Diploma Thesis

MACHINE LEARNING FOR TEST CASE PRIORITIZATION

In satisfaction of the requirements for the degree of

Master of Science

Under the direction of

Univ. Prof i.R. Dipl.-Ing. Dr. techn. Hermann Kaindl
(Institute for Computer Technology)

Dipl.-Ing. Dr. techn. Benjamin Schwendinger
(Institute for Computer Technology)

Submitted at the TU Wien

Faculty of Electrical Engineering and Information Technology

by

Bernhard Slamanig
01229081

Vienna, June 2025

Bernhard Slamanig

TECHNISCHE UNIVERSITÄT WIEN

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In - noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Vienna, June 2025

Bernhard Slamanig

Acknowledgments

I would like to express my sincere gratitude to Professor Kaindl for making this project possible and for providing the essential information and resources necessary for its successful completion. His guidance and expertise were instrumental in shaping the direction of this work. I am also deeply grateful to my supervisor, Benjamin Schwendinger, for his unwavering support and mentorship throughout this journey. His openness to questions, constructive feedback and ability to offer practical solutions to the challenges I have encountered have been invaluable and have contributed significantly to the quality of this project.

I would also like to extend my heartfelt thanks to my mother, whose constant love, encouragement and sacrifices have been fundamental to my academic and personal growth. Her unwavering belief in me has been a source of strength during the more challenging moments of this process.

Finally, I am grateful to my friends and colleagues, especially Ali and Horia, whose constant encouragement, insightful advice and willingness to share their knowledge have played a significant role in my success. Their support, both academically and emotionally, has been indispensable in helping me reach this point, and I truly appreciate their friendship and guidance.

Abstract

In the context of this thesis, a reinforcement learning system is developed to prioritize test cases for regression testing. The system reduces the number of test cases that need to be executed, while maintaining the same level of fault detection. This approach is applicable to any project involving regression testing across multiple cycles.

A key aspect of this work involves comparing various machine learning techniques used to prioritize test cases. The thesis provides an analysis of state-of-the-art machine learning systems that address similar challenges, identifying and selecting the most promising solutions for further evaluation. Four distinct approaches to test case prioritization are explored in this thesis. The first solution is a non-machine learning approach that directly utilizes the failure rate to prioritize test cases. Two additional approaches leverage supervised learning techniques to address the problem. Finally, the fourth solution is based on reinforcement learning. At each regression step, these solutions enable the prioritization of all test cases, with only the highest-ranked ones being executed. In machine learning-based approaches, the outcome of executed test cases is used to refine the prioritization for the next regression step. The proposed solutions are compared with each other, as well as with similar approaches from existing research.

Kurzfassung

Im Rahmen dieser Arbeit wird ein Reinforcement Learning System entwickelt, um Testfälle für Regressionstests zu priorisieren. Das System reduziert die Anzahl der auszuführenden Testfälle bei gleichbleibender Fehlererkennung. Dieser Ansatz ist auf jedes Projekt anwendbar, das Regressionstests über mehrere Zyklen hinweg umfasst.

Ein wichtiger Aspekt dieser Arbeit ist der Vergleich verschiedener maschineller Lernverfahren, die zur Priorisierung von Testfällen eingesetzt werden. Die Arbeit bietet eine Analyse der modernsten maschinellen Lernsysteme, die ähnliche Herausforderungen angehen und identifiziert und wählt die vielversprechendsten Lösungen für eine weitere Evaluierung aus. Vier verschiedene Ansätze zur Priorisierung von Testfällen werden in dieser Arbeit untersucht. Bei der ersten Lösung handelt es sich um einen Ansatz ohne maschinelles Lernen, der direkt die Fehlerrate zur Priorisierung von Testfällen nutzt. Zwei weitere Ansätze nutzen supervised learning, um das Problem zu lösen. Die vierte Lösung schließlich basiert auf Reinforcement Learning. Bei jedem Regressionsschritt ermöglichen diese Lösungen die Priorisierung aller Testfälle, wobei nur die am höchsten eingestuften Fälle ausgeführt werden. Bei Ansätzen, die auf maschinellem Lernen basieren, wird das Ergebnis der ausgeführten Testfälle verwendet, um die Priorisierung für den nächsten Regressionsschritt zu verfeinern. Die vorgeschlagenen Lösungen werden miteinander sowie mit ähnlichen Ansätzen aus der bestehenden Forschung verglichen.

Table of Content

1 Introduction

The following chapter provides a brief overview and understanding of the framework, conditions and aims of this thesis, as well as outlining the structure of the thesis. Subsequent chapters will address the problem statement in detail, followed by a comprehensive introduction to the relevant concepts and background information. In addition, the methodological approach of this research will be explained together with the rationale behind the choice of methods and their application to the problem at hand.

1.1 Motivation

Regression testing is the testing of the parts of the system that have already been tested, when the system changes. The purpose of regression testing is to identify potential changes that might have had an unexpected impact on another part of the system. In large projects, with increasing complexity and frequent cycles, this often leads to a great number of test cases all of which would require a lot of time [1], computing resources and sometimes manual effort to run. As a result, there is considerable economic and scientific interest in reducing the number of test cases that need to be executed during regression testing, which could lead to improved efficiency and lower costs.

Therefore, this thesis addresses this challenge by using and comparing various machine learning techniques used to prioritize test cases. The techniques should ensure that only the highest ranked test cases, those most likely to detect defects, are performed by giving a rank to each test case based on learned patterns. The different machine learning, have one goal in common and that is to reduce the size of the regression suite, while also maintaining a high fail rate detection. This approach ensues high probability of discovering defects in the system under tests, even if new code changes are pushed into the code base.

Furthermore, the results of this thesis could be relevant to researchers working in software quality and test automation.

1.2 General Introduction

The concept of artificial intelligence (AI) is not new. For the past decades, the idea of endowing machines with intelligent behaviour has fascinated numerous researchers and scientists. A significant milestone in this development was reached in 1950 when Alan Turing introduced the famous "Turing Test" proposed to determine whether a machine can be considered intelligent [2]. During this test, the machine engages in a conversation with a human and attempts to behave in a way that mimics human behaviour. The goal was to convince the test subjects that they are interacting with another person. If the machine successfully creates this illusion, the Turing Test is considered passed [3]. Although originally conceived as a theoretical thought experiment, the test continues to hold significant relevance in contemporary research and is frequently referenced in discussions surrounding AI.

No machine has yet successfully passed the Turing Test. Nevertheless, AI is proving extremely useful in many aspects of our daily lives. It is used in areas such as object recognition,

personalised advertising, email spam filtering, online stock trading and traffic forecasting. It also plays a crucial role in robotics and the development of self-driving cars [1][2]. More recently, there have been significant advances in AI capabilities with the advent of ChatGPT, a language model from OpenAI. ChatGPT is capable of human-like conversations and is a demonstration of the potential of natural language processing [4]. This development demonstrates how AI can simulate human communication to a degree that blurs the lines between machine and human interaction, further stimulating discussions about the relevance of the Turing Test in modern AI research.

As a subfield of AI, machine learning (ML) allows computers to throw conclusions from data. It is used, for example, in the prediction of house prices or cancer detection. Also, ML is very effective in test case classification [5]. In this way, the failure probability of test cases can be estimated. The probability of failure can then be used for prioritization, which is what is done in this thesis.

1.3 Problem and Aim of the Work

In practice, it is a common challenge that test cases take a lot of time and effort to run, as every single test has to be performed in every test cycle. This often leads to inefficiencies as redundant testing can be time and resource consuming. Hence the goal of this work is to develop a ML component, which should reduce the testing effort in systems that use regression testing. The component should use the test results to form an evaluation function that allows test cases to be prioritized so that only high-priority test cases need to be run [6].The component´s results should be evaluated and presented graphically. The results should then be compared with other test case prioritization components to assess the adequacy of the new component. It is not the aim of this thesis to create a component that runs tests, although this extension is theoretically possible. Instead, the component should use datasets of tests already run and their output for evaluation. Another limitation is that the component must not independently create test cases.

1.4 Solution Approach and Work Packages

The first step of this thesis consists of a literature review to gain a deeper understanding of the key issues. A background section was also created to describe and compare the different approaches of existing studies. Following the literature review, four very promising approaches are implemented. The baseline approach works with failure rates and does not involve ML. Two approaches use supervised learning techniques. One uses Naïve Bayes classification. The other is a random forest algorithm. The last approach uses reinforcement learning. Various tests and measurements are carried out for all approaches. The approaches are compared with each other.

1.5 Structure of the Work

Chapter 2 presents the background section of this thesis, providing a basic overview of key concepts related to testing and ML. This chapter serves as a basis for understanding the

technical and theoretical aspects relevant to the research and gives context to the subsequent analysis and implementation.

Chapter 3 focuses on a comprehensive review and analysis of existing work in the field. Different methods and approaches are examined, systematically described and compared to assess their strengths and weaknesses. The analysis identifies the most promising solutions for this project, ensuring that the selected approaches are consistent with the research objectives and requirements.

Chapter 4 delves deeper into the methodologies used in this study. It provides a detailed explanation of the selected methodologies, including the rationale for their selection and their application in the specific context of this thesis. The aim of this chapter is to provide transparency regarding the implementation process and to justify the choices made during the research.

Chapter 5 deals with the measurement system and data sets used in this thesis. It outlines the processes involved in data collection and analysis and provides insight into the evaluation framework used to assess the effectiveness of the proposed solutions. In addition, the chapter presents a comparative analysis of the results obtained from the different solutions, highlighting their relative performance and applicability.

Chapter 6 provides a summary of the main findings and conclusions of the study. It reflects on the results obtained and discusses their implications for the field. The chapter also explores potential directions for future research, suggesting further projects that could build on the findings of this thesis.

2 Background Section

2.1 Testing and Analysis

Testing is the process of systematically assessing and validating that the software behaves as intended, ensuring that it meets its specified requirements and that it performs its functions correctly [7]. Software testing is most effective when it is part of continuous integration, i.e. when testing is started at the design stage and is continued throughout the development process of the software. Testing is an important part of developing software, because it ensures that all components function as intended. On the other hand, analysis involves the process of defining test cases by determining what needs to be tested and specifying the methods for doing so. Testing and analysis are two distinct processes, yet they are closely interconnected and complement each other. However, both involve understanding the software specifications, defining test objectives and identifying potential risks. When combined, testing and analysis ensure comprehensive coverage and effective faults detection. Ultimately, this contributes to higher software quality and reliability. The following subchapter provides an overview of regression testing, a key subject of this thesis.

2.1.1 Regression Testing

Some systems are incremental, where the built system must be tested after each step, this is called regression testing. Regression testing ensures that the changes that have been made do not add new errors [5][8][9].

However, retesting the whole system would be very cost, resource, and time-consuming [8] [10][11][12] that is why are various techniques used to minimise test effort, such as test case selection, where a carefully selected subset of test cases is executed and test case prioritization, where all test cases are ranked and executed in a predefined order [2][8][10][11] . This process continues iteratively, optimising the testing procedure until either the allocated runtime is exhausted or a predefined coverage threshold is met [9][11][12].

2.2 Machine Learning

ML focuses on enabling computers and machines to mimic the way humans learn. ML teaches a computer how to perform a task without explicitly programming it to do it. Instead, data is fed into an algorithm to gradually improve the result with experience. The term was coined in 1959 by Arthur Samuel at IBM, who was developing various fields within AI. Although ML has a wide range of applications today, it performs two main tasks, one is to classify data and the other is to make predictions about future outcomes.

There are six main steps, each of which performs a specific task, in the conventional ML approach Figure 1 [2].

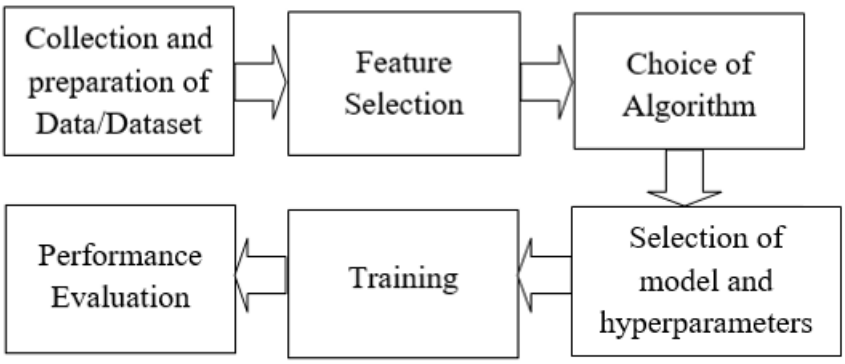

Figure 1: Traditional Machine Learning Model (Copy from [2])

Collection and Preparation of Data: It is well known that this data preparation phase can greatly improve or deteriorate the ML outcome. It is important to use collected data sets from reliable sources. It should be meaningful and there should be a sufficient amount of data. Data often needs to be cleaned and pre-processed before it can be used as input to the ML algorithm [2]. The preparation phase generally includes three steps: Data cleaning, i.e., handling missing data and outliers, data reduction, i.e., reducing the data size by aggregation, elimination redundant feature, etc. and data normalization [13].

