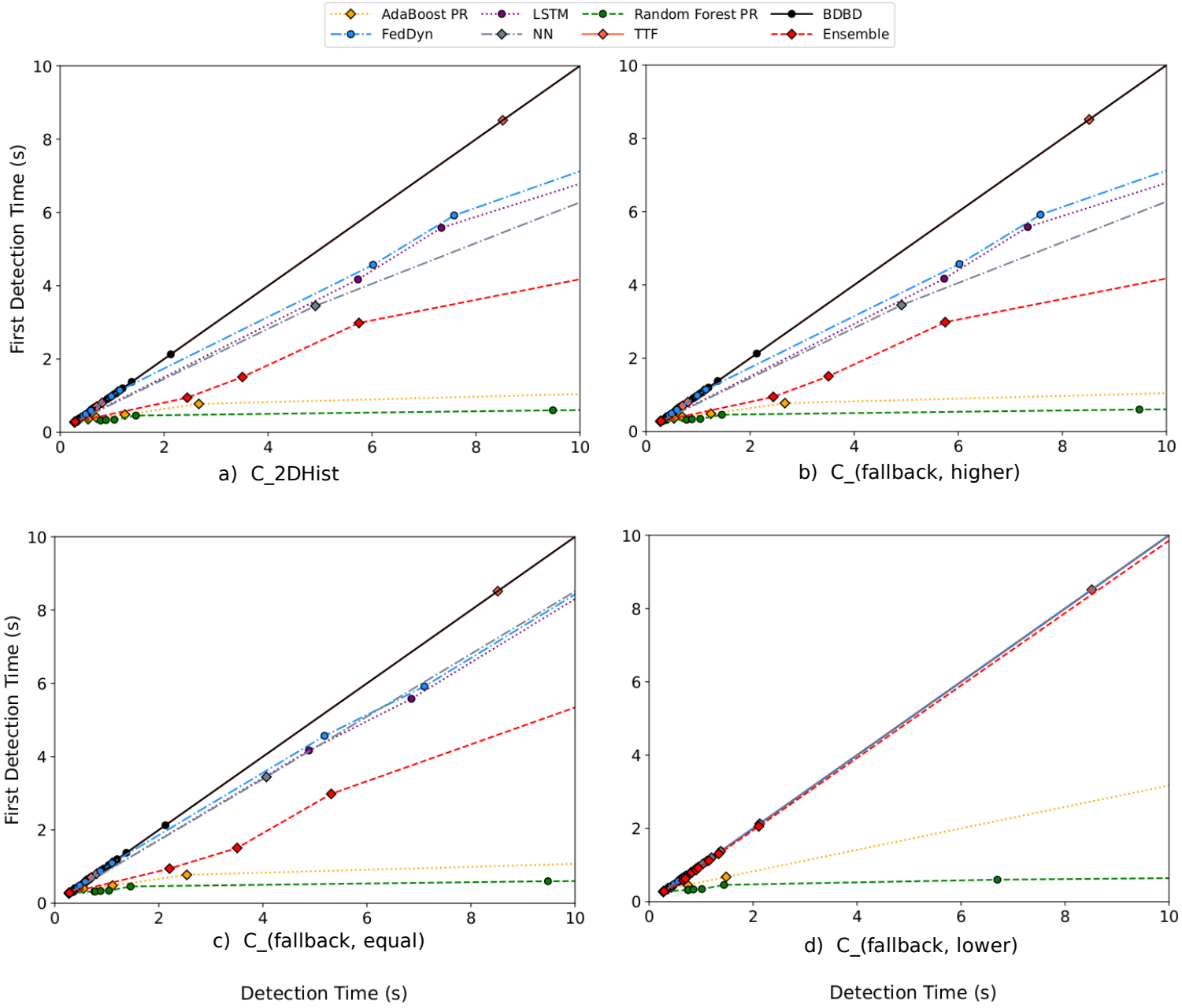

Figure 8.7: The trade-off between T_D and T_{D1} for all configurations.

8.7 Interpretation of Results

Starting from the results of $C_{analytical}$, we find an indication, via the TTF model, that extending the BDBD solution with further temporal dimensionality and simple

statistics based on observed data, can yield lower detection times.

The results of C_{ML} show that the detection times can be further reduced through the use of ML. The classifiers perform extremely well on our data set, with near-perfect classification metrics as seen in Table 8.5. This indicates that our parameter model is effective for the studied problem. The differences between the classifiers are, except for FedDyn, a few hundred FPs and FNs, which could be an indication that most data points are easy to classify, and that a few of them are very difficult.

We hypothesize that these data points correspond to extremes in terms of RTCs and TSHs for their respective classes, which makes their correct classification difficult as they seemingly belong to the other class. Concretely, we believe that the FPs are caused by negative data points corresponding to large RTCs and TSHs, and that the FNs are caused by positive data points which instead correspond to low RTCs and TSHs.

The results of C_{PD} indicate that our PD module is an effective means to lower λ_M. Following this, the results of $C_{fallback}$ show that a fallback for the ML-based FDs can improve their performance in some instances, and worsen it in others. Most ML-based FDs were mainly unaffected by $C_{fallback,higher}$, which is likely because the FDs had already made their final suspicions when the fallback was hit. $C_{fallback,equal}$ showed that the equal fallback threshold mostly resulted in an improvement in detection time in comparison to C_{PD}, with the exception of FedDyn at $\lambda_M \leq 1 \times 10^{-3}$. $C_{fallback,lower}$ instead showed a decrease in performance in most instances. This is most likely because the fallback threshold is too low in comparison to the PD module threshold, which results in the ML-based FD being overridden by the BDBD fallback. This can also be seen in Figure 8.6 as many of the ML-based FDs follow the BDBD line exactly. However, for $C_{fallback,lower}$ we observe consistent improvements for the PR curve optimized Random Forest in comparison to C_{PD}, and a few improvements for the PR curve optimized AdaBoost.

From C_{ML} and C_{PD}, it is shown that the LSTM FD has similar performance to the remaining models, which hints that its sequential dimension was not advantageous in the context of this project. This is probably due to the fact that our parameter model already has the sequential dimension built-in in the form of Trends and CDFs of windows.

In terms of comparing T_D and T_{D1}, we see in Figure (a) of 8.7 that for C_{PD}, the FDs tend to have a lower T_{D1} than T_D, which means that the FDs do not generally stick with their first decision of suspecting. This is most likely a side effect of the way that the PD module is designed. As a simple case, consider the histogram in Figure 5.1. It can be seen that if for instance a threshold of 3-9s is set for making suspicions, then suspicions will be avoided between 450ms and 1125ms, but suspicions might still be raised before and after this interval, causing a gap between T_D and T_{D1}. Such behavior could be avoided by implementing a PD module that is monotonically increasing as RTC and TSH increase. Since the TTF model's classification function does have this property, one could also try to use this as a means of preventing FPs. Further, as the fallback threshold is decreased for $C_{fallback}$, we see that T_D sticks closer to T_{D1}, which is expected as the BDBD fallback forces the ML-based FDs to

make their final suspicion earlier than they would have done on their own.

As we noted in the introduction of this chapter, we found that P_A's inverse almost exactly followed the same relation to T_D as that of λ_M. We interpret that this means that the mistake durations were generally kept at around 500ms, which is the shortest duration that a mistake can be. This essentially means that they both represent the probability that a query will result in a mistake. This can be made more intuitive by writing these metrics as $\lambda_M = \frac{\text{number of mistakes}}{\text{total experiment time / 500ms}}$ and $1 - P_A = \frac{\text{time spent in mistakes}}{\text{total experiment time}} = \frac{\text{number of mistakes} \cdot \text{avg mistake duration}}{\text{total experiment time}}$. Then, $\lambda_M = 1 - P_A$ if avg mistake duration = 500ms.

8.8 Limitations

We will now explain the limitations of our results. Firstly, since the test set only has a size of $15 \cdot 10^6$ data points, one should be careful when interpreting results below a λ_M of 10^{-5}, as they depend on a few specific data points of the test set. Since this inherently provides very coarse granularity for $\lambda_M \leq 10^{-5}$, we performed interpolation in an attempt to perform somewhat fair comparisons between the FDs at this level.

Furthermore, one must take into consideration the sparseness of the points shown in the plots throughout this chapter. Firstly, we resort to interpolation at levels 1×10^{-5} and 1×10^{-6}, which adds a level of uncertainty since the interpolation might not necessarily reflect the actual relation between the points. Secondly, it is difficult to fairly compare the results between different FDs at λ_M levels above 1×10^{-5}, as their lines start at different levels of λ_M. Therefore, the actual λ_M values considered for $\lambda_M > 1 \times 10^{-5}$ are possibly different between FDs and between configurations. Thus, one must be careful when interpreting both how the T_D of FDs compare to each other, and how the T_D is altered between configurations.

One must also take into consideration that PlanetLab Europe, on which the data was collected, was highly stable during the data collection. This was seen in Tables 8.1 and 8.2 which showed that RTTs between nodes are often kept short. Therefore, there is no guarantee that the performance would be similar to our results if the data was collected from a more unstable system in which it is common for a non-crashed node to not reply for several seconds.

Further, the FDs were evaluated on a distribution that was notably different from the one that they were trained on, as can be seen in Section 8.1. This distribution shift seems to have had an impact on the performance of the FDs, as the FDs achieved a lower specificity in the training set compared to the test set when run with the same configuration. If the train and test distributions were even more similar, the results might have looked different.

Lastly, in terms of classification metrics for the ML-based FDs, the number of FNs is very low, which is a surprising result, as intuitively it should be difficult for a classifier to correctly classify a data point as positive if the node corresponding to it was heard from very recently. We believe that this might be a consequence of

the data leakage problem described in Section 6.1.3, which may have resulted in increased classifier performance.

9

Discussion

Through this work, we aimed to answer our research question *can ML-based FDs balance the trade-off between detection time, the probability of making an incorrect suspicion, and the cost of a state transfer better than existing solutions?* We decided to answer this by comparing our results to a simple unreliable FD by Blanchard et al. [20] which we called the BDBD solution. The BDBD solution suspects a node p_i if Θ round-trips are completed by other nodes $p_j \neq p_i$ since p_i was last heard from.

Before considering ML, however, we created an FD which was a simple extension to the BDBD solution in the sense that it also uses the time since a node p_i was last heard from when making suspicions. This solution, called the Timed and Time-Free (TTF) model, was shown to consistently be at least 19.5% faster than the BDBD solution, at various mistake rates.

We then moved on to using ML, and we modeled the problem of failure detection as a binary classification problem. We did this by developing a method that converted commonly available message-passing system data into parameters, which could be used by popular ML methods. The resulting parameter model consisted of two types of parameters, namely timed parameters, which are calculated by using clocks (or timers), and time-free (round-trip-based) parameters, which are calculated by counting round-trips completed in the system. We learned that our parameter model was well suited for modeling the problem, even in an FL setting, since all ML classifiers solved the problem nearly perfectly, as shown by the resulting classification metrics. In particular, our best performing classifier, namely the PR optimized Random Forest classifier, achieved an accuracy of 0.999973 and a specificity of 0.999980. We believe that specificity is of particular relevance for the studied problem, as it (somewhat) reflects the probability that our FDs `trust` correct nodes.

However, as the detection time (T_D) of the BDBD solution depends on how many mistakes we allow it to make, we also needed a way to evaluate how our ML-based FDs perform at different mistake rates (λ_M), so that the solutions could be more accurately compared at multiple levels of λ_M. We, therefore, developed a module called the Precision-Distribution (PD) module, which tracks the precision of the ML-based FDs through the use of a 2D-Histogram based on how long ago a node p_i was last heard from, and how many round-trips had been completed by other nodes $p_j \neq p_i$ since p_i was last heard from. The PD module was shown to successfully reduce λ_M for our ML-based FDs through the use of a threshold which prohibited

the FDs from raising suspicions if the precision was too low based on histogram data.

One strength of the BDBD solution is that it is of class $\diamond P$, which means that it eventually suspects all crashed nodes, and eventually it never suspects any correct nodes. These properties, *i.e.,* FD completeness and accuracy, are not inherent to our ML-based FDs. Because of this, we implemented the BDBD solution as a fallback for the ML-based FDs to guarantee completeness. However, we do not guarantee accuracy. This approach was, in some cases, able to increase the performance of our ML-based FDs, but more research is required if one aims to do this, as we do not provide a method of strategically selecting fallback thresholds.

There are however some limitations to our results. Explicitly, we only have a relatively low number of data points (15.8×10^6 in the test set), which makes it difficult to draw general conclusions about certain aspects of our results, in particular the observed detection times at low mistake rates. Additionally, our data was collected from a highly stable system consisting of only eight nodes, and thus the data is not very diverse. Finally, we found a potential data leak that may have influenced the performance of our classifiers in the sense that they might have performed better in our evaluation suites than they would in a production setting.

We answer our research question with an affirmative through our proposed solution, which consists of our ML-based FDs, the PD module, and the BDBD solution fallback, where the latter is to guarantee FD completeness. As the input to our parameter model is very simple, we believe that most, if not all, message-passing applications should be able to implement our solution. Because of this, we hope that our solution can facilitate studies about more efficient distributed systems, especially those that can utilize consensus algorithms, such as State Machine Replication, Blockchain, and Distributed Machine Learning.

Bibliography

[1] V. Hadzilacos and S. Toueg, "A modular approach to fault-tolerant broadcasts and related problems," Cornell University, Tech. Rep., 1994.

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

[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. [Online]. Available: https://doi.org/10.1145/3149.214121 [Accessed: Jul. 1, 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. [Online]. Available: https://doi.org/10.1145/226643.226647 [Accessed: Jul. 1, 2022]

[5] K. Sozinov and K. Hammar. (2017) Machine learning for failure detection in distributed systems — a course project report. [Online]. Available: https://limmen.dev/assets/papers/mlfd.pdf [Accessed: Jul. 1, 2022]

[6] R. Wattenhofer, "Graph neural networks as application of distributed algorithms," in *ApPLIED@PODC*. ACM, 2022, p. 1.

[7] S. Farhadkhani, R. Guerraoui, N. Gupta, R. Pinot, and J. Stephan, "Byzantine machine learning made easy by resilient averaging of momentums," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 162. PMLR, 2022, pp. 6246–6283.

[8] N. Gupta and N. H. Vaidya, "Byzantine fault-tolerant parallelized stochastic gradient descent for linear regression," in *Allerton*. IEEE, 2019, pp. 415–420.

[9] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, "Renaissance: A self-stabilizing distributed SDN control plane using in-band communications," *J. Comput. Syst. Sci.*, vol. 127, pp. 91–121, 2022.

[10] O. Lundström, M. Raynal, and E. M. Schiller, "Self-stabilizing multivalued consensus in asynchronous crash-prone systems," in *EDCC*. IEEE, 2021, pp. 111–118.

[11] ——, "Self-stabilizing indulgent zero-degrading binary consensus," in *ICDCN*. ACM, 2021, pp. 106–115.

[12] ——, "Self-stabilizing set-constrained delivery broadcast (extended abstract)," in *ICDCS*. IEEE, 2020, pp. 617–627.

[13] ——, "Self-stabilizing uniform reliable broadcast," in *NETYS*, ser. Lecture Notes in Computer Science, vol. 12129. Springer, 2020, pp. 296–313.

[14] R. Duvignau, M. Raynal, and E. M. Schiller, "Self-stabilizing Byzantine-tolerant broadcast," *CoRR*, vol. abs/2201.12880, 2022.

[15] C. Georgiou, I. Marcoullis, M. Raynal, and E. M. Schiller, "Loosely-self-stabilizing Byzantine-tolerant binary consensus for signature-free message-passing systems," in *NETYS*, ser. Lecture Notes in Computer Science, vol. 12754. Springer, 2021, pp. 36–53.

[16] R. Duvignau, M. Raynal, and E. M. Schiller, "Self-stabilizing Byzantine- and intrusion-tolerant consensus," *CoRR*, vol. abs/2110.08592, 2021.

[17] S. Dolev, C. Georgiou, I. Marcoullis, and E. M. Schiller, "Self-stabilizing Byzantine tolerant replicated state machine based on failure detectors," in *CSCML*, ser. Lecture Notes in Computer Science, vol. 10879. Springer, 2018, pp. 84–100.

[18] S. Dolev, O. Liba, and E. M. Schiller, "Self-stabilizing Byzantine resilient topology discovery and message delivery - (extended abstract)," in *NETYS*, ser. Lecture Notes in Computer Science, vol. 7853. Springer, 2013, pp. 42–57.