Feature Selection: The next step is the choice of meaningful features. Features must have a relevant influence on the output variable.

Choice of Algorithm: There are several different algorithms in ML. Each algorithm makes different assumptions. Depending on its assumptions, an algorithm may be suitable as a solution to a problem. Some examples of algorithms are Naïve Bayes, Decision Tree and Support Vector Machines.

Selection of model and Hyperparameters: Most ML algorithms need hyperparameters to be set before training [2].

Training: The input data is divided into a training set, validation set and a test set. This split is implemented because the training data is used to train the model, and a portion of the data is held back to assess the model's performance on previously unseen data.

The training set should contain about 70% of the total data [1]. The training set is used to give the model examples of which input features lead to which output. The ML algorithm tries to model this relationship as a function.

The validation dataset, which represents approximately 15% of the total data, is used to refine the model and obtain performance metrics. The results obtained from the evaluation of the model on the validation dataset allow further refinement of the model.

The test dataset should contain approximately 15% of the data. It is used to test the actual data output. Testing requires the use of data not known to the algorithm [1]. The test data is used to predict the results of the inputs and compare them with the actual output. This can be used later for performance evaluation. The percentages quoted in this work are unique and may depend solely on the data and balances.

Performance Evaluation: Test and validation data is used to evaluate the ML algorithm. Important metrics for classification are accuracy, precision and recall [2]. The metrics are described in more detail in Section 2.2.5.

However, there are big changes happening in the field of ML during the last few years. This is mainly due to introduction of deep learning and large computing power. Therefore, newer ML approaches drift away from the traditional ones, they concentrate more on areas like deep neural networks [14].

2.2.1 Machine Learning Models

Machine learning can be divided into models based on how an algorithm is trained and the availability of output during training [1]. This thesis describes some of the most important ML models, which are:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

2.2.2 Supervised Learning

Supervised learning is ML with data with given output (label). The algorithm learns key characteristics from examples. After that, the algorithm can predict the output of the test data [1][2][15]. The success of supervised learning depends heavily on the quality and representativeness of the labelled training data, and the ability of the model to generalise to unseen data. It is widely used in areas such as image recognition, speech processing and predictive analytics. There are several models of supervised learning, which are introduced in the following chapters.

2.2.2.1 Classification

Machine learning algorithms can be used for various problems. A well-known problem is the classification problem, in which the algorithm attempts to assign a set of input features to a given number of output classes [1][2] . As in object recognition, where objects are classified into categories such as “cars”, “boats” or “animals”, classification is used to determine specific outcomes. For example, it can predict whether a test is likely to result in a “pass” or a “fail”, so these categories are used as predefined classes.

2.2.2.2 Regression

Regression is another supervised ML task. Regression models are trained to understand the relationship between different independent variables and an outcome. The goal of this learning task is to estimate unknown dependencies from training data with good predictive ability for future data. Based on the number of predictor variables and the nature of the relationship between the variables, regression can be classified into several types such as logistic regression

(Section 2.2.2.3), linear regression (simple, multivariate, and multiple), polynomial regression, non binary regression, ridge and lasso regression, and similar.

2.2.2.3 Logistic Regression

Logistic regression is a great solution for ML problems involving binary classification, where the probability of each input being in one of two different categories is determined. The algorithm models the relationship between the input features and a binary target variable using the logistic (sigmoid) function, which maps predicted values to probabilities between 0 and 1. A threshold, often set at 0.5, is then used to classify observations into one of two classes**.** It has become a preferred choice in fields as diverse as medical diagnosis, credit scoring, and marketing because of its simplicity, interpretability, and solid mathematical foundation [16]. This model is explicitly described here because it is applicable to our use case, which has two classes, failing test case and not failing test case. It is also used in the paper by Lachmann et al. (Section 3.2.2).

2.2.2.4 K-Nearest-Neighbours

Another supervised learning model is k-nearest-neighbours (KNN). The algorithm calculates the distance between the feature point and its neighbours, typically using metrics such as Euclidean distance. The class with the highest number of votes among these neighbours is then assigned as the new class of the feature point [10]. An interesting case happens when the two major results of a vote result in a tie between two classes. In this case the result needs to be determined by applying some additional strategy. For example, choosing k as an odd number resolves this issue. Another strategy is random tie breaking where the determined class is chosen randomly between the winning classes.

2.2.2.5 Support Vector Machine

Support Vector Machine creates an N-dimensional feature vector space and classifies the given feature points in such a way that the point-free area between the class boundaries is maximised [10]. The goal of SVM is to distinguish data points in an N-dimensional feature space and classify them according to this distinction. Herby a hyperplane is used as boundary between classes. That’s why they are also called decision boundary. The hyperplane can have different dimension depending on the dimension of the feature space. For example, in a 2-dimension feature space the hyperplane is a line.

A specific case is where there are points in the so-called point-free region. These points are then classified according to which side of the hyperplane they are on [17].

2.2.2.6 Artificial Neural Networks

Artificial neural networks (ANNs) are inspired by how the neurons work. An ANN is made up of interconnected nodes called artificial neurons. Each artificial neuron has several inputs and an output. Depending on the inputs, the output can be activated. The artificial neurons can be layered so that the output of one artificial neuron is the input of another. The weights of the connections in the neural network are adjusted during training. Because of these weights, the sensitivity of the outputs to certain inputs can be set [1].

ANNs possess powerful capabilities for handling complex data. They are well suited for identifying underlying patterns. But they often need high computational power, another downside is the lack of transparency in decision making. Important for ANNs is the universal approximation theorem, which states that any continuous function can be approximated arbitrarily well by a neural network with at least one hidden layer with a finite number of weights [17].

2.2.2.7 Naïve Bayes

Naïve Bayes is a simple learning algorithm that uses Bayes's rule combined with a strong assumption that the attributes are conditionally independent of each other. Naïve Bayes often achieves competitive classification accuracy even though this independence assumption is often violated in practice. This, combined with its computational efficiency and many other desirable features, leads to the widespread use of Naïve Bayes in practice [18].

2.2.2.8 Random Forest

Decision trees (DTs) are a class of simple predictors. These predictors essentially represent a sequence of conditional steps that must be taken to arrive at a decision. Their popularity is largely due to their efficiency. However, there are some drawbacks when it comes to DTs. The main one is overfitting, i.e. the model performs well on one dataset but generalizes poorly to others. To make generalized predictions, Random Forest is used, which is based on collective intelligence. As the name suggests, a random forest is a tree-based ensemble where each tree depends on a collection of random variables. It performs well on a wide range of data sets and is a flexible algorithm with a wide range of applications.

2.2.3 Unsupervised Learning

Unsupervised learning is a type of ML that uses training data but does not use the correct label. With this input data, the algorithm can find similarities, while the algorithm identifies clusters or groups with similar characteristics [2][1][10]. This approach is often used for discovering hidden patterns in data, such as customer segmentation or anomaly detection.

2.2.4 Reinforcement Learning

In reinforcement learning (RL), an agent that uses the interaction of trial and error and tries to maximise the rewards [2][19][20]. The rewards and penalties are given by a reward function. RL is often used in changing or partially unknown environments where strategic decisions are required, for example in games [19][20].

RL also uses an environment with different states. The states depend on the current properties of the environment. Depending on the state of the environment, the agent performs an action from a predefined set of actions. This results in a transition to a new state [1][21].
The result of the action is then rewarded with a reward function. The agent tries to perform the actions that lead to the highest rewards [15][19]. This interaction is shown in Figure 2.

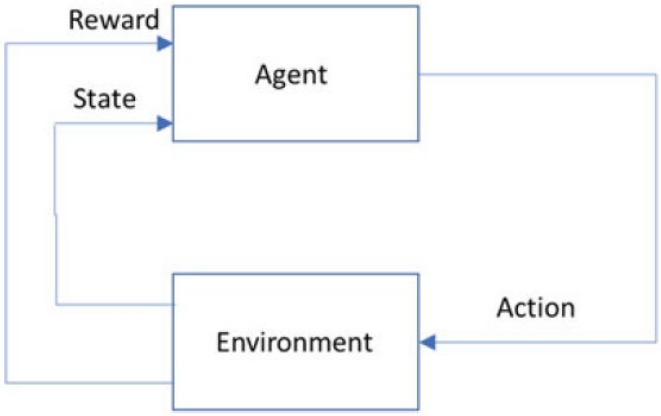

Figure 2:Interaction between Agent and Environment (copy from [1])

2.2.4.1 Model-Based and Model-Free Reinforcement Learning

Model-based reinforcement learning uses a Markov Decision Process model. A reinforcement learning algorithm is called *model-free* if it operates within the Markov Decision Process (MDP) framework but does not use or learn the transition probabilities or reward function explicitly. Instead, it learns optimal behavior directly from interaction with the environment to learn a policy of value function. [1] An example of a model-free algorithm is Q-learning.

2.2.4.2 Q-Learning Algorithm

The Q-learning algorithm is a fundamental method in reinforcement learning for estimating rewards. The Q-value represents the expected future reward for taking a particular action in a given state. After each step, the Q-value is updated according to a predefined update rule, allowing the algorithm to iteratively improve its policy and optimise decisions over time.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\big(r_t(s_t, a_t) + \gamma\, max_a\, Q_t(s_{t+1}, a) \; - \; Q_t(s_t, a_t)\big)$$

with

$Q_t(s_t, a_t)$ … old Q − value

$Q_{t+1}(s_t, a_t)$ … new Q − value

α … learning rate between 0 and 1

$r_t(s_t, a_t)$ … reward received for transition between state s_t and s_{t+1}

γ … discount factor between 0 and 1

$max_a\, Q_t(s_{t+1}, a)$ … estimate of optimal future value

The learning rate α is a tuning parameter for the learning process. A high learning rate leads to big changes in the Q-values, while with a smaller learning rate only small changes are made. Typically, algorithms start with a high learning rate which gets continuously reduced in later steps [20].

Another tuning parameter is the discount factor γ, which causes rewards received in earlier steps to be valued lower than rewards received closer to the current step.

Depending on a probability value ε, the agent decides between two activities, exploration and exploitation. During exploration, the agent chooses the action at random. During exploitation, the agent performs the action with the highest Q-value [15]. This helps the agent to explore the input state and prevents the agent from getting stuck in a poor strategy [19], ε is reduced in later steps. With the formula:

$$\varepsilon_{t+1} = \varepsilon_t \alpha$$

2.2.5 Performance Evaluation for Classifiers

Different metrics are used to evaluate performance. There are four possible outcomes for ML systems designed for binary classification tasks. These outcomes are illustrated in the confusion matrix shown in Table 1.

	The Element is predicted to belong to the class	The Element is predicted not to belong to the class
The element belongs to the class	**True positive**	**False negative**
The element does not belong to the class	**False positive**	**True negative**

Table 1: Confusion Matrix

From these outcomes, different metrics can be calculated.

2.2.5.1 Accuracy

Gives the ratio between correct predictions and the number of total predictions [11][21].

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{total predictions}} \quad [11]$$

or

$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{true positives} + \text{true negatives} + \text{false positives} + \text{false negatives}} \quad [11]$$

2.2.5.2 Precision

Gives the ratio between elements that are predicted to belong to a class and do with respect to all elements that are predicted to belong to the same class [11][21].

$$\text{precision} = \frac{\text{true positives}}{\text{false positives} + \text{true positives}} \quad [11]$$

2.2.5.3 Recall

Gives the ratio of elements that are predicted to belong to a class and all elements that belong to the same class [11][21].

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \quad [11]$$

In cases where the classes are not evenly distributed, it is better to use precision and recall for evaluation. In these cases, the precision can be high because most elements are predicted to belong to the larger class. If the recall of the smaller class is low, the prediction is not very useful [21].

For example if a classifier is designed to label all test cases as "pass" when distinguishing between "fail" and "pass," the accuracy may appear high, as the majority of test cases typically fall into the "pass" category. However, the recall for the "fail" class would be low, and the precision for the "pass" class would also be poor. Consequently, using recall and precision is often a more suitable approach for evaluating classifiers when class distributions are imbalanced.

The ML algorithms are each trained with a training dataset which consists of all tests which are executed till the end of the current cycle. The evaluation is always performed on the same test-set which size is 15% of the whole dataset and contains the tests and results of the last cycles.

The recall is calculated and displayed graphically. To compensate for the limitations of the recall metric, in the graphs of this thesis another curve is added to the graphs which represents the percentage of test cases classified positive (failing).

2.2.6 Performance Evaluation Metrics

Test case prioritization techniques that produce a continuous range of values as output, rather than discrete classes, require alternative metrics for evaluation. Such techniques typically assign a score to each test case which indicates the relative importance of the test case or the likelihood of a defect being detected.

2.2.6.1 Average Percentage of Faults Detected (APFD)

For determining the APFD value the ranks of failure detecting test cases in the test execution order are used [6] [9] [10] The following formula is used for calculating APFD:

$$APFD(T') = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{n \cdot m} + \frac{1}{2 \cdot n}$$ [9][19][20]

with

T' ... prioritized test suite

T ... Test suite containing n test cases

F ... set of m faults revealed by T

m ... number of faults contained in the system under test

n ... total number of test cases

TF_i ... the position of the first test in T that exposes fault i

APFD can be between 0 and 1. A high APFD value indicates that the prioritization of test cases is effective and that many defects would be detected even if only the high-priority test cases were executed [5][9][10].