[19] J. Aspnes, "Time- and space-efficient randomized consensus," *J. Algorithms*, vol. 14, no. 3, pp. 414–431, 1993. [Online]. Available: https://doi.org/10.1006/jagm.1993.1022 [Accessed: Jul. 1, 2022]

[20] 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*, ser. Lecture Notes in Computer Science, G. Noubir and M. Raynal, Eds., vol. 8593. Springer, 2014, pp. 99–121. [Online]. Available: https://doi.org/10.1007/978-3-319-09581-3_8 [Accessed: Jul. 1, 2022]

[21] J. Beauquier and S. Kekkonen-Moneta, "Fault-tolerance and self-stabilization: impossibility results and solutions using self-stabilizing failure detectors," *Int. J. Syst. Sci.*, vol. 28, no. 11, pp. 1177–1187, 1997. [Online]. Available: https://doi.org/10.1080/00207729708929476 [Accessed: Jul. 1, 2022]

[22] 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. [Online]. Available: https://doi.org/10.1109/TC.2002.1004595 [Accessed: Jul. 1, 2022]

[23] V. K. Ayyadevara, *Pro Machine Learning Algorithms : A Hands-On Approach to Implementing Algorithms in Python and R*. Berkeley, CA: Apress, 2018. [Online]. Available: https://doi.org/10.1007/978-1-4842-3564-5 [Accessed: Jul. 1, 2022]

[24] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1,

pp. 119–139, 1997. [Online]. Available: https://doi.org/10.1006/jcss.1997.1504 [Accessed: Jul. 1, 2022]

[25] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729586 [Accessed: Jul. 1, 2022]

[26] 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. [Online]. Available: https://doi.org/10.1214/aoms/1177729392 [Accessed: Jul. 1, 2022]

[27] Vikramkumar, B. Vijaykumar, and Trilochan, "Bayes and naive bayes classifier," 2014. [Online]. Available: https://arxiv.org/abs/1404.0933 [Accessed: Jul. 1, 2022]

[28] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: https://doi.org/10.1007/BF00994018 [Accessed: Jul. 1, 2022]

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735 [Accessed: Jul. 1, 2022]

[30] G. V. Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5929–5955, 2020. [Online]. Available: https://doi.org/10.1007/s10462-020-09838-1 [Accessed: Jul. 1, 2022]

[31] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, ser. Proceedings of Machine Learning Research, A. Singh and X. J. Zhu, Eds., vol. 54. PMLR, 2017, pp. 1273–1282. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html [Accessed: Jul. 1, 2022]

[32] 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 [Accessed: Jul. 1, 2022]

[33] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006. [Online]. Available: https://doi.org/10.1016/j.patrec.2005.10.010 [Accessed: Jul. 1, 2022]

[34] 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*, ser.

Lecture Notes in Computer Science, H. Blockeel, K. Kersting, S. Nijssen, and F. Zelezný, Eds., vol. 8190. Springer, 2013, pp. 451–466. [Online]. Available: https://doi.org/10.1007/978-3-642-40994-3_29 [Accessed: Jul. 1, 2022]

[35] L. Lamport, R. E. Shostak, and M. C. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982. [Online]. Available: http://doi.acm.org/10.1145/357172.357176 [Accessed: Jul. 1, 2022]

[36] PlanetLabEurope. Home. [Online]. Available: https://www.planet-lab.eu/Home [Accessed: Jul. 1, 2022]

[37] J. Brownlee. (2016) Data leakage in machine learning. [Online]. Available: https://machinelearningmastery.com/data-leakage-machine-learning/ [Accessed: Jul. 1, 2022]

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf [Accessed: Jul. 1, 2022]

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://dl.acm.org/doi/10.5555/1953048.2078195 [Accessed: Jul. 1, 2022]

A

Additional Results

A.1 Machine Learning Metric Results

Table A.1 shows the confusion metric results as well as the derived ML metrics for each classifier over the test set.

Classifier	FP	TP	FN	TN	Accuracy	Sensitivity	Specificity	Precision	Fall-Out	AUROC
AdaBoost	579	3728578	144	12117790	0.999 954	0.999 961	0.999 952	0.999 845	0.000 048	0.999 999 959
AdaBoost PR	271	3726732	1990	12118098	0.999 857	0.999 466	0.999 978	0.999 927	0.000 022	0.999 999 957
AdaBoost ROC	708	3728684	38	12117661	0.999 953	0.999 990	0.999 942	0.999 810	0.000 058	0.999 999 957
Decision Tree	866	3728701	21	12117503	0.999 944	0.999 994	0.999 929	0.999 768	0.000 071	0.999 982 482
Decision Tree PR	424	3726179	2543	12117945	0.999 813	0.999 318	0.999 965	0.999 886	0.000 035	0.999 982 481
Decision Tree ROC	424	3726197	2525	12117945	0.999 814	0.999 323	0.999 965	0.999 886	0.000 035	0.999 982 481
FedDyn	10995	3737877	2632	12107374	0.999 141	0.999 296	0.999 093	0.997 067	0.000 907	0.999 194 526
Logistic Regression	857	3728237	485	12117512	0.999 915	0.999 870	0.999 929	0.999 770	0.000 071	0.999 999 202
Logistic Regression PR	519	3726413	2309	12117850	0.999 822	0.999 381	0.999 957	0.999 861	0.000 043	0.999 999 212
Logistic Regression ROC	988	3728359	363	12117381	0.999 915	0.999 903	0.999 918	0.999 735	0.000 082	0.999 999 212
LSTM	899	3728573	26	12109602	0.999 942	0.999 993	0.999 926	0.999 759	0.000 074	0.999 959 397
Naive Bayes	1042	3717473	11249	12117327	0.999 224	0.996 983	0.999 914	0.999 720	0.000 086	0.999 718 598
Naive Bayes PR	1623	3723333	5389	12116746	0.999 558	0.998 555	0.999 866	0.999 564	0.000 134	0.999 719 211
Naive Bayes ROC	2930	3726889	1833	12115439	0.999 699	0.999 508	0.999 758	0.999 214	0.000 242	0.999 719 211
NN	830	3728640	82	12117539	0.999 942	0.999 978	0.999 932	0.999 777	0.000 068	0.999 954 759
Random Forest	490	3728680	42	12117879	0.999 966	0.999 989	0.999 960	0.999 869	0.000 040	0.999 998 050
Random Forest PR	240	3728537	185	12118129	0.999 973	0.999 950	0.999 980	0.999 936	0.000 020	0.999 998 173
Random Forest ROC	422	3728650	72	12117947	0.999 969	0.999 981	0.999 965	0.999 887	0.000 035	0.999 998 173
SVM	779	3728695	27	12117590	0.999 949	0.999 993	0.999 936	0.999 791	0.000 064	0.999 998 222
SVM PR	514	3727070	1652	12117855	0.999 863	0.999 557	0.999 958	0.999 862	0.000 042	0.999 997 403
SVM ROC	754	3728677	45	12117615	0.999 950	0.999 988	0.999 938	0.999 798	0.000 062	0.999 997 403

Table A.1: The confusion matrix results alongside the derived ML metrics, for each classifier. Sorted by name.

A.2 Detection Time Comparisons by Mistake Rate

Table A.2 shows, for each FD, the highest λ_M that is lower than `level`, where `level` $\in \{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}\}$. The table also lists the corresponding T_D for each λ_M, as well as the T_D normalized by the BDBD solution's T_D, referred to as $BDBD_{T_D}$, for the same `level`. The table is sorted first by `level`, and then by the normalized T_D, $\frac{T_D}{BDBD_{T_D}}$. For each `level`, only one result

is displayed per FD from the configurations C_{PD}, $C_{fallback,lower}$, $C_{fallback,equal}$, and $C_{fallback,higher}$, however the individual results of each configuration are shown in Tables A.3, A.4, A.5, and A.6, respectively. The column 'FD (config)' indicates from which configuration each result is from. Within the parenthesis in the column, t signifies the target specificity of the PD module, and Θ the target specificity of the BDBD solution used as a fallback. If no parenthesis is added, the result is from an FD without any enhancements. A single t indicates that the result is from C_{PD}, and an accompanying Θ that it is from $C_{fallback}$, where the set threshold provides further indication as to what $C_{fallback}$ configuration it was taken from.