3 Literature Analysis

This chapter will be a review of the current methodology for test case prioritization. Approaches are compared with one another according to their strengths and weaknesses. There are many different ML solutions for test case prioritization. This thesis describes and compares the following characteristics are compared:

- Used ML technique
- Features used for prioritization
- Used performance evaluation metrics

To improve software safety, it is essential to perform regression testing on every system update. In a continuous integration environment, regression testing requires test cases that can provide rapid feedback. As a result, it is critical to effectively prioritize test cases within a specific time frame to maximize defect detection and increase the defect detection rate of the testing process [22]. Regression testing is essential for verifying that changes to code do not alter its intended functionality. However, executing all test cases can be time-consuming and resource-intensive, especially with the increasing use of Agile development in web applications, which results in frequent software builds. To address this challenge, test case selection and prioritization (TCP) strategies have been developed to optimize the testing process by selecting and ordering test cases in a way that provides timely feedback to developers. Recently, researchers have increasingly turned to ML techniques to develop more effective ML-based TCP approaches. In regression testing the system changes in each regression step, which might make adaptive solutions like reinforcement learning more suitable for this problem [6][12].Therefore, this literature analysis focuses on test case prioritization solutions with reinforcement learning.

Finally, some approaches for test case prioritization with ML will be analyzed to get a good overview of currently used techniques and their performance. The interested reader may be pointed toward [5] for further information.

3.1 Solutions with Reinforcement Learning

3.1.1 RETECS

“Reinforcement learning, clustering, ranking, and models based on natural language processing are the key ML approaches used for TCP” [23]. Spieker et al. [6] describe the reinforced test case selection (RETECS) method. This method uses Reinforcement learning (online learning) for test case prioritization and selection. Online learning is a ML technique that allows constant learning during runtime. This makes RETECS an adaptive method. Adaptive means that the method adapts to the environment, which is changing during the regression steps.

For prioritization, the RETECS method uses the features duration, previous last execution and the failure history of test cases. The number of executed test cases is constrained by time, so the total duration of execution must stay under a predefined threshold. So only the higher ranked test cases are executed and only these results are used in the learning process. To determine the reward the paper compares three different reward functions. The Failure Count Reward tries to

maximize the number of failed tests of the whole selection, while the Test Case Failure reward determines the reward considering the individual test cases. The third reward function (Time-ranked Reward) takes the previous order of the test cases into account. However, the Test Case Failure Reward seems to work best.

For Evaluation, RETECS uses the Normalized Average Percentage of Faults Detected (NAPFD), to make their system, which only orders test cases till the time constraint threshold comparable to other systems that use APFD for evaluation. In most cases, the NAPFD of their experiments is something about 0,4, which indicates that the RETECS method detects approximately 40% of the defects early in the test execution, suggesting a moderate level of fault detection within the given time constraints.

3.1.2 Extended Diagraphs

According to Emam et al. [20], a model-based testing technique is presented. The technique uses Reinforcement learning together with a hidden-Markov model (HMM).

For graphical user interface (GUI) testing Model-based techniques deliver good results [24]. But they can also be used for other applications where HMMs could be built. For the paper Auto-Black-Test [15] a tool for the automatic generation of GUI test cases is used to generate the test cases.

The technique prioritizes tests based upon the number of computations (changes) that a test case may cause in the system under test. Test cases with a higher number of changes are more likely to lead to system failures. The used technique is Q-learning. The test cases are ordered by using the Q-values in descending order.

For the performance Evaluation, the APFD and other metrics are used. We will concentrate on the APFD because it is easier to compare. The Mean of APFDs values is for RL-based HMM between 0.6865 and 0.9339, depending on the tested application.

3.1.3 Ranking to Learn

An alternative approach described in the paper by Bertolino et al. [25] that is well suited for dynamic contexts is RL. RL algorithms applied to ranking are referred to as ranking to learn (RTL) as opposed to learning to rank (LTR) (described in 3.2.4), because RTL uses ranking information at each step to refine the model's predictions. According to Zurek-Mortka et al. [23], RTL offers notable advantages for test prioritization in CI environments because it can naturally adapt to changes in the test suite - such as the addition or removal of tests in each CI cycle - and to adjustments in the CI process itself [25]. The main difference between LTR and RTL is how they learn and how they deal with situations that are always changing.

3.1.4 Regression Testing based on Q-Learning with Autosys

Q-learning is a reinforcement learning algorithm that has been integrated into regression testing for test case prioritization. The objective of this approach is to speed up the deployment process by effectively prioritizing test cases, while ensuring bug-free software updates through continuous test case prioritization and full automation. By continuously learning from system

feedback, Q-learning assigns priority levels (high, medium, and low) to test cases based on various factors, helping to optimize the order of test execution. In this approach, the Q-learning algorithm iteratively adjusts its actions using action values of different states, improving the agent's performance over time. As a model-free technique, Q-learning does not require a predefined model of the environment and adapts on-the-fly based on observed transitions and rewards [26]. At each transition, the learning agent takes an action, receives a reward, and moves to the next state until it reaches a final state, completing the process.

Once the test cases have been prioritized in each cycle, the test execution process is often automated through the use of AI integrated with ML-driven test case scheduling. Autosys is a cross-platform job management system that handles the scheduling, monitoring, and reporting of various tasks called Autosys jobs. It can autonomously perform a wide range of tasks, including the execution of multiple test cases, without the need for human intervention. It also provides real-time feedback on the results of the tests. An overview of the Autosys workflow is shown in Figure 3[26].

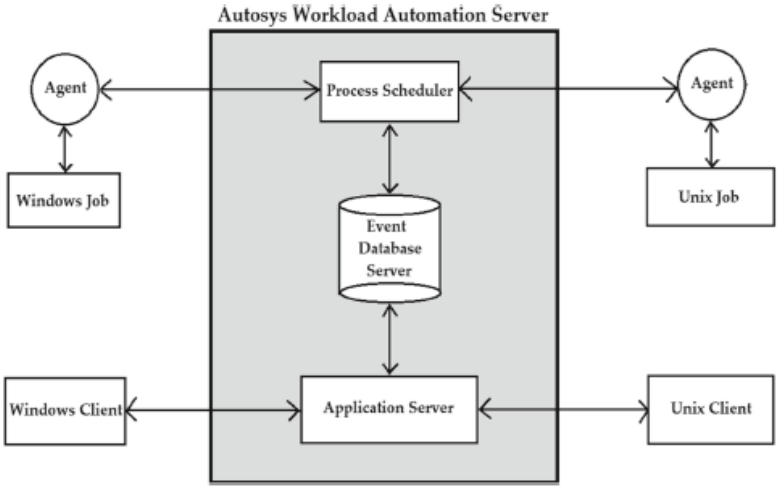

Figure 3: Approach Architecture as Discussed (Copy from [26])

3.2 Other Solutions

3.2.1 Learning Software Agents

Abele et al. [8] used learning software agents for the test case fault-proneness prediction. The system uses fuzzy logic rules. To create Fuzzy logic rules, rules that apply to testing are formulated by experts. These rules are translated into many-valued truth values, which can then be interpreted by the machine. In this way, the fuzzy logic rules represent expert knowledge on testing like: “Complex modules are more fault-prone than simple ones”.

The fuzzy logic rules are used together with a genetic algorithm to predict the fault-proneness. The genetic algorithm optimizes the fuzzy logic, by adapting the weight factors of single fuzzy logic rules. Features used for this optimization are the number of past faults, the number of

recent changes, the criticality of the system, the complexity of the system and the number of found faults in the previous test.

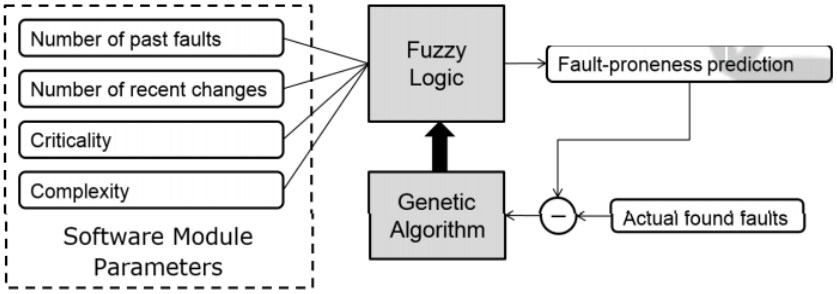

Figure 4: Fault-Proneness Prediction with Fuzzy Logic Rules and Genetic Algorithm (Copy from [10])

Genetic algorithms are inspired by biological organisms. They adapt to their input and try to fit their environment. This model is called evolutionary learning. For the evaluation of this approach, the fault-proneness prediction of the system was compared to a classic prediction made by a developer. Unfortunately, there were no evaluation metrics used.

3.2.2 Machine Learning Approaches for Black Box Software Testing

In his paper [10] Lachmann et al. compares a variety of ML approaches to black box testing of software. In testing, access to the source code is not always available, so black-box approaches are necessary. Interestingly, the help of a test expert is used as part of this ML system. The test expert selects a set of positive and a set of negative test cases (i.e. a set of important and a set of less important test cases) from a test case database. These data sets are later used for the ML algorithm. The concept is illustrated in Figure 5.

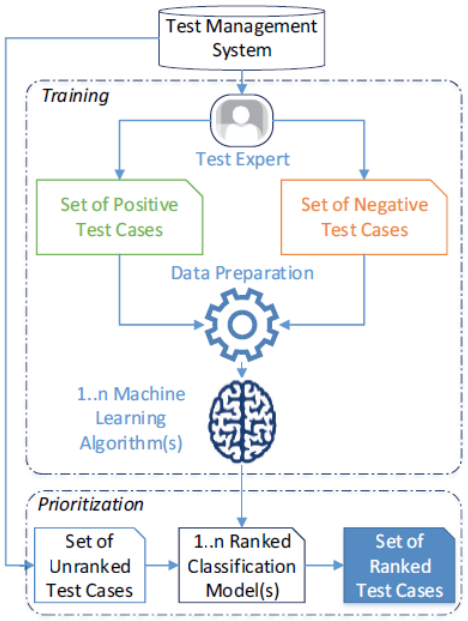

Figure 5: Concept of Remo Lachmann's Approach (Copy from [10])

For ML, the following four algorithms are applied in the paper:

1. Ranked support vector machines
2. K-Nearest- Neighbour
3. Logistic regression
4. Artificial neural network

The results of these techniques are also combined and it is called ensemble learning. Features used for test case prioritization in the paper are title, number of linked requirements, execution duration and the test case description.

For Performance evaluation, the APFD performance metric was used. Results are shown in Table 2. Boost stands for the ensemble learning technique.

System	Average APFD value with description				
	SVM	*KNN*	*Log Reg*	*Neural*	*Boost.*
A	0.68	0.69	**0.75**	0.7	0.72
B	0.64	0.52	**0.66**	0.56	0.57
C	0.68	0.44	0.69	0.63	**0.7**
Avg.	0.67	0.55	**0.7**	0.63	0.66

Table 2:APFD Values of the different Techniques used by Remo Lachmann (Copy from [10])

3.2.3 Ranking SVM

In the paper [12], a supervised ML approach is used for test case prioritization. The used ML technique is Ranking SVM.

This model was introduced by Thorsten Joachims who is a well-known researcher in ML and information retrieval. Ranking SVM is a pair-based ranking approach used primarily in applications such as search engines and recommendation systems. It learns preference relationships between pairs of data points rather than classifying individual data points [27].

The input features are test case description, requirements coverage, failure count, failure age, failure priority and execution costs.

For evaluation, the APFD value was used. Two datasets have been evaluated. The results are APFD=0.92 for the first dataset and APFD=0.81 for the second dataset

3.2.4 Learning to Rank

A recent learning strategy in ML is LTR, which primarily involves supervised algorithms. LTR has proven valuable in areas such as information retrieval and natural language processing. In software engineering, it has been effectively applied to tasks such as defect prediction, where modules are ranked according to their likelihood of containing defects [25]. Similarly, in test prioritization, LTR can be used to rank test targets (e.g., test cases or test classes) based on supervised learning problem, LTR requires prior training. However, if the operational context changes from the training environment, the model may no longer be representative and could lose its predictive accuracy. This is particularly relevant in CI scenarios where such discrepancies can occur.

LTR is used to generate ranking functions from training data sets. These ranking functions are then used to order the documents retrieved in response to a user query. Figure 6 illustrates the typical architecture of LTR approaches that most learning-based methods follow to address the IR ranking challenge [16].

In paper [25], the Rank Percentile Average (RPA) is defined as a performance evolution designed to adapt the RPA to the prioritization problem by calculating how closely a predicted ranking matches the actual ranking. This metric can evaluate a ranking independently of the specific test criteria (e.g., error detection).

Whereas in paper [16] Mean Average Precision and Normalized Discounted Cumulative Gain were used as performance evaluation metrics.

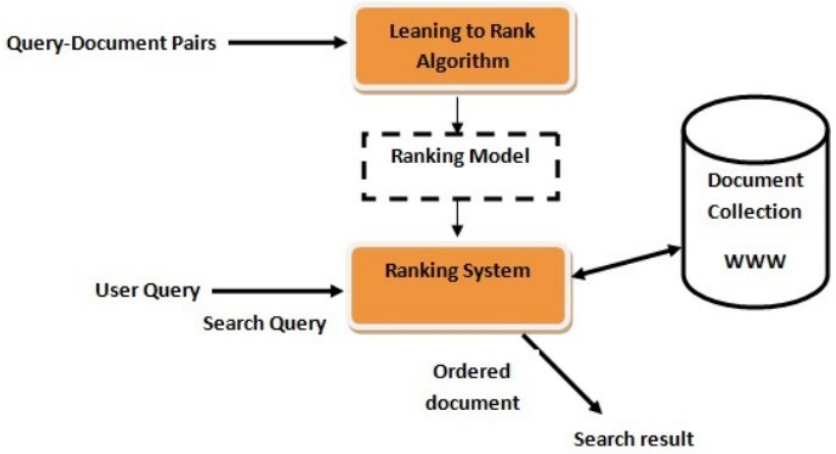

Figure 6: Approach Architecture as Discussed (Copy from [16])

3.3 Conclusion of the Literature Analysis

In the literature analysis, we learned that there are many different ML models are used for test case prioritization. Like the ML-model, the ML-techniques used in each approach vary. It is currently unclear which model and which techniques are best suited for test case prioritization.

As much as the ML models vary, the features used are similar in many approaches. Commonly used features are

- Failure history
- Amount of changes
- Duration or execution costs
- Test case description

For the performance metrics, the APFD value was used in most of the techniques.

The results of the whole literature analysis are summarized in Table 3. The table gives a good overview of the different ML approaches.

Name	Machine learning technique	Features used for prioritization
RETECS [7]	Online learning	duration, previous last execution and the failure history of test cases
Extended Diagraphs[19]	RL	amount of computations (changes)
Learning Software Agents [8]	Ranking SVM	number of past faults, number of recent changes, critically of the system, complexity of the system, number of found faults in the previous test
Different ML approaches for Black Box Software Testing [10]	SVM, KNN, Log Reg	title, number of linked requirements, execution duration, test case description
Different ML approaches for Black Box Software Testing [10]	Layered Neural Network	title, number of linked requirements, execution duration, test case description
Ranking SVM [12]	Ranking SVM	test case description, requirements coverage, failure count, failure age, failure priority and execution costs
Learning to Rank [25]	KNN,RF,L-MART,…	duration, previous last execution and the failure history of test cases
Ranking to Learn [25]	RL	duration, previous last execution and the failure history of test cases
Regression Testing based on Q-Learning [26]	RL	-

Table 3: Machine Learning Approaches for Test Case Prioritization

The conclusions of the Literature analysis are used in the following Sections to select meaningful features and algorithms.

4 Experiments

This chapter gives an overview of the concept and explains which procedures were used to implement the project. It explains the chosen algorithms and hyperparameters. It is also described how the system could be applied in a useful way. Additionally, the approach to evaluating and validating the model is discussed to ensure traceability and reproducibility of the results.

4.1 Concept

Different ML models are used to determine the priorities of test cases. Four different datasets are used in this thesis.

From those datasets, a selected set of features is used to determine the priorities of the test cases in the dataset. Priorities are determined in each regression step and then various evaluation metrics are applied to evaluate the quality of the used models.

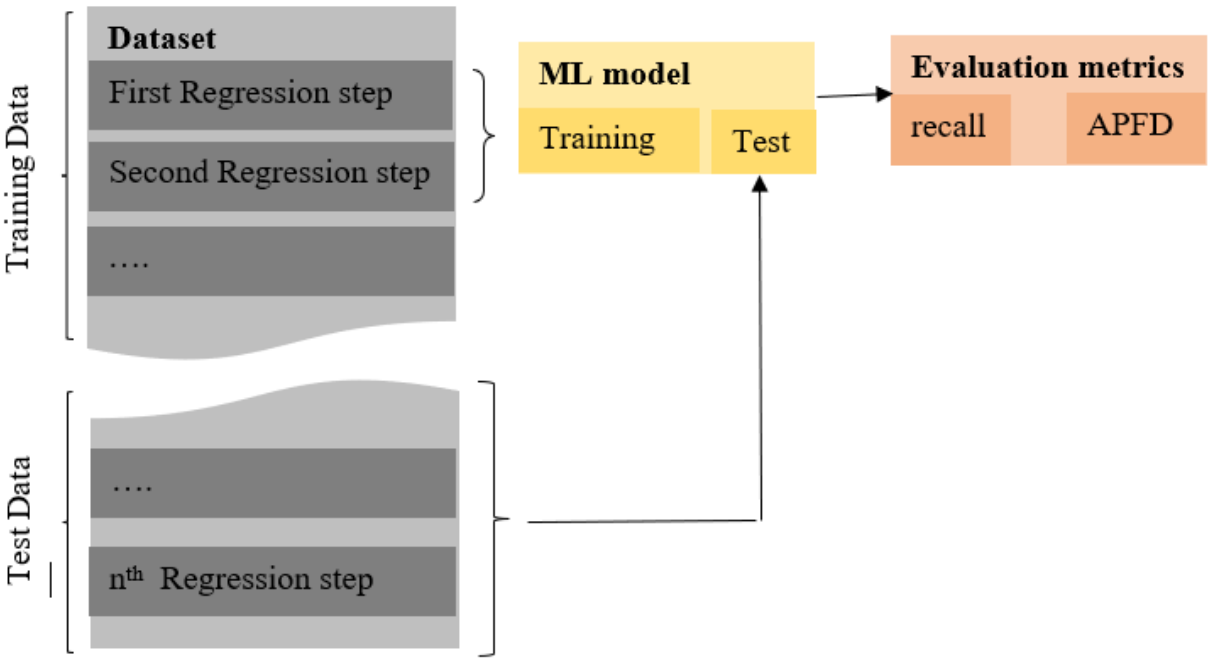

Figure 7: Example of the Concept

Figure 7 shows an example of how the concept is applied to the second regression step of a dataset.

The first phase is to split the dataset into a training and a test dataset.

Second, the data collected during the previous regression steps (one and two) is used to train the machine learning model. In the case of the failure rate model, no training takes place, the collected data is used here to calculate the failure rates.

The third phase is performed on the data from the test set and is used to test the trained ML model. The ML model outputs the different scoring metrics. The scoring metrics can then be compared between our different models and with other models from the literature. This process

is repeated for every regression step which gives full insight into how each regression step affects the metrics.

4.2 Implementation

The ML algorithms used in this project were implemented in Python using the Scikit-Learn Reference Library [29]. Scikit-learn is a widely used open-source library for Python that provides a comprehensive set of tools for machine learning tasks. It includes algorithms for classification, regression, clustering, dimensionality reduction and model selection. The library is well documented and widely used in both academic and industrial settings due to its ease of use and efficient implementation of machine learning techniques. Additionally, scikit-learn offers a wide range of utilities for model evaluation and tuning, making it an invaluable resource for developing and optimizing machine learning models.

4.3 Feature Selection

For a functioning ML system, the choice of meaningful features is of utmost importance. The features are extracted from the datasets under the use of a Matlab script.

To train and test the ML algorithm, the following four different features are used:

- last_run_number_of_cycles_ago
- last_fail_number_of_cycles_ago
- fail_rate
- number_of_runs

The following paragraphs explain the various features and the reason why they were selected.

last_run_number_of_cycles_ago:

This feature indicates how many cycles have passed since the test was last carried out. It is expected that tests that have not been carried out for a long time tend to fail, because the system underwent many changes.

last_fail_number_of_cycles_ago:

This feature indicates how many cycles have passed since the test was last carried out and failed. If a test failed many cycles ago it may be an indication that the problem which triggered the failure of the test is solved. Vice versa a test that failed recently may be an indication of an unsolved problem.

fail_rate:

The fail rate is the proportion between the number of times a test failed and the number of times a test was executed.

$$\text{fail rate} = \frac{\text{number of times a test failed}}{\text{number of times a test was executed}}$$

This feature may be important. As it may be a good indication of the probability that a test fails in the future.

number_of_runs:

This feature contains information about how often a test had been executed so far. This may, together with the fail rate, indicate the likelihood of a test to fail.

Along with these features, the test results are also extracted from the datasets. This information is then used both to train the model and to evaluate its performance. By incorporating the test results into the training process, the model is able to learn patterns and relationships that help predict future results, ensuring that its predictions are consistent with real-world scenarios.

4.4 Choice of Algorithm

For the implementation, several machine learning algorithms were selected for comparison in order to assess their performance on the given task. The chosen algorithms are:

- Gaussian Naïve Bayes
- Random forest with hyperparameter tuning
- Reinforcement learning

Each algorithm is described in detail in the following sections, with an explanation of the reasons for their selection based on the problem's requirements and the desired outcomes. Additionally, a fourth algorithm based on the failure rate is utilized. While this approach is not a machine learning technique, it serves as a baseline for evaluating test case prioritization and is also explained in the following paragraphs.

4.4.1 Gaussian Naïve Bayes

The Naïve Bayes algorithm is based on the Bayes' theorem. We also make the naïve assumption that our features are conditional independent [28].

For the implementation of the Naïve Bayes algorithm, the machine learning model from the scikit-learn library is used. The model is trained on the training data and generates predictions for the test set in each cycle, as outlined in Section 4.1. The model's output consists of predicted probabilities for each test case in the test set, indicating the likelihood of a test failing. These probabilities are used to rank the test cases in descending order, starting with the test case that has the highest probability of failure. This ranked list can then be utilized to calculate the APFD value, as described in Section 2.2.6.1.

Another output of the model is a list of predictions, categorizing each test as either a pass or a fail. These predictions are used to compute the accuracy, precision, and recall of the model, as discussed in Section 2.2.52.2.5. These metrics are crucial for evaluating the performance of the model.

Finally, various plots are generated based on the model's outputs, which are presented in Chapter 4.7. Additionally, the runtime of the model is measured for performance evaluation purposes.

4.4.2 Random Forest with Hyperparameter Tuning

The random forest classifier combines several decision tree classifiers. Decision trees are described in Section 2.2.2.8.

For the random forest with hyperparameter tuning, different parameters in a certain range are chosen randomly. With these hyperparameters, a part of the dataset is trained and tested. This process is repeated a hundred times. The best result determines the hyperparameters used for the actual training. More information about the hyperparameters is given in section 4.5.

For the implementation of this ML model, the Random Forest Classifier from scikit-learn is used. The hyperparameters which have been determined by the previous step are set.

The model is trained with the training data and predicts the outputs for the test data in each cycle as described in Section 4.1 .With these outputs, certain metrics are determined, which are later used for evaluation.

For the determination of the APFD value test cases are ordered according to their probability of failing.

Different plots are recorded for recall and APFD values. The runtime is also measured.

4.4.3 Reinforcement Learning

As the third ML algorithm, reinforcement learning with Q-learning is used. More detailed information on reinforcement learning is given in Section 2.2.4.

The model gets trained with the training dataset and must determine the output of the test cases in the test dataset. Its output is a prediction for the outcome of the test cases in this dataset. This information is used for the calculation of various metrics. The process is repeated for each cycle as described in Section 4.1.

For the calculation of the APFD values test cases in the datasets are ordered according to their Q-value from the Q-table. With this information, the APFD value can be calculated and used for evaluation purposes.

Different plots are determined, and the runtime is measured.

4.4.4 Model based on Fail Rate

For the model based on fail rate no ML based model is used. The fail rate is used to determine if a certain test case is likely to fail. A higher fail rate means that the test failed often in the past if it was executed. It is assumed that it is likely that the test will fail again in the future if it has a high fail rate.

Tests which are likely to fail are labeled positive, because these are the test cases which we want to execute during our test phase. On the other hand test cases with a low probability to fail will be labeled negative as they won’t be executed.

A threshold needs to be chosen to determine at which fail rate a test in the test dataset is predicted to fail. It is essential to balance the value of the threshold. If the threshold is too high there will be many test cases predicted false positive (We expect the test to fail but it does not). If the threshold is set to low there will be some false negative test cases (We expect the test to pass but it does not).

In this thesis, a certain recall goal is used as a hyperparameter to tune the threshold. The threshold is determined with the training data in a way that the recall of the training set classified with this threshold is close to the recall goal.

The model output is a prediction of which test cases in the test dataset will fail and which ones will pass. With this information several metrics for evaluation are determined.

To calculate the APFD value, the test cases are ranked according to their failure rate. In addition, several graphs are generated based on the model's predictions, and runtime is measured for performance evaluation.

4.5 Hyperparameter Tuning

4.5.1 Random Forest

The hyperparameters which are tuned in this random forest ML algorithm are

- Max_depth
- Num_estimators
- Min_samples_leaf
- Min_samples split
- Boot

In the following sections, the hyperparameters are described. Also, the range within the hyperparameters are chosen randomly is stated.