We note that for `level` = 1×10^{-6}, the PR curve optimized Random Forest has a T_D that is 88.7% lower than the corresponding T_D of the BDBD solution, at the same exact λ_M. Further, we note that for `level` = 1×10^{-4}, a large majority of the best results for each FD are from $C_{fallback,equal}$, with T_D improvements ranging between $10 - 73\%$ in comparison to the BDBD solution. Finally, we note that the values in this table were not based on interpolation, and as such were instead based on measured results.

FD (config)	$\lambda_M \leq$ `level`	λ_M	T_D (s)	$\frac{T_D}{BDBD_{T_D}}$
Random Forest ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.721×10^{-4}	0.275	0.304
SVM ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.690×10^{-4}	0.275	0.304
Random Forest ROC ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.668×10^{-4}	0.275	0.304
SVM ROC ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.744×10^{-4}	0.275	0.304
AdaBoost ROC ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.670×10^{-4}	0.275	0.305
NN ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.768×10^{-4}	0.276	0.305
Random Forest PR ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.570×10^{-4}	0.278	0.308
AdaBoost ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.678×10^{-4}	0.279	0.309
Ensemble ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.548×10^{-4}	0.286	0.316
SVM PR ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.645×10^{-4}	0.334	0.369
LSTM ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.562×10^{-4}	0.343	0.380
Logistic Regression ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.569×10^{-4}	0.35	0.387
AdaBoost PR ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.572×10^{-4}	0.355	0.393
Decision Tree ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.504×10^{-4}	0.361	0.400
Logistic Regression PR ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.641×10^{-4}	0.368	0.407
Decision Tree ROC ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.547×10^{-4}	0.395	0.438
Decision Tree PR ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.547×10^{-4}	0.395	0.438
FedDyn ($t = 0$)	$\lambda_M \leq 1 \times 10^{-3}$	6.842×10^{-4}	0.402	0.445
Logistic Regression ROC ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.567×10^{-4}	0.416	0.461
TTF ($t = 0.999$)	$\lambda_M \leq 1 \times 10^{-3}$	3.682×10^{-4}	0.572	0.633
Naive Bayes ROC ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.500×10^{-4}	0.659	0.729
Naive Bayes PR ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.503×10^{-4}	0.675	0.747
Naive Bayes ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.489×10^{-4}	0.729	0.807
BDBD ($t = 0.999$)	$\lambda_M \leq 1 \times 10^{-3}$	1.217×10^{-4}	0.903	1.
Random Forest ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.813×10^{-5}	0.275	0.265
Random Forest ROC ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.288×10^{-5}	0.275	0.265
AdaBoost ROC ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.893×10^{-5}	0.275	0.265
LSTM ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.770×10^{-5}	0.276	0.266
NN ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.801×10^{-5}	0.276	0.266

Ensemble ($t = 0.9995, \Theta = 0.9995$)	$\lambda_M \leq 1 \times 10^{-4}$	9.319×10^{-5}	0.278	0.268
Random Forest PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.313×10^{-5}	0.279	0.268
AdaBoost ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.374×10^{-5}	0.28	0.269
Logistic Regression ROC ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	9.115×10^{-5}	0.291	0.280
Logistic Regression ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	8.041×10^{-5}	0.297	0.286
SVM PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.035×10^{-5}	0.337	0.325
AdaBoost PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.294×10^{-5}	0.358	0.345
Logistic Regression PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.004×10^{-5}	0.373	0.359
Decision Tree ROC ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.060×10^{-5}	0.396	0.381
Decision Tree PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.060×10^{-5}	0.396	0.381
Naive Bayes ROC ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	9.646×10^{-5}	0.422	0.406
Naive Bayes PR ($t = 0.999$)	$\lambda_M \leq 1 \times 10^{-4}$	8.060×10^{-5}	0.535	0.515
FedDyn ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	8.480×10^{-5}	0.586	0.564
Decision Tree ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.430×10^{-5}	0.767	0.738
Naive Bayes ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.906×10^{-5}	0.774	0.745
TTF ($t = 0.9997$)	$\lambda_M \leq 1 \times 10^{-4}$	5.289×10^{-5}	0.808	0.778
SVM ROC ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.437×10^{-5}	0.885	0.852
SVM ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.430×10^{-5}	0.931	0.897
BDBD ($t = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
Decision Tree ROC ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	8.578×10^{-6}	0.664	0.312
AdaBoost PR ($t = 0.99996$)	$\lambda_M \leq 1 \times 10^{-5}$	7.714×10^{-6}	0.696	0.327
Random Forest PR ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.901×10^{-6}	0.75	0.352
Decision Tree ($t = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	8.887×10^{-6}	0.762	0.358
Random Forest ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	6.974×10^{-6}	0.806	0.379
Decision Tree PR ($t = 0.99992, \Theta = 0.99992$)	$\lambda_M \leq 1 \times 10^{-5}$	6.418×10^{-6}	0.907	0.426
FedDyn ($t = 0.9997$)	$\lambda_M \leq 1 \times 10^{-5}$	9.627×10^{-6}	0.986	0.463
Random Forest ROC ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	3.086×10^{-6}	1.050	0.493
Naive Bayes ($t = 0.9998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.986×10^{-6}	1.087	0.511
Naive Bayes ROC ($t = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	9.442×10^{-6}	1.207	0.567
AdaBoost ROC ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.493×10^{-6}	2.086	0.980
AdaBoost ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.246×10^{-6}	2.086	0.980
SVM PR ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.092	0.983
SVM ROC ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.108	0.991
SVM ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.109	0.991
Ensemble ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.109	0.991
Logistic Regression PR ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Logistic Regression ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Logistic Regression ROC ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
LSTM ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Naive Bayes PR ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
NN ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
BDBD ($t = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
TTF ($t = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.469×10^{-7}	8.515	4.001
Random Forest PR ($t = 0.999996$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	1.457	0.113
Random Forest ROC ($t = 0.999992, \Theta = 0.99992$)	$\lambda_M \leq 1 \times 10^{-6}$	8.640×10^{-7}	1.569	0.122
Ensemble ($t = 0.99991$)	$\lambda_M \leq 1 \times 10^{-6}$	6.789×10^{-7}	3.506	0.272
Random Forest ($t = 0.999992$)	$\lambda_M \leq 1 \times 10^{-6}$	9.257×10^{-7}	3.708	0.287

SVM PR (t = 0.99993)	$\lambda_M \leq 1 \times 10^{-6}$	9.874×10^{-7}	6.324	0.490
LSTM (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.335	0.569
Logistic Regression (t = 0.99992)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.342	0.569
Logistic Regression ROC (t = 0.99992)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.342	0.569
FedDyn (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
Naive Bayes (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
Naive Bayes ROC (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
Naive Bayes PR (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
TTF (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	8.515	0.660
AdaBoost PR (t = 0.99999, Θ = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	11.030	0.855
Decision Tree PR (t = 0.99999, Θ = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	11.04	0.856
AdaBoost (t = 0.99994)	$\lambda_M \leq 1 \times 10^{-6}$	4.937×10^{-7}	11.729	0.909
Decision Tree (t = 0.99999, Θ = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	3.703×10^{-7}	11.868	0.920
Decision Tree ROC (t = 0.99999, Θ = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	11.868	0.920
AdaBoost ROC (t = 0.99999, Θ = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	6.789×10^{-7}	12.897	1.
Logistic Regression PR (t = 0.99999, Θ = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
NN (t = 0.99999, Θ = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
BDBD (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.898	1.
SVM ROC (t = 0.99992, Θ = 0.99992)	$\lambda_M \leq 1 \times 10^{-6}$	3.703×10^{-7}	14.261	1.106
SVM (t = 0.99992, Θ = 0.99992)	$\lambda_M \leq 1 \times 10^{-6}$	3.086×10^{-7}	14.81	1.148

Table A.2: The lowest T_D for each FD, in one of C_{PD}, $C_{fallback,lower}$, $C_{fallback,equal}$, $C_{fallback,higher}$, also normalized by the BDBD solution's T_D, given that a λ_M of at most `level` is acceptable.