Max_depth

Defines the maximum depth of the tree. If this value is chosen None, all nodes are expanded until all leaves contain less than min_samples_split samples [29].

This parameter was chosen randomly in the range from 10 to 100 with the step size 10 additional the parameter could also be None. The default value is None.

Num_estimators

Defines the number of trees in the forest [29]. This parameter was chosen randomly in the range from 100 to 1900 with a step size of 200. The default value is 100 [29].

Min_samples_leaf

Defines the minimal number of samples a node requires to be a leaf node

This parameter was chosen randomly in the range from 1 to 3 with the step size 2. The default value is 1.

Min_samples_split

Defines the minimum number of samples required to split an internal node [29].

This parameter was chosen randomly in the range from 2 to 10 with the step size 2. The default value is 2.

Boot

Defines if bootstrap samples are used in the building of trees. If this parameter is chosen false, the whole dataset is used to build each of the trees [29].

This parameter was chosen randomly and it can be true or false. The default value is true.

4.5.2 Reinforcement Learning

The hyperparameters which could be tuned in the reinforcement learning algorithm are

- Alpha
- Gamma
- Epsilon
- Rewards

The following sections, describe the hyperparameters and the range within the hyperparameters are chosen randomly.

Alpha

Alpha is the learning rate. Q-values change much if the learning rate is high and less if the learning rate is small [20].

For this parameter different values between 0.1 and 0.9 have been tried with a step size of 0.1 and the value combination which leads to the best results has been selected.

Gamma

The discount factor γ affects that rewards that are received in earlier steps are valued lower than rewards nearer to the actual step.

For this parameter different values between 0.1 and 0.9 have been tried with a step size of 0.1 and the value combination with leads to the best results has been selected.

Epsilon

The probability value ε determines the probability if the agent decides between the two activities, exploration and exploitation. During exploration, the agent chooses the action randomly. During exploitation, the agent runs the action with the highest Q-value [15].

For this parameter different values between 0.05 and 0.95 have been tried with a step size of 0.05 and the value combination which leads to the best results has been selected.

Rewards

For each of the possible outputs true positive, true negative, false positive, false negative a reward is needed to indicate the system if its decision was good or bad. In the case of the bad decision the reward is negative, so the system is punished.

In the case of a true positive classification, e.g. the agent classifies the test failing and it fails. The model gets a big positive reward.

In the case of a true negative classification, e.g. the agent classifies the test passing and it passes. The model gets a small positive reward.

In the case of a false positive classification, e.g. the agent classifies the test failing and it passes. The model gets a small negative reward.

In the case of a false negative classification, e.g. the agent classifies the test passing and it fails. The model gets a big negative reward.

For each parameter different values between 1 and 100 (respectively -1 and -100) have been tried with a step size of 20 and the value combination with leads to the best results has been selected.

4.5.3 Fail Rate

The only hyperparameters which could be tuned using the fail rate algorithm is the recall goal.

Recall goal

The recall goal is used to determine the threshold of the fail rate model. Therefore, the training data is used, and the threshold is set in a way that the recall of the trainings data fits the recall goal.

4.6 Training

For the training/evaluation/test split a 70/15/15 split is used. This means that 70% of the dataset is used for training, 15% for evaluation and 15% for testing. For the split the split function is imported from the scikit learn software library.

The training data set grows with each regression step as the graphs are plotted. The metrics should also get better at each regression step, as the training dataset gets bigger [22].

4.7 Application of the System

The goal of the system is to reduce the number of executed test cases through prioritization. This means that only those test cases with a high probability of failure need to be executed. This saves processing time and hardware capacity. Various ML algorithms are used to prioritize test cases. Low priority test cases don't need to be executed. High priority test cases should be executed. That means the ML algorithms are used to split all test cases into test cases that should be executed (which have high priority) and rejected test cases (with low priority).

5 Evaluation

In the first section of this chapter the datasets used for the training and evaluation are described in detail. The second section describes the used parameters for the performance evaluation. In the third section the actual measurement is described. The fourth section contains the interpretation of the measurement results. The timing analysis of the different models is described in section five of this chapter.

5.1 Datasets

For training and evaluation, four different datasets were used. Two of them are industrial datasets from ABB Robotics Norway, called Paint Control and IOF/ROL. Another dataset, named Bosch, originates from an industrial project by Bosch, and the last dataset is the Google Shared Dataset of Test Suite Results (GSDTSR). Three datasets are frequently used in various research studies referenced in this master's thesis [6][22], making them well-established and reliable sources for training machine learning models..

These datasets are valuable for machine learning tasks because they contain detailed information about the executed tests, such as the test ID, the time of the last execution, and the result (pass or fail). The datasets are organized into cycles, where each cycle represents a regression step. Within each cycle, multiple tests are run and each test is assigned to the cycle in which it was run. This structure allows test performance to be analyzed and trends to be identified over time, which is essential for building predictive models. The GSDTSR dataset is the largest of the four and provides a wide range of test results, making it ideal for training robust machine learning models. The Bosch dataset, on the other hand, is smaller but of particular importance due to its proprietary nature. It was collected as part of an industrial research project within Bosch, making it a valuable source of data for understanding test results in real industrial contexts.

In summary, the combination of these datasets provides a rich and diverse structure for training and evaluating machine learning models, with the Bosch dataset providing proprietary insights from a real-world industrial project.

Table 4 contains details about the number of executed tests, number of cycles and number of failed tests from the different test sets. The GSDTSR dataset is by far the largest dataset while the Bosch dataset is the smallest.

Name of the test set	Number of executed tests	Number of failed tests	Number of tests which did not find failures	Number of cycles
GSDTSR	1048575	2859	1045716	266
Bosch	530	130	400	48
Paint control	25594	4956	20638	352
IOF/ROL	32260	9289	22971	320

Table 4: Information about the Data Sets

5.2 Performance Evaluation

For the performance evaluation of the classifiers different metrics were used these are:

- Recall
- Precision
- APDF

These metrics are explained in Section 2.2.

This thesis does not broach the issue of performance evaluation with accuracy. Because the datasets contain many tests which did not fail so the accuracy may be high even if none or few failed test are found by the model.

5.3 Hyperparameters

5.3.1.1 Random Forest hyperparameter

For the random forest model, the following hyperparameter are found.

Hyperparameter	Value
Max_depth	None
Num_estimators	100
Min_samples_leaf	2
Min_samples split	4
Boot	true

Table 5: Selected Hyperparameters for Random Forest

5.3.1.2 Reinforcement learning hyperparameter

For the reinforcement model, the following hyperparameter are set.

Hyperparameter	Value
Alpha	0.8
Gamma	0.6
Epsilon	0.05
True positive reward	100
True negative reward	1
False negative reward	-100
False positive reward	-1

Table 6: Selected Hyperparameters for Reinforcement Learning

5.3.2 Calculation of Precision and Recall

Precision and recall and their calculations are described in Section 2.2.

The ML algorithms are each trained with a training dataset which consists of all tests which are executed till the end of the current cycle. The tests are always performed on the same test-set which size is 15% of the whole dataset and contains the tests and results of the last cycles.

5.3.3 APFD Calculation

For the APFD analysis the new Test cases need to be ranked before each cycle and then the APFD value needs to be calculated. Calculation of the APFD value is described in Section 2.2.6.1. The calculated values are then used to retrain the model. Calculation of the APFD value therefore needs a different approach in terms of software than the calculation of recall and precision.

First the model is trained with the test cases of the first cycle. It then predicts the probability of failure for each test case of the second cycle. Note that in case of the fail rate, it is used as probability of failure also for later cycles. The test cases are then ranked by the probability of failure ranking values tests a higher probability to fail on top as it is more crucial to execute them.

After the ranking the APFD value is calculated taking the results of the ranked tests into account, note that the APFD value is higher if a failing test case is ranked high up.

In this manner it is iterated over all cycles and the APFD value is calculated for each cycle.

5.4 Results

5.4.1 Precision and Recall on GSDTSR dataset

The following figures, Figure 8 and Figure 9 are showing the recall for different models over different regression cycles. The recall is shown in Figure 8. Figure 9 shows which percentage of test cases are classified as failing (true positives plus false positives), note that a method that classifies a lower number of test cases as failing will increase the performance in terms of runtime but also will it make harder to get better performance values. Figure 10 shows a

comparison between the different precisions of the models. The GSDTSR is the biggest dataset analyzed in this thesis and contains 1.048.575 test cases in 266 test cycles. Out of all these cases 0.27% have the verdict fail.

5.4.1.1 Fail Rate Model

The blue curve in Figure 8 and Figure 9 shows how the fail rate model performs on the GSDTSR dataset. The recall turns out to be values around 0.7 while the percentage of the test cases classified as failing is very low, close to zero. At the first few circles the recall is a bit higher than in the later cycles. This is because the percentage of test cases classified as failing is much higher at approximately 0.35%. At later cycles we see learning effects and the recall is rising, since more training cycles are completed. If we look at the blue curve in Figure 10 we see that in the first few cycles the precision is lower than in the later cycles were less test cases are classified as failing. The lower precision value is at around 0.025 the higher value fluctuates between 0.2 and 0.4.

5.4.1.2 Naïve Bayes Model

At the yellow curve in Figure 8 we see the recall of the naïve bayes model performing on the GSDTSR dataset. We see that the recall and the percentage of test cases classified as failing (yellow curve in Figure 9) start with very low values at the first few cycles. Then it increases rapidly. But the test cases classified as failing also increase rapidly, this might be a reaction on the low recall of the first few cycles. The system tries to compensate by selecting a larger number of test cases as failing. This peaks approximately at the 8 cycle where nearly all test cases are classified failing. After this peak learning results can be perceived, because the percentage of test cases selected as failing decreases continuously while the recall stays high only decreasing slightly. The recall in this stage is about 0.9 or higher. The percentage of test cases classified as failing drops from nearly 100% to about 10%. At Figure 10 we see that the precision of the naïve bayes model is generally low on this dataset.

5.4.1.3 Random Forest Model

The green graph at Figure 8 shows the results of the random forest model applied on the GSDTSR dataset. The figure shows that the recall is varying around 0.4. The percentage of test cases classified as failing is very low close to zero. There are strong variations in the recall it varies between 0.2 and 0.55. Figure 10 shows that the precision is very high with the random forest model. Most of the time it varies between 0.4 and 0.9.

5.4.1.4 Reinforcement Learning Model

In Figure 8 the red curve shows the performance of the reinforcement learning model on the GSDTDSR dataset. The percentage of test cases classified as failing is very low close to zero while the recall varies around 0.58. Because of the axis resolution in Figure 8 we don't see a learning effect. But Figure 28 where only the recall is displayed clearly shows that the recall is getting higher in later test cycles. In Figure 10 the precision of the reinforcement learning model is displayed, it is quite low at 0.025.

5.4.1.5 Comparison

Comparing the recalls in Figure 8 we notice that the naïve bayes model leads to the highest recall but is also classifying large chunks of the test cases as failing. In practice this means many test cases would have to be executed. The second-best performance concerning the recall is done by the fail rate model. With a recall around 0.7. But the recall varies at this model. The least fluctuating recall we recorded with the reinforcement learning model. But it is not as high as the recall of the other models. The worst performance in terms of both magnitude and stability of the recall is done by the random forest model.

As it is displayed in Figure 10 the direct comparison between the different precisions shows that the random forest model performs best on the GSDTSR model in terms of precision. Its precision is way higher than the precision of other models. The fail rate model also has a quite good performance at this parameter. The naïve Bayes and the reinforcement learning model both underperform in this category.