FD (config)	$\lambda_M \leq$ `level`	λ_M	T_D (s)	$\frac{T_D}{BDBD_{T_D}}$
AdaBoost ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	3.937×10^{-5}	0.276	0.306
SVM (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	4.271×10^{-5}	0.278	0.308
Random Forest (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	3.746×10^{-5}	0.278	0.308
Ensemble (t = 0)	$\lambda_M \leq 1 \times 10^{-3}$	3.339×10^{-5}	0.278	0.308
LSTM (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	5.110×10^{-5}	0.279	0.308
SVM ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	4.166×10^{-5}	0.279	0.308
Random Forest ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	3.129×10^{-5}	0.28	0.31
NN (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	4.869×10^{-5}	0.280	0.310
Decision Tree (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	4.826×10^{-5}	0.284	0.314
Logistic Regression ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	6.208×10^{-5}	0.292	0.323
Logistic Regression (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	5.122×10^{-5}	0.298	0.329
AdaBoost (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	3.407×10^{-5}	0.311	0.344
Random Forest PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	1.876×10^{-5}	0.315	0.349
Naive Bayes ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	1.521×10^{-4}	0.363	0.401
Logistic Regression PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	2.771×10^{-5}	0.389	0.431
FedDyn (t = 0)	$\lambda_M \leq 1 \times 10^{-3}$	6.842×10^{-4}	0.402	0.445
Decision Tree ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	2.154×10^{-5}	0.405	0.449
Decision Tree PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	2.154×10^{-5}	0.431	0.477
SVM PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	2.771×10^{-5}	0.446	0.494
Naive Bayes PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	8.060×10^{-5}	0.535	0.592
AdaBoost PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	1.975×10^{-5}	0.537	0.595

TTF (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	3.682×10^{-4}	0.572	0.633
Naive Bayes (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	4.647×10^{-5}	0.817	0.905
BDBD (t = 0.999)	$\lambda_M \leq 1 \times 10^{-3}$	1.217×10^{-4}	0.903	1.
AdaBoost ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	3.937×10^{-5}	0.276	0.266
SVM (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	4.271×10^{-5}	0.278	0.268
Random Forest (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	3.746×10^{-5}	0.278	0.268
Ensemble (t = 0)	$\lambda_M \leq 1 \times 10^{-4}$	3.339×10^{-5}	0.278	0.268
LSTM (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	5.110×10^{-5}	0.279	0.268
SVM ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	4.166×10^{-5}	0.279	0.268
Random Forest ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	3.129×10^{-5}	0.28	0.27
NN (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	4.869×10^{-5}	0.280	0.27
Decision Tree (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	4.826×10^{-5}	0.284	0.274
Logistic Regression ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	6.208×10^{-5}	0.292	0.281
Logistic Regression (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	5.122×10^{-5}	0.298	0.287
AdaBoost (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	3.407×10^{-5}	0.311	0.299
Random Forest PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	1.876×10^{-5}	0.315	0.304
Logistic Regression PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	2.771×10^{-5}	0.389	0.375
Decision Tree ROC (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	2.154×10^{-5}	0.405	0.390
Decision Tree PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	2.154×10^{-5}	0.431	0.415
Naive Bayes ROC (t = 0.9996)	$\lambda_M \leq 1 \times 10^{-4}$	7.134×10^{-5}	0.435	0.419
SVM PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	2.771×10^{-5}	0.446	0.430
Naive Bayes PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	8.060×10^{-5}	0.535	0.515
AdaBoost PR (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	1.975×10^{-5}	0.537	0.518
FedDyn (t = 0.9996)	$\lambda_M \leq 1 \times 10^{-4}$	6.424×10^{-5}	0.593	0.571
TTF (t = 0.9997)	$\lambda_M \leq 1 \times 10^{-4}$	5.289×10^{-5}	0.808	0.778
Naive Bayes (t = 0.999)	$\lambda_M \leq 1 \times 10^{-4}$	4.647×10^{-5}	0.817	0.787
BDBD (t = 0.9994)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
Decision Tree ROC (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	8.455×10^{-6}	0.672	0.316
AdaBoost PR (t = 0.99996)	$\lambda_M \leq 1 \times 10^{-5}$	7.714×10^{-6}	0.696	0.327
Decision Tree (t = 0.9998)	$\lambda_M \leq 1 \times 10^{-5}$	8.887×10^{-6}	0.762	0.358
Random Forest PR (t = 0.99999)	$\lambda_M \leq 1 \times 10^{-5}$	2.654×10^{-6}	0.781	0.367
Random Forest (t = 0.99998)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	0.867	0.407
Decision Tree PR (t = 0.99992)	$\lambda_M \leq 1 \times 10^{-5}$	6.295×10^{-6}	0.941	0.442
FedDyn (t = 0.9997)	$\lambda_M \leq 1 \times 10^{-5}$	9.627×10^{-6}	0.986	0.463
Random Forest ROC (t = 0.99999)	$\lambda_M \leq 1 \times 10^{-5}$	2.839×10^{-6}	1.052	0.494
Naive Bayes (t = 0.9998)	$\lambda_M \leq 1 \times 10^{-5}$	4.505×10^{-6}	1.137	0.534
AdaBoost (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	4.999×10^{-6}	1.178	0.554
Naive Bayes ROC (t = 0.9998)	$\lambda_M \leq 1 \times 10^{-5}$	9.442×10^{-6}	1.207	0.567
SVM PR (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	3.765×10^{-6}	1.504	0.707
AdaBoost ROC (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	3.271×10^{-6}	2.126	0.999
BDBD (t = 0.9998)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Logistic Regression PR (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	3.209×10^{-6}	2.406	1.130
Ensemble (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	1.975×10^{-6}	2.445	1.149
SVM ROC (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	2.715×10^{-6}	3.106	1.459
Logistic Regression (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	1.913×10^{-6}	3.295	1.548
SVM (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	2.098×10^{-6}	3.683	1.731
Logistic Regression ROC (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	1.543×10^{-6}	3.721	1.749

NN (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	1.543×10^{-6}	4.910	2.307
LSTM (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	1.111×10^{-6}	5.730	2.693
Naive Bayes PR (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	1.049×10^{-6}	6.022	2.83
TTF (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-5}$	2.469×10^{-7}	8.515	4.001
Random Forest PR (t = 0.999996)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	1.457	0.113
Random Forest ROC (t = 0.999992)	$\lambda_M \leq 1 \times 10^{-6}$	7.406×10^{-7}	1.571	0.122
Ensemble (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	6.789×10^{-7}	3.506	0.272
Random Forest (t = 0.999992)	$\lambda_M \leq 1 \times 10^{-6}$	9.257×10^{-7}	3.708	0.287
SVM PR (t = 0.99993)	$\lambda_M \leq 1 \times 10^{-6}$	9.874×10^{-7}	6.324	0.490
LSTM (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.335	0.569
Logistic Regression (t = 0.99992)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.342	0.569
Logistic Regression ROC (t = 0.99992)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.342	0.569
FedDyn (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
Naive Bayes (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
Naive Bayes ROC (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
Naive Bayes PR (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
TTF (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	8.515	0.660
Decision Tree PR (t = 0.99997)	$\lambda_M \leq 1 \times 10^{-6}$	1.234×10^{-7}	10.696	0.829
AdaBoost (t = 0.99994)	$\lambda_M \leq 1 \times 10^{-6}$	4.937×10^{-7}	11.729	0.909
BDBD (t = 0.9999)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.898	1.
AdaBoost ROC (t = 0.99993)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	15.719	1.219
Decision Tree (t = 0.99995)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	17.779	1.378
Decision Tree ROC (t = 0.99995)	$\lambda_M \leq 1 \times 10^{-6}$	1.234×10^{-7}	17.779	1.378
Logistic Regression PR (t = 0.99993)	$\lambda_M \leq 1 \times 10^{-6}$	0.000	21.898	1.698
SVM ROC (t = 0.99992)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	23.561	1.827
SVM (t = 0.99992)	$\lambda_M \leq 1 \times 10^{-6}$	1.851×10^{-7}	23.718	1.839
NN (t = 0.99991)	$\lambda_M \leq 1 \times 10^{-6}$	0.000	24.936	1.933
AdaBoost PR (t = 0.99999)	$\lambda_M \leq 1 \times 10^{-6}$	0.000	88.307	6.847

Table A.3: The lowest T_D for each FD in C_{PD}, also normalized by the BDBD solution's T_D, given that a λ_M of at most `level` is acceptable.

FD (config)	$\lambda_M \leq$ `level`	λ_M	T_D (s)	$\frac{T_D}{BDBD_{T_D}}$
Random Forest (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.721×10^{-4}	0.275	0.304
SVM (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.690×10^{-4}	0.275	0.304
Random Forest ROC (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.668×10^{-4}	0.275	0.304
SVM ROC (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.744×10^{-4}	0.275	0.304
AdaBoost ROC (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.670×10^{-4}	0.275	0.305
NN (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.768×10^{-4}	0.276	0.305
Random Forest PR (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.570×10^{-4}	0.278	0.308
AdaBoost (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.678×10^{-4}	0.279	0.309
Ensemble (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.548×10^{-4}	0.286	0.316
SVM PR (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.645×10^{-4}	0.334	0.369
LSTM (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.562×10^{-4}	0.343	0.380
Logistic Regression (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.569×10^{-4}	0.35	0.387
AdaBoost PR (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.572×10^{-4}	0.355	0.393
Decision Tree (t = 0.9998, Θ = 0.998)	$\lambda_M \leq 1 \times 10^{-3}$	5.504×10^{-4}	0.361	0.400

Logistic Regression PR ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.641×10^{-4}	0.368	0.407
Decision Tree ROC ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.547×10^{-4}	0.395	0.438
Decision Tree PR ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.547×10^{-4}	0.395	0.438
Logistic Regression ROC ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.567×10^{-4}	0.416	0.461
TTF ($t = 0.999, \Theta = 0.99$)	$\lambda_M \leq 1 \times 10^{-3}$	3.682×10^{-4}	0.572	0.633
Naive Bayes ROC ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.500×10^{-4}	0.659	0.729
Naive Bayes PR ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.503×10^{-4}	0.675	0.747
Naive Bayes ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.489×10^{-4}	0.729	0.807
FedDyn ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-3}$	5.490×10^{-4}	0.731	0.809
BDBD ($t = 0.9991, \Theta = 0.991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.217×10^{-4}	0.903	1.
Random Forest ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.813×10^{-5}	0.275	0.265
Random Forest ROC ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.288×10^{-5}	0.275	0.265
Random Forest PR ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.313×10^{-5}	0.279	0.268
AdaBoost PR ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.294×10^{-5}	0.358	0.345
Decision Tree PR ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.430×10^{-5}	0.697	0.672
Decision Tree ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.430×10^{-5}	0.767	0.738
AdaBoost ROC ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.467×10^{-5}	0.768	0.740
AdaBoost ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.443×10^{-5}	0.769	0.741
SVM PR ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	0.795	0.766
TTF ($t = 0.9997, \Theta = 0.997$)	$\lambda_M \leq 1 \times 10^{-4}$	5.289×10^{-5}	0.808	0.778
Decision Tree ROC ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	0.812	0.782
SVM ROC ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.437×10^{-5}	0.885	0.852
Ensemble ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	0.905	0.872
SVM ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.430×10^{-5}	0.931	0.897
Logistic Regression PR ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.036	0.998
FedDyn ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
Logistic Regression ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
Logistic Regression ROC ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
LSTM ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
Naive Bayes ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
Naive Bayes ROC ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
Naive Bayes PR ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
NN ($t = 0.99994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
BDBD ($t = 0.9994, \Theta = 0.994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
Random Forest PR ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.901×10^{-6}	0.75	0.352
Random Forest ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	6.974×10^{-6}	0.806	0.379
Random Forest ROC ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	3.086×10^{-6}	1.050	0.493
AdaBoost PR ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.246×10^{-6}	1.482	0.697
AdaBoost ROC ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.493×10^{-6}	2.086	0.980
Decision Tree ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.122×10^{-6}	2.086	0.980
AdaBoost ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.246×10^{-6}	2.086	0.980
Decision Tree ROC ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.089	0.982
Decision Tree PR ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.089	0.982
SVM PR ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.092	0.983
SVM ROC ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.108	0.991
SVM ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.109	0.991
Ensemble ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.109	0.991

Logistic Regression PR ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
FedDyn ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Logistic Regression ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Logistic Regression ROC ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
LSTM ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Naive Bayes ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Naive Bayes ROC ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Naive Bayes PR ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
NN ($t = 0.99998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
BDBD ($t = 0.9998, \Theta = 0.998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
TTF ($t = 0.9999, \Theta = 0.999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.469×10^{-7}	8.515	4.001
Random Forest PR ($t = 0.999996, \Theta = 0.99996$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	1.441	0.112
Random Forest ROC ($t = 0.999992, \Theta = 0.99992$)	$\lambda_M \leq 1 \times 10^{-6}$	8.640×10^{-7}	1.569	0.122
Random Forest ($t = 0.999992, \Theta = 0.99992$)	$\lambda_M \leq 1 \times 10^{-6}$	9.257×10^{-7}	2.734	0.212
TTF ($t = 0.9999, \Theta = 0.999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	8.515	0.660
AdaBoost PR ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	11.030	0.855
Decision Tree PR ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	11.04	0.856
Decision Tree ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	3.703×10^{-7}	11.868	0.920
Decision Tree ROC ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	11.868	0.920
AdaBoost ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	4.320×10^{-7}	12.897	1.
AdaBoost ROC ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	6.789×10^{-7}	12.897	1.
FedDyn ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
Logistic Regression ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
Logistic Regression ROC ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
Logistic Regression PR ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
LSTM ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
Naive Bayes ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
Naive Bayes ROC ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
Naive Bayes PR ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
NN ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
SVM ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
SVM ROC ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
SVM PR ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
Ensemble ($t = 0.99999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.897	1.
BDBD ($t = 0.9999, \Theta = 0.999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.898	1.

Table A.4: The lowest T_D for each FD in $C_{fallback,lower}$, also normalized by the BDBD solution's T_D, given that a λ_M of at most `level` is acceptable.

FD (config)	$\lambda_M \leq$ `level`	λ_M	T_D (s)	$\frac{T_D}{BDBD_{T_D}}$
Decision Tree ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.518×10^{-4}	0.274	0.304
SVM ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.494×10^{-4}	0.275	0.304
Random Forest ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.453×10^{-4}	0.275	0.304
Random Forest ROC ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.400×10^{-4}	0.275	0.304
SVM ROC ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.489×10^{-4}	0.275	0.305
AdaBoost ROC ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.461×10^{-4}	0.275	0.305
LSTM ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.551×10^{-4}	0.275	0.305

NN ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.555×10^{-4}	0.276	0.305
Ensemble ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.403×10^{-4}	0.278	0.308
Random Forest PR ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.303×10^{-4}	0.278	0.308
AdaBoost ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.410×10^{-4}	0.279	0.309
Logistic Regression ROC ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.681×10^{-4}	0.291	0.322
Logistic Regression ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.577×10^{-4}	0.296	0.328
SVM PR ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.376×10^{-4}	0.335	0.371
AdaBoost PR ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.303×10^{-4}	0.356	0.394
Naive Bayes ROC ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	2.468×10^{-4}	0.36	0.399
Logistic Regression PR ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.372×10^{-4}	0.369	0.409
Decision Tree ROC ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.278×10^{-4}	0.395	0.438
Decision Tree PR ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.278×10^{-4}	0.395	0.438
FedDyn ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	4.228×10^{-4}	0.406	0.449
Naive Bayes PR ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.789×10^{-4}	0.497	0.550
TTF ($t = 0.999, \Theta = 0.999$)	$\lambda_M \leq 1 \times 10^{-3}$	3.682×10^{-4}	0.572	0.633
Naive Bayes ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.532×10^{-4}	0.704	0.779
BDBD ($t = 0.9991, \Theta = 0.9991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.217×10^{-4}	0.903	1.
Random Forest ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.813×10^{-5}	0.275	0.265
SVM ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.134×10^{-5}	0.275	0.265
Decision Tree ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.356×10^{-5}	0.275	0.265
Random Forest ROC ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.288×10^{-5}	0.275	0.265
SVM ROC ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.072×10^{-5}	0.275	0.265
AdaBoost ROC ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.893×10^{-5}	0.275	0.265
LSTM ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.770×10^{-5}	0.276	0.266
NN ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.801×10^{-5}	0.276	0.266
Ensemble ($t = 0.9995, \Theta = 0.9995$)	$\lambda_M \leq 1 \times 10^{-4}$	9.319×10^{-5}	0.278	0.268
Random Forest PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.313×10^{-5}	0.279	0.268
AdaBoost ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.374×10^{-5}	0.28	0.269
Logistic Regression ROC ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	9.115×10^{-5}	0.291	0.280
Logistic Regression ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	8.041×10^{-5}	0.297	0.286
SVM PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.035×10^{-5}	0.337	0.325
AdaBoost PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.294×10^{-5}	0.358	0.345
Logistic Regression PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	9.004×10^{-5}	0.373	0.359
Decision Tree ROC ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.060×10^{-5}	0.396	0.381
Decision Tree PR ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	8.060×10^{-5}	0.396	0.381
Naive Bayes ROC ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	9.646×10^{-5}	0.422	0.406
FedDyn ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	8.480×10^{-5}	0.586	0.564
Naive Bayes ($t = 0.9996, \Theta = 0.9996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.906×10^{-5}	0.774	0.745
Naive Bayes PR ($t = 0.9997, \Theta = 0.9997$)	$\lambda_M \leq 1 \times 10^{-4}$	3.839×10^{-5}	0.779	0.750
TTF ($t = 0.9997, \Theta = 0.9997$)	$\lambda_M \leq 1 \times 10^{-4}$	5.289×10^{-5}	0.808	0.778
BDBD ($t = 0.9994, \Theta = 0.9994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.
AdaBoost PR ($t = 0.99996, \Theta = 0.99996$)	$\lambda_M \leq 1 \times 10^{-5}$	7.714×10^{-6}	0.553	0.26
Decision Tree ROC ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	8.578×10^{-6}	0.664	0.312
Random Forest PR ($t = 0.999991, \Theta = 0.999991$)	$\lambda_M \leq 1 \times 10^{-5}$	2.654×10^{-6}	0.781	0.367
Random Forest ($t = 0.99998, \Theta = 0.99998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	0.867	0.407
Decision Tree PR ($t = 0.99992, \Theta = 0.99992$)	$\lambda_M \leq 1 \times 10^{-5}$	6.418×10^{-6}	0.907	0.426
Random Forest ROC ($t = 0.99999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.839×10^{-6}	1.052	0.494