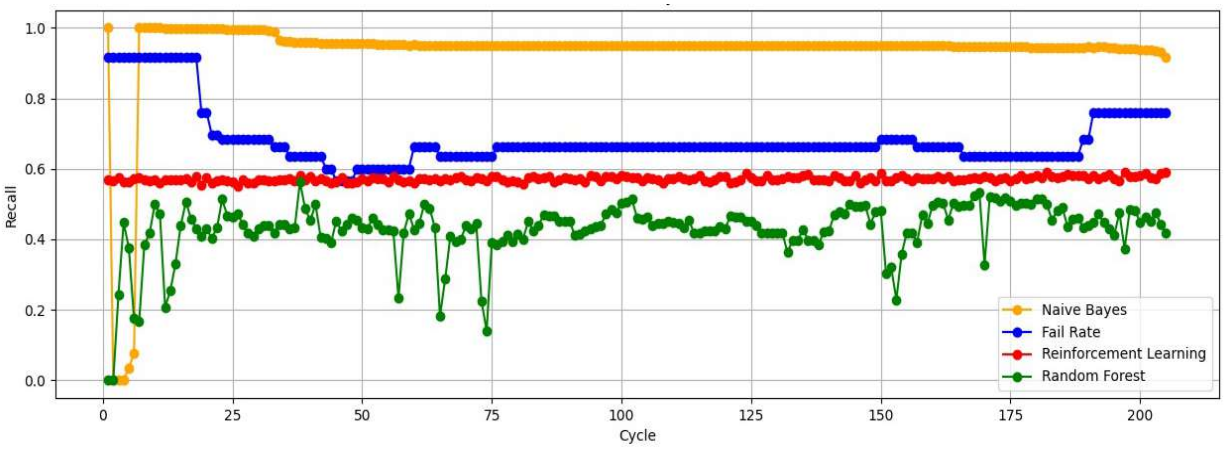

Figure 8: Recall on GSDTSR Dataset

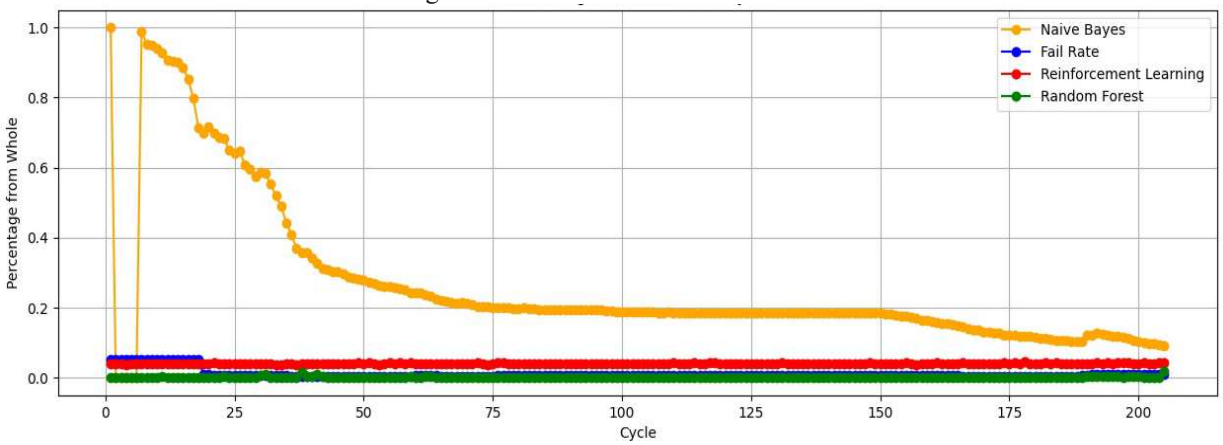

Figure 9: Percentage from whole on GSDTSR Dataset

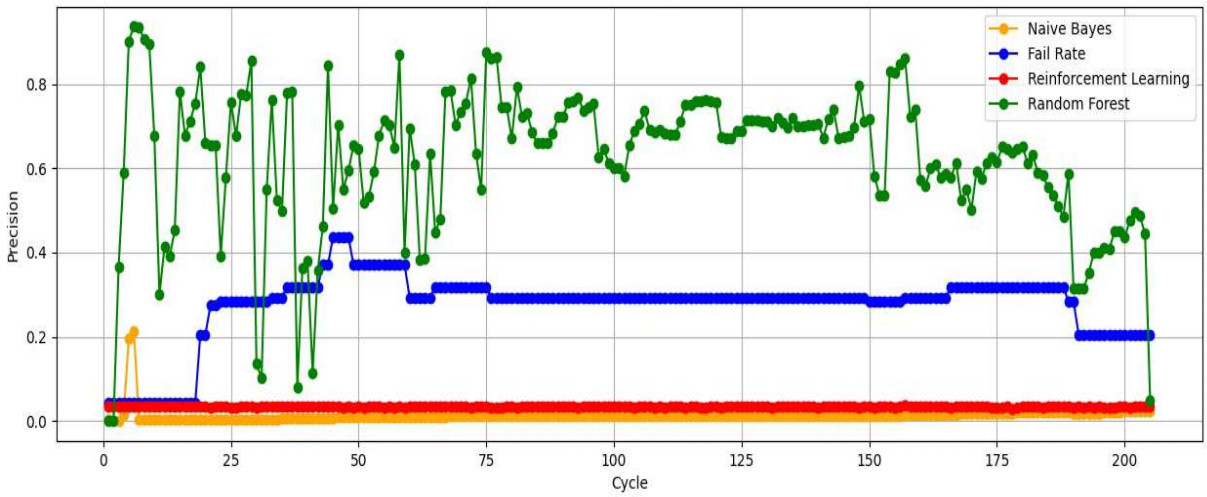

Figure 10: Precision on GSDTSR Dataset

5.4.2 Precision and Recall on Bosch dataset

In the figures Figure 11 to Figure 13 the recall and the precision of different models on the Bosch dataset is shown. The Bosch dataset is the smallest dataset analyzed in this thesis. The dataset contains 530 test cases and has 48 regression cycles. 24.52 % of the test cases in the Bosch dataset fail. Figure 11 shows the recall of different models. Figure 12 shows which percentage of test cases are classified as failing. In Figure 13 we see a comparison of the precisions of the different models used on the Bosch dataset.

5.4.2.1 Fail Rate Model

The blue curve in Figure 11 shows the recall of the fail rate model on the Bosch dataset. In the first twelve cycles the recall is about 0.35 but there is also just a small number of test cases selected as failing. The test cases selected as failing are kept constant by the model. The recall follows this trend. At the cycles 13 to 30 the percentage of test cases classified as failing is increased again. It varies around 0.4. The recall at this phase varies around 0.8. An interesting phenomenon which can be recognized is that the recall follows the changes of the percentage of test cases classified as failing. There is no relevant learning success visible. The reason could be that the dataset might be too small. Figure 13 displays that the precision of the fail rate model is pretty high when only a few test cases are classified as failing. At the moment where more test cases are classified as failing the precision drops from almost a 100% to about 60%.

5.4.2.2 Naïve Bayes Model

In Figure 11 the yellow curve shows the recall of the naïve bayes algorithm applied to the Bosch dataset. At the first few cycles about 80 % of the test cases are classified as failing the recall is 1. At cycle 8 the test cases classified as failing drop to around 20 % and varies around this value for the rest of the cycles. At cycle 8 the recall also drops. The recall increases over the next cycles, so a small learning success can be recognized. In Figure 13 the precision of the naïve bayes algorithm is shown. We see the precision start low and then increases with each cycle due to the learning progress. It starts at almost 0 and increases up to 80%.

5.4.2.3 Random Forest Model

The green graph at Figure 11 shows the performance in terms of recall of the random forest model on the Bosch dataset. The percentage of test cases classified as failing is about 10 % at the first 15 cycles of the graph. Sometimes there are some peaks in this part of the graph the biggest one at cycle 7 where 60 % of the test cases are classified failing. The recall follows the course of the test cases classified failing. It is about 0.35 with some peaks. In the later cycles the percentage of test cases classified failing is increased slightly at each new cycle. The recall also increases but with a slightly higher rate. A small learning success is visible. Figure 13 displays that the precision of the random forest algorithm is very high, most of the time it is over 70%.

5.4.2.4 Reinforcement Learning Model

In Figure 11 the red curve shows the course of the recall produced by the reinforcement learning model applied to the Bosch dataset. The percentage of test cases classified as failing starts at about 15 % and is then increased slightly on the following test cycles. The recall follows the course of the test cases classified failing and increases at each test cycle. Displayed in Figure 13 we see that the precision is generally medium and only drops slightly from 0.6 to 0.4 when more test cases are classified as failing.

5.4.2.5 Comparison

Comparing the recall of the different models we notice that the fail rate model has the highest recall, but the recall varies strongly at about 0.8. The second-best recall is produced by the naïve bayes model, but it also varies strongly. Random forest and reinforcement learning model are performing similar in terms of recall. In both the recall varies at about 0.4. In the experiment with the random forest model the fewest test cases are classified failing.

In all four graphs the recall follows the course of the percentage of test cases classified as failing. Only with the naïve bayes and the random forest model, small learning success can be spotted. The reason is that the dataset is not big enough to result in higher learning success.

Comparing the precisions in Figure 13 it is visible that all models perform good in terms of precision. Concerning this parameter the best model is the random forest model, also the naïve bayes and the fail rate model perform very good. The reinforcement model is not as good but only slightly weaker than the others.

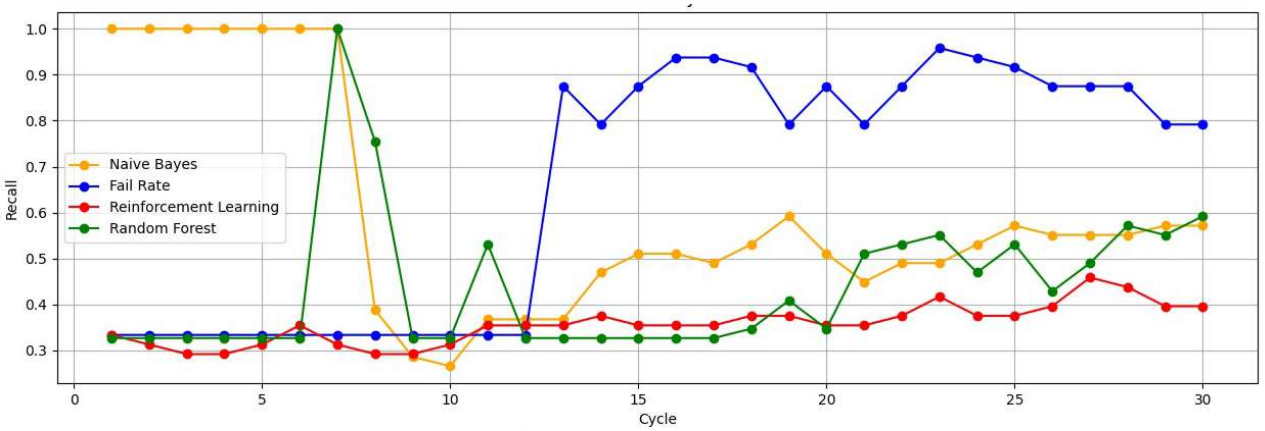

Figure 11: Recall on Bosch Dataset

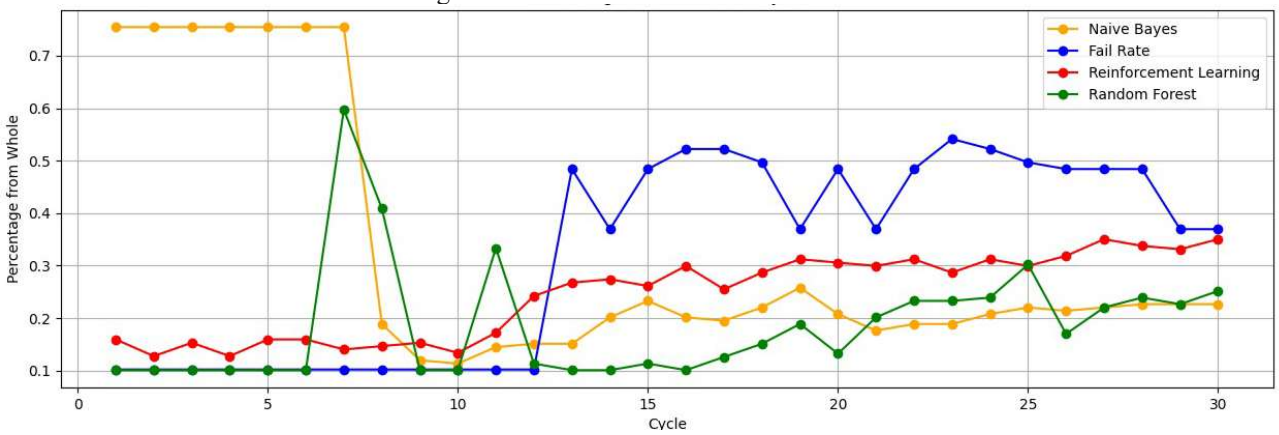

Figure 12: Percentage from whole on Bosch Dataset

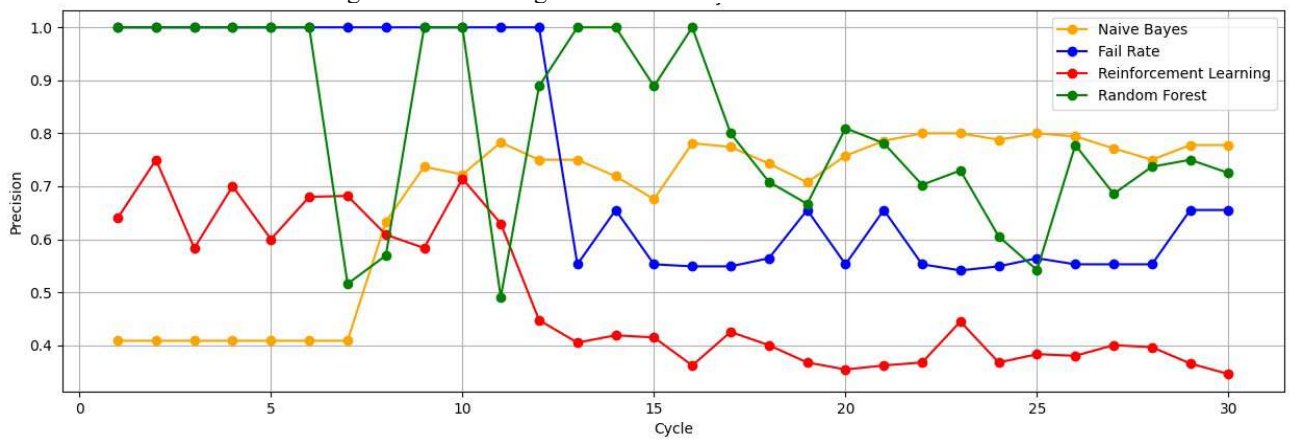

Figure 13: Precision on Bosch Dataset

5.4.3 Precision and Recall on paint control dataset

The figures Figure 14 to Figure 16 show how the different models used perform, in terms of recall, on the paint control dataset. With 25594 test cases the paint control dataset is a medium size dataset. 19.36 % of its test cases fail. Figure 16 displays the performance in terms of precision for all the different models.