Naive Bayes ($t = 0.9998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.986×10^{-6}	1.087	0.511
FedDyn ($t = 0.9998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	6.789×10^{-6}	1.101	0.517
AdaBoost ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	4.875×10^{-6}	1.15	0.540
SVM PR ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	3.765×10^{-6}	1.437	0.675
Decision Tree ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	3.641×10^{-6}	1.611	0.757
AdaBoost ROC ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	3.271×10^{-6}	2.126	0.999
BDBD ($t = 0.9998, \Theta = 0.9998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Logistic Regression PR ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	3.209×10^{-6}	2.164	1.017
Ensemble ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.975×10^{-6}	2.207	1.037
SVM ROC ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.715×10^{-6}	2.739	1.287
Logistic Regression ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.913×10^{-6}	3.048	1.432
Logistic Regression ROC ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.543×10^{-6}	3.359	1.578
SVM ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.098×10^{-6}	3.404	1.6
NN ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.543×10^{-6}	4.068	1.912
LSTM ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.111×10^{-6}	4.886	2.296
Naive Bayes ROC ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.049×10^{-6}	5.186	2.437
Naive Bayes PR ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.049×10^{-6}	5.186	2.437
TTF ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.469×10^{-7}	8.515	4.001
Random Forest PR ($t = 0.999996, \Theta = 0.999996$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	1.457	0.113
Random Forest ROC ($t = 0.999992, \Theta = 0.999992$)	$\lambda_M \leq 1 \times 10^{-6}$	7.406×10^{-7}	1.571	0.122
Ensemble ($t = 0.99991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-6}$	6.789×10^{-7}	3.506	0.272
Random Forest ($t = 0.999992, \Theta = 0.999992$)	$\lambda_M \leq 1 \times 10^{-6}$	9.257×10^{-7}	3.708	0.287
Logistic Regression ($t = 0.99992, \Theta = 0.99992$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	6.617	0.513
Logistic Regression ROC ($t = 0.99992, \Theta = 0.99992$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	6.617	0.513
LSTM ($t = 0.99991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	6.856	0.532
FedDyn ($t = 0.99991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.106	0.551
Naive Bayes ($t = 0.99991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.106	0.551
Naive Bayes ROC ($t = 0.99991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.106	0.551
Naive Bayes PR ($t = 0.99991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.106	0.551
TTF ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	8.515	0.660
AdaBoost ($t = 0.99994, \Theta = 0.99994$)	$\lambda_M \leq 1 \times 10^{-6}$	4.937×10^{-7}	10.074	0.781
Decision Tree PR ($t = 0.99997, \Theta = 0.99997$)	$\lambda_M \leq 1 \times 10^{-6}$	1.234×10^{-7}	10.667	0.827
Decision Tree ($t = 0.99995, \Theta = 0.99995$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	11.020	0.854
Decision Tree ROC ($t = 0.99995, \Theta = 0.99995$)	$\lambda_M \leq 1 \times 10^{-6}$	1.234×10^{-7}	11.020	0.854
BDBD ($t = 0.9999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.898	1.
SVM PR ($t = 0.99994, \Theta = 0.99994$)	$\lambda_M \leq 1 \times 10^{-6}$	6.171×10^{-8}	13.098	1.016
AdaBoost ROC ($t = 0.99993, \Theta = 0.99993$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	13.764	1.067
Logistic Regression PR ($t = 0.99993, \Theta = 0.99993$)	$\lambda_M \leq 1 \times 10^{-6}$	1.234×10^{-7}	13.863	1.075
SVM ROC ($t = 0.99992, \Theta = 0.99992$)	$\lambda_M \leq 1 \times 10^{-6}$	3.703×10^{-7}	14.261	1.106
SVM ($t = 0.99992, \Theta = 0.99992$)	$\lambda_M \leq 1 \times 10^{-6}$	3.086×10^{-7}	14.81	1.148
NN ($t = 0.99991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-6}$	1.234×10^{-7}	15.367	1.191
AdaBoost PR ($t = 0.999991, \Theta = 0.999991$)	$\lambda_M \leq 1 \times 10^{-6}$	0.000	88.307	6.847

Table A.5: The lowest T_D for each FD in $C_{fallback,equal}$, also normalized by the BDBD solution's T_D, given that a λ_M of at most `level` is acceptable.