5.4.3.1 Fail Rate Model

In Figure 14 the recall of the fail rate model used on the paint control dataset is displayed in blue. At the first test cycle a high number of test cases is classified as failing. The recall is also high at about 0.8. In the next few cycles, the percentage of test cases classified as failing is reduced to about 5 %. The recall drops significantly to about 0.3. Then at about cycle 10 till the last cycle, the number of test cases classified increases to about 95% of the whole dataset. The recall also increases to about 0.95. As one can see in Figure 16 the precision of the failrate model is low but also very constant at about 10%.

5.4.3.2 Naïve Bayes Model

The yellow curve in Figure 14 shows the recall of the naïve bayes model on the paint control dataset. The percentage of test cases classified as failing in this graph varies in a huge range. The recall follows this trend. In the later cycles a learning effect is visible as the recall compared to the number of test cases selected as failing is increasing. In Figure 16 the precision of the naïve bayes model is displayed it fluctuates highly between 10 and 80% but in generall its values are pretty high.

5.4.3.3 Random Forest Model

At Figure 14 the green curve displays the recall of the random forest model used on the paint control dataset. In Figure 15 the percentage of test cases classified as failing stays pretty low, under 10 %, most of the time but there are also many peaks. The highest at about 80%. The recall follows this trend in the early cycles till about cycle 100. After about cycle 100 a small learning effect is visible as the recall compared to the number of test cases selected as failing is increasing. Figure 16 shows that the precision fluctuates highly between 10 and 80%.

5.4.3.4 Reinforcement Learning Model

The red curve in Figure 14 displays the recall of the reinforcement model on the paint control dataset is shown. It shows that the recall is relatively constant at about 0.6. The percentage of test cases classified as failing is also relatively constant at about 20%. In Figure 37 only the recall is displayed because of the different axis resolution a small learning effect is visible. The recall increases from about 0.6 to 0.63. For the reinforcement learning model Figure 16 shows a constant precision of about 25%.

5.4.3.5 Comparison

Comparing the recall of the different models the fail rate model produces the highest recall at nearly 1. The recall is also relatively constant. The recall of the naïve bayes model is the second highest but it varies strongly. The third best recall is provided by the random forest model, but it varies strongly. The lowest but also the most constant recall is produced by the reinforcement learning model. The lowest number of test cases selected as failing is provided by the reinforcement learning model and it is also pretty low.

An interesting observation is that the recall often follows the trends of the percentage of test cases selected as failing. A learning effect is visible for most of the models.

In terms of precision three models perform very good. Naïve bayes and random forest have generally high accuracy values but high variation. The reinforcement model has a precision of about 25% percent but stays consistent over all cycles. The fail rate model performs worst in terms of precision with a constant precision of about 10%.

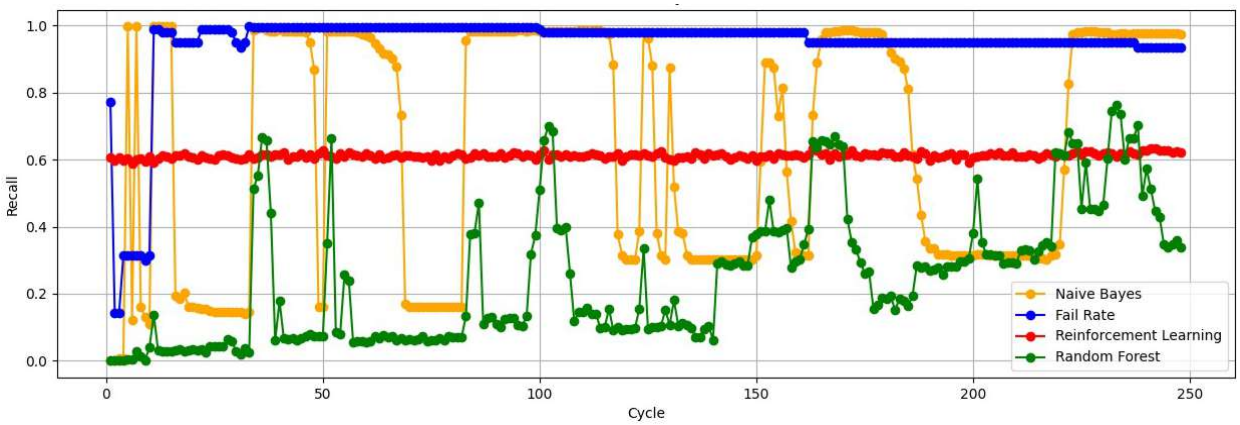

Figure 14: Recall on Paint Control Dataset

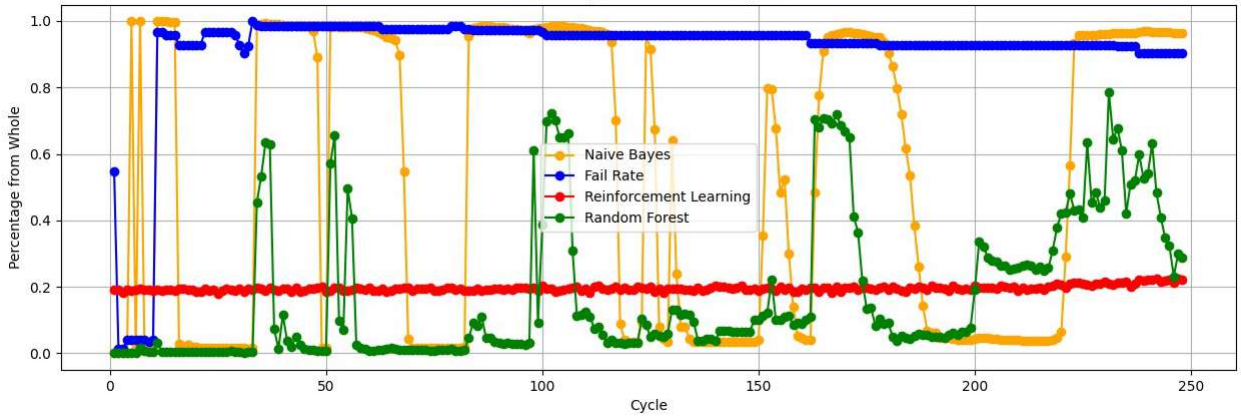

Figure 15: Percentage from whole on Paint Control Dataset

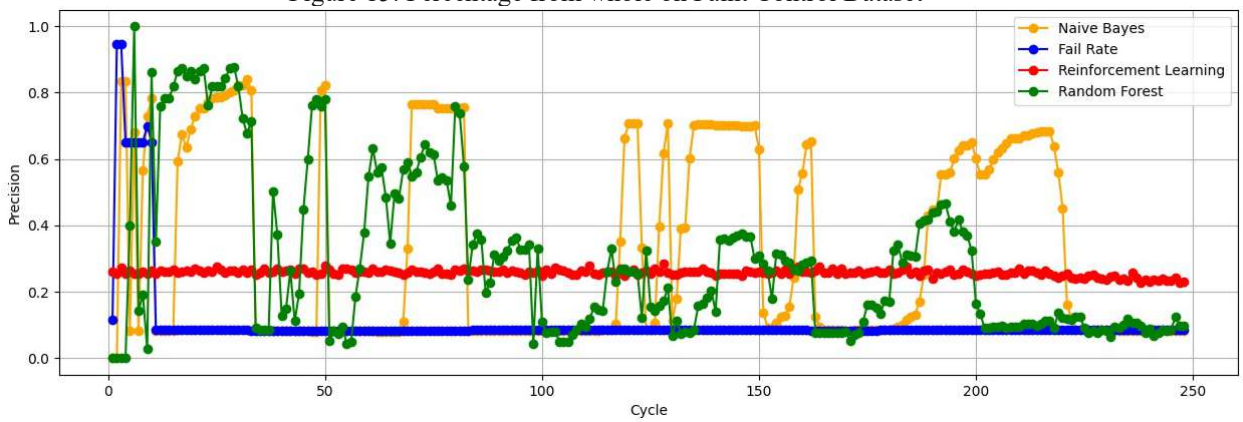

Figure 16: Precision on Paint Control Dataset

5.4.4 Precision and Recall on IOF/ROL dataset

In the figures Figure 17 to Figure 18 the recall of the different models used on the IOF/ROL dataset is displayed. Figure 19 shows the precision of the different models on the IOF/ROL dataset. The IOF/ROL dataset contains 32260 test cases so its of medium size compared to the other datasets. 28.79 % of its test cases fail.

5.4.4.1 Fail Rate Model

The fail rate model does not work for this dataset. The reason is that test cases are not repeated in this dataset so no fail rate can be calculated.

5.4.4.2 Naïve Bayes Model

The yellow curve in Figure 17 displays the performance of the naïve bayes algorithm in terms of recall. At the first cycles, till about cycle 100 the recall adapts to the percentage of test cases classified as failing. Both fluctuate up and down. In the later Cycles the gap between does two starts to increase. This indicates that a learning effect happened. For the precision displayed in Figure 16, the naïve bayes model has a pretty high precision fluctuating between 0.3 and 0.7.

5.4.4.3 Random Forest Model

In Figure 17 the green curve shows the recall of the random forest algorithm. It shows that for the first 120 cycles the recall behaves similar to the percentage of test cases classified as failing. Both fluctuate strongly. After the 150 cycle a learning effect is strongly visible as the two values drift apart. In Figure 16 the precision of the random forest is displayed, it is pretty constant only fluctuating slightly around 40%.

5.4.4.4 Reinforcement Learning Model

The recall of the reinforcement algorithm is displayed by the red curve in Figure 17. It is relatively constant at about 0.95. Also, the percentage of test cases selected as failing is constant at about 80%. We see a small learning affect as the recall is increasing slightly over time from 0.94 to 0.95. Displayed in Figure 16 we see the precision of the reinforcement learning, it is constant at 40%.

5.4.4.5 Comparison

Comparing the different recalls with each other the reinforcement algorithm performs best at about 95%. Its recall is also very constant. The second-best performance in terms of recall is shown by the naïve bayes algorithm, which recall is a bit lower and not as stable. A similar only slightly worse performance is shown by the Random Forest algorithm.

Precision wise the tree different models perform similar. The reinforcement learning algorithm has the most constant results. While the naïve Bayes algorithm gives slightly better values but also fluctuates more.