FD (config)	$\lambda_M \leq$ `level`	λ_M	T_D (s)	$\frac{T_D}{BDBD_{T_D}}$

SVM ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	4.271×10^{-5}	0.275	0.305
Decision Tree ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	4.838×10^{-5}	0.276	0.305
AdaBoost ROC ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	3.937×10^{-5}	0.276	0.306
SVM ROC ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	4.166×10^{-5}	0.276	0.306
Random Forest ($t = 0.9993, \Theta = 0.99993$)	$\lambda_M \leq 1 \times 10^{-3}$	3.752×10^{-5}	0.277	0.306
Random Forest ROC ($t = 0.999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-3}$	3.154×10^{-5}	0.278	0.308
NN ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	4.869×10^{-5}	0.278	0.308
Ensemble ($t = 0.9995, \Theta = 0.99995$)	$\lambda_M \leq 1 \times 10^{-3}$	3.333×10^{-5}	0.278	0.308
LSTM ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	5.110×10^{-5}	0.279	0.308
AdaBoost ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	3.407×10^{-5}	0.282	0.313
Random Forest PR ($t = 0.999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-3}$	1.901×10^{-5}	0.284	0.314
Logistic Regression ROC ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	6.208×10^{-5}	0.292	0.323
Logistic Regression ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	5.122×10^{-5}	0.298	0.329
Naive Bayes ROC ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.521×10^{-4}	0.362	0.401
AdaBoost PR ($t = 0.9993, \Theta = 0.99993$)	$\lambda_M \leq 1 \times 10^{-3}$	1.981×10^{-5}	0.374	0.414
SVM PR ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	2.777×10^{-5}	0.379	0.419
Logistic Regression PR ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	2.771×10^{-5}	0.386	0.428
Decision Tree ROC ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	2.166×10^{-5}	0.397	0.439
Decision Tree PR ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	2.166×10^{-5}	0.397	0.439
FedDyn ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	3.416×10^{-4}	0.418	0.463
Naive Bayes PR ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	8.060×10^{-5}	0.535	0.592
TTF ($t = 0.999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-3}$	3.682×10^{-4}	0.572	0.633
Naive Bayes ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	4.647×10^{-5}	0.817	0.905
BDBD ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-3}$	1.217×10^{-4}	0.903	1.
SVM ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	4.271×10^{-5}	0.275	0.265
Decision Tree ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	4.838×10^{-5}	0.276	0.266
AdaBoost ROC ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	3.937×10^{-5}	0.276	0.266
SVM ROC ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	4.166×10^{-5}	0.276	0.266
Random Forest ($t = 0.9993, \Theta = 0.99993$)	$\lambda_M \leq 1 \times 10^{-4}$	3.752×10^{-5}	0.277	0.266
Random Forest ROC ($t = 0.999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-4}$	3.154×10^{-5}	0.278	0.268
NN ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	4.869×10^{-5}	0.278	0.268
Ensemble ($t = 0.9995, \Theta = 0.99995$)	$\lambda_M \leq 1 \times 10^{-4}$	3.333×10^{-5}	0.278	0.268
LSTM ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	5.110×10^{-5}	0.279	0.268
AdaBoost ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	3.407×10^{-5}	0.282	0.272
Random Forest PR ($t = 0.999, \Theta = 0.9999$)	$\lambda_M \leq 1 \times 10^{-4}$	1.901×10^{-5}	0.284	0.273
Logistic Regression ROC ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	6.208×10^{-5}	0.292	0.281
Logistic Regression ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	5.122×10^{-5}	0.298	0.287
AdaBoost PR ($t = 0.9993, \Theta = 0.99993$)	$\lambda_M \leq 1 \times 10^{-4}$	1.981×10^{-5}	0.374	0.360
SVM PR ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	2.777×10^{-5}	0.379	0.365
Logistic Regression PR ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	2.771×10^{-5}	0.386	0.372
Decision Tree ROC ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	2.166×10^{-5}	0.397	0.382
Decision Tree PR ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	2.166×10^{-5}	0.397	0.382
Naive Bayes ROC ($t = 0.9996, \Theta = 0.99996$)	$\lambda_M \leq 1 \times 10^{-4}$	7.134×10^{-5}	0.435	0.419
Naive Bayes PR ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	8.060×10^{-5}	0.535	0.515
FedDyn ($t = 0.9996, \Theta = 0.99996$)	$\lambda_M \leq 1 \times 10^{-4}$	6.424×10^{-5}	0.593	0.571
TTF ($t = 0.9997, \Theta = 0.99997$)	$\lambda_M \leq 1 \times 10^{-4}$	5.289×10^{-5}	0.808	0.778
Naive Bayes ($t = 0.9991, \Theta = 0.99991$)	$\lambda_M \leq 1 \times 10^{-4}$	4.647×10^{-5}	0.817	0.787
BDBD ($t = 0.9994, \Theta = 0.99994$)	$\lambda_M \leq 1 \times 10^{-4}$	7.424×10^{-5}	1.038	1.

Decision Tree ROC ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	8.455×10^{-6}	0.672	0.316
AdaBoost PR ($t = 0.99996, \Theta = 0.999996$)	$\lambda_M \leq 1 \times 10^{-5}$	7.714×10^{-6}	0.696	0.327
Decision Tree ($t = 0.9998, \Theta = 0.99998$)	$\lambda_M \leq 1 \times 10^{-5}$	8.887×10^{-6}	0.762	0.358
Random Forest PR ($t = 0.999991, \Theta = 0.9999991$)	$\lambda_M \leq 1 \times 10^{-5}$	2.654×10^{-6}	0.781	0.367
Random Forest ($t = 0.99998, \Theta = 0.999998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	0.867	0.407
Decision Tree PR ($t = 0.99992, \Theta = 0.999992$)	$\lambda_M \leq 1 \times 10^{-5}$	6.295×10^{-6}	0.941	0.442
FedDyn ($t = 0.9997, \Theta = 0.99997$)	$\lambda_M \leq 1 \times 10^{-5}$	9.627×10^{-6}	0.986	0.463
Random Forest ROC ($t = 0.99999, \Theta = 0.999999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.839×10^{-6}	1.052	0.494
Naive Bayes ($t = 0.9998, \Theta = 0.99998$)	$\lambda_M \leq 1 \times 10^{-5}$	4.505×10^{-6}	1.137	0.534
AdaBoost ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	4.999×10^{-6}	1.162	0.546
Naive Bayes ROC ($t = 0.9998, \Theta = 0.99998$)	$\lambda_M \leq 1 \times 10^{-5}$	9.442×10^{-6}	1.207	0.567
SVM PR ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	3.765×10^{-6}	1.504	0.707
AdaBoost ROC ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	3.271×10^{-6}	2.126	0.999
BDBD ($t = 0.9998, \Theta = 0.99998$)	$\lambda_M \leq 1 \times 10^{-5}$	5.061×10^{-6}	2.128	1.
Logistic Regression PR ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	3.209×10^{-6}	2.406	1.130
Ensemble ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.975×10^{-6}	2.445	1.149
SVM ROC ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.715×10^{-6}	3.106	1.459
Logistic Regression ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.913×10^{-6}	3.295	1.548
SVM ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.098×10^{-6}	3.683	1.731
Logistic Regression ROC ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.543×10^{-6}	3.721	1.749
NN ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.543×10^{-6}	4.910	2.307
LSTM ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.111×10^{-6}	5.730	2.693
Naive Bayes PR ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	1.049×10^{-6}	6.022	2.83
TTF ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-5}$	2.469×10^{-7}	8.515	4.001
Random Forest PR ($t = 0.999996, \Theta = 0.9999996$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	1.457	0.113
Random Forest ROC ($t = 0.999992, \Theta = 0.9999992$)	$\lambda_M \leq 1 \times 10^{-6}$	7.406×10^{-7}	1.571	0.122
Ensemble ($t = 0.99991, \Theta = 0.999991$)	$\lambda_M \leq 1 \times 10^{-6}$	6.789×10^{-7}	3.506	0.272
Random Forest ($t = 0.999992, \Theta = 0.9999992$)	$\lambda_M \leq 1 \times 10^{-6}$	9.257×10^{-7}	3.708	0.287
SVM PR ($t = 0.99993, \Theta = 0.999993$)	$\lambda_M \leq 1 \times 10^{-6}$	9.874×10^{-7}	6.324	0.490
LSTM ($t = 0.99991, \Theta = 0.999991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.335	0.569
Logistic Regression ($t = 0.99992, \Theta = 0.999992$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.342	0.569
Logistic Regression ROC ($t = 0.99992, \Theta = 0.999992$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.342	0.569
FedDyn ($t = 0.99991, \Theta = 0.999991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
Naive Bayes ($t = 0.99991, \Theta = 0.999991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
Naive Bayes ROC ($t = 0.99991, \Theta = 0.999991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
Naive Bayes PR ($t = 0.99991, \Theta = 0.999991$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	7.580	0.588
TTF ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	8.515	0.660
Decision Tree PR ($t = 0.99997, \Theta = 0.999997$)	$\lambda_M \leq 1 \times 10^{-6}$	1.234×10^{-7}	10.696	0.829
AdaBoost ($t = 0.99994, \Theta = 0.999994$)	$\lambda_M \leq 1 \times 10^{-6}$	4.937×10^{-7}	11.729	0.909
BDBD ($t = 0.9999, \Theta = 0.99999$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	12.898	1.
AdaBoost ROC ($t = 0.99993, \Theta = 0.999993$)	$\lambda_M \leq 1 \times 10^{-6}$	5.554×10^{-7}	15.719	1.219
Decision Tree ($t = 0.99995, \Theta = 0.999995$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	17.779	1.378
Decision Tree ROC ($t = 0.99995, \Theta = 0.999995$)	$\lambda_M \leq 1 \times 10^{-6}$	1.234×10^{-7}	17.779	1.378
Logistic Regression PR ($t = 0.99993, \Theta = 0.999993$)	$\lambda_M \leq 1 \times 10^{-6}$	0.000	21.898	1.698
SVM ROC ($t = 0.99992, \Theta = 0.999992$)	$\lambda_M \leq 1 \times 10^{-6}$	2.469×10^{-7}	23.561	1.827
SVM ($t = 0.99992, \Theta = 0.999992$)	$\lambda_M \leq 1 \times 10^{-6}$	1.851×10^{-7}	23.718	1.839
NN ($t = 0.99991, \Theta = 0.999991$)	$\lambda_M \leq 1 \times 10^{-6}$	0.000	24.936	1.933
AdaBoost PR ($t = 0.999991, \Theta = 0.9999991$)	$\lambda_M \leq 1 \times 10^{-6}$	0.000	88.307	6.847

Table A.6: The lowest T_D for each FD in $C_{fallback,higher}$, also normalized by the BDBD solution's T_D, given that a λ_M of at least `level` must be achieved.