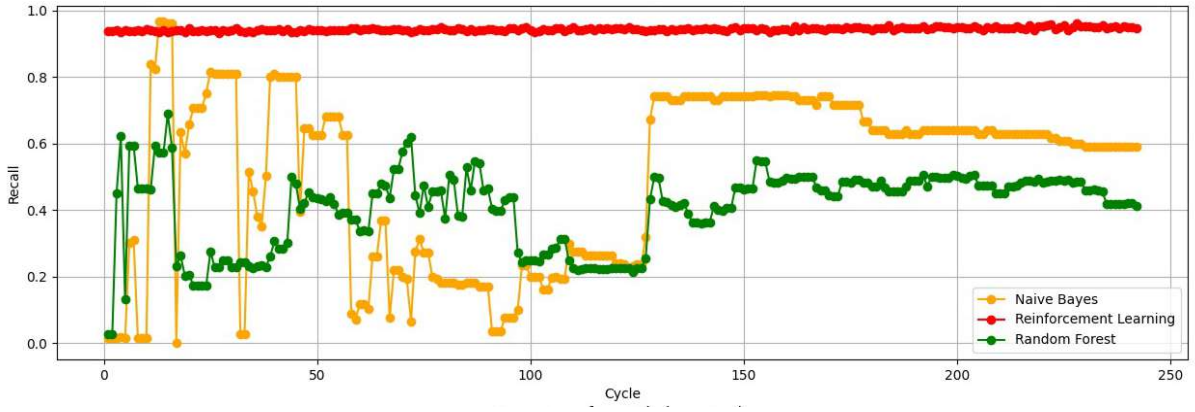

Figure 17: Recall on IOF/ROL Dataset

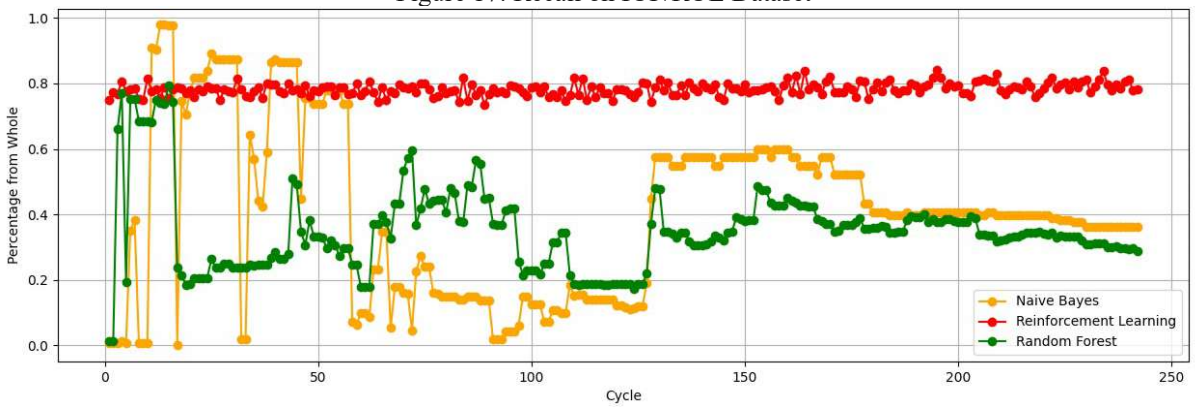

Figure 18: Percentage from whole on IOF/ROL Dataset

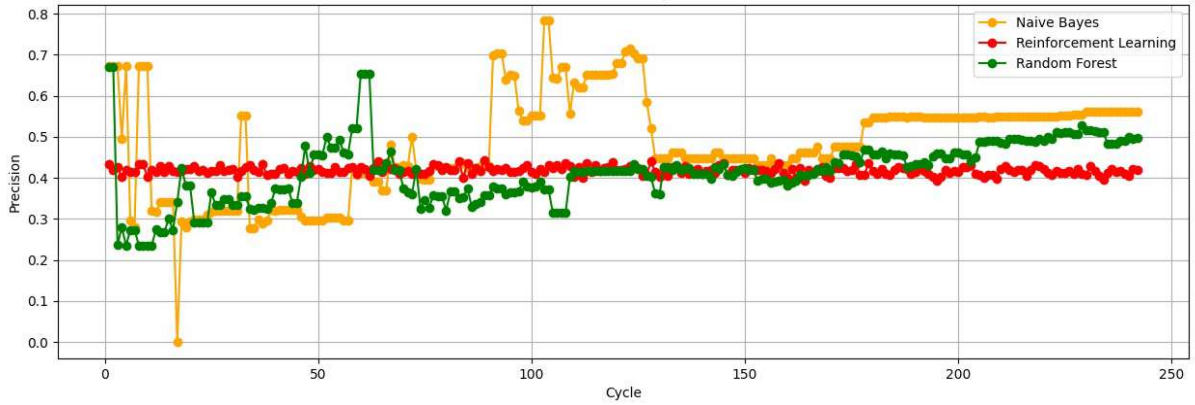

Figure 19: Precision on IOF/ROL Dataset

5.4.5 APFD on GSDTSR dataset

Figure 20 presents a comparison of the APFD values across different models on the GSDTSR dataset. These values are calculated as described in Section 5.3.3. It is important to note that some values are missing from the diagram, as APFD values cannot always be calculated for every test case.

The failure rate model generally achieves very high APFD values, ranging between 0.8 and 1, with most values equal to 1, indicating strong performance in ranking the test cases. Similarly, the Naïve Bayes model also shows high APFD values, suggesting performance close to that of the failure rate model.

In contrast, the Random Forest model exhibits somewhat lower values compared to both the failure rate and Naïve Bayes models, indicating relatively weaker performance.

The Reinforcement Learning model demonstrates the weakest performance in terms of APFD on this dataset. This suggests that, in terms of ranking test cases, the Reinforcement Learning model is less effective compared to the other models on the GSDTSR dataset.

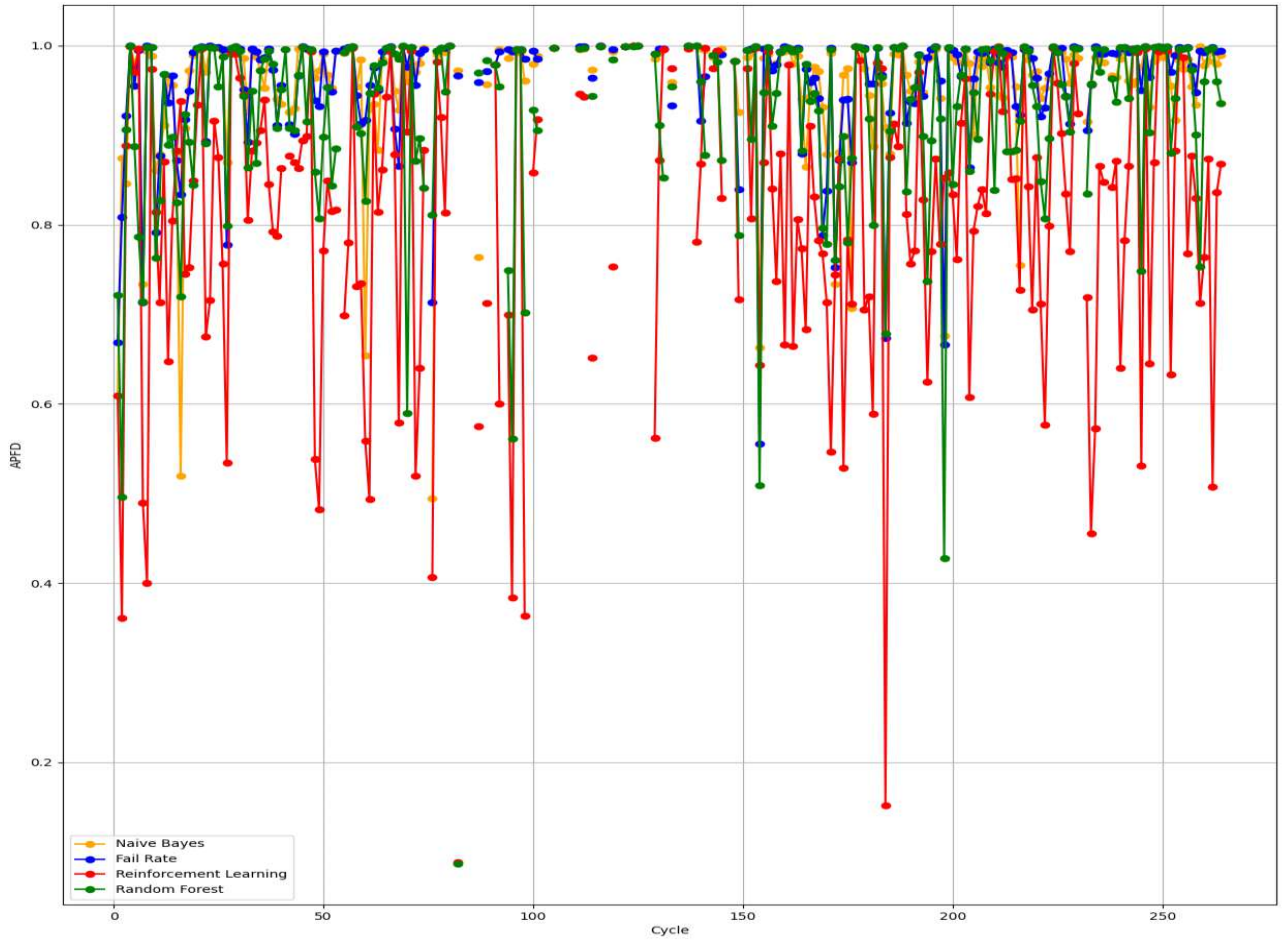

Figure 20: Comparison of APFD values on GSDTSR Dataset

5.4.6 APFD on Bosch dataset

This section presents a comparison of the APFD values across different models, evaluated on a dataset provided by Bosch. The APFD metric is used to assess the effectiveness of test case prioritization, with higher values indicating better performance in detecting faults early in the testing process. The Bosch dataset contains a series of test cases with known outcomes, enabling a thorough evaluation of each model's ability to prioritize test cases based on their likelihood of failure.

In this analysis, several models are compared, including failure rate-based models and machine learning algorithms. The APFD values are computed for each model to assess their relative performance in ranking test cases. The results are visually represented in Figure 21, where the APFD values for each model are plotted, revealing distinct performance characteristics.

The failure rate model fluctuates between 0.9 and 0.6, showing high instability, likely due to the small size of the dataset. The Naïve Bayes model also fluctuates but to a lesser extent than the failure rate model, with values ranging from 0.6 to 0.8. The Reinforcement Learning model demonstrates even more fluctuation than the other two, with values ranging from 0.4 to 0.95, indicating lower stability.

In contrast, the Random Forest model exhibits higher and more consistent APFD values, ranging between 0.65 and 0.95, and performs better than the other models.

These fluctuations highlight the varying degrees of stability and performance across the different models on the Bosch dataset.

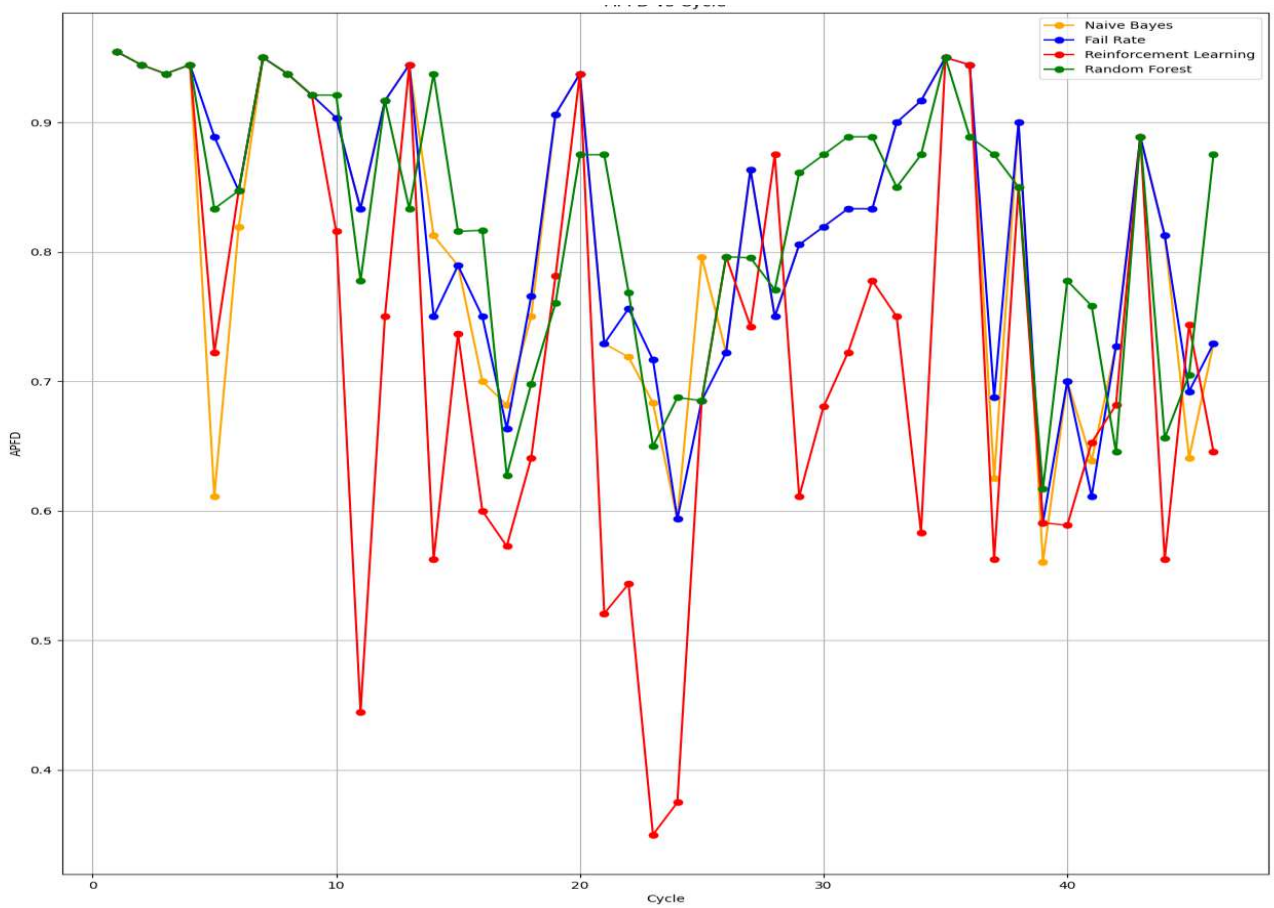

Figure 21: Comparison of APFD values on Bosch Dataset

5.4.7 APFD on paint control dataset

This section presents a comparison of the APFD values on the Paint Control dataset (Figure 22).

The APFD values for all models in the Paint Control dataset exhibit significant fluctuation.

The failure rate model fluctuates substantially, ranging between 0.3 and 0.95, with APFD values generally lower than those of the other models. The Naïve Bayes model also fluctuates

significantly, ranging between 0.1 and 0.95, yet it demonstrates relatively strong performance overall.

The Random Forest model performs the best, with values fluctuating between 0.25 and 0.95, indicating more consistent and effective test case prioritization. In contrast, the Reinforcement Learning model shows the weakest performance, with values ranging from 0.1 to 0.95, suggesting that this model is less stable and less effective in ranking test cases compared to the others.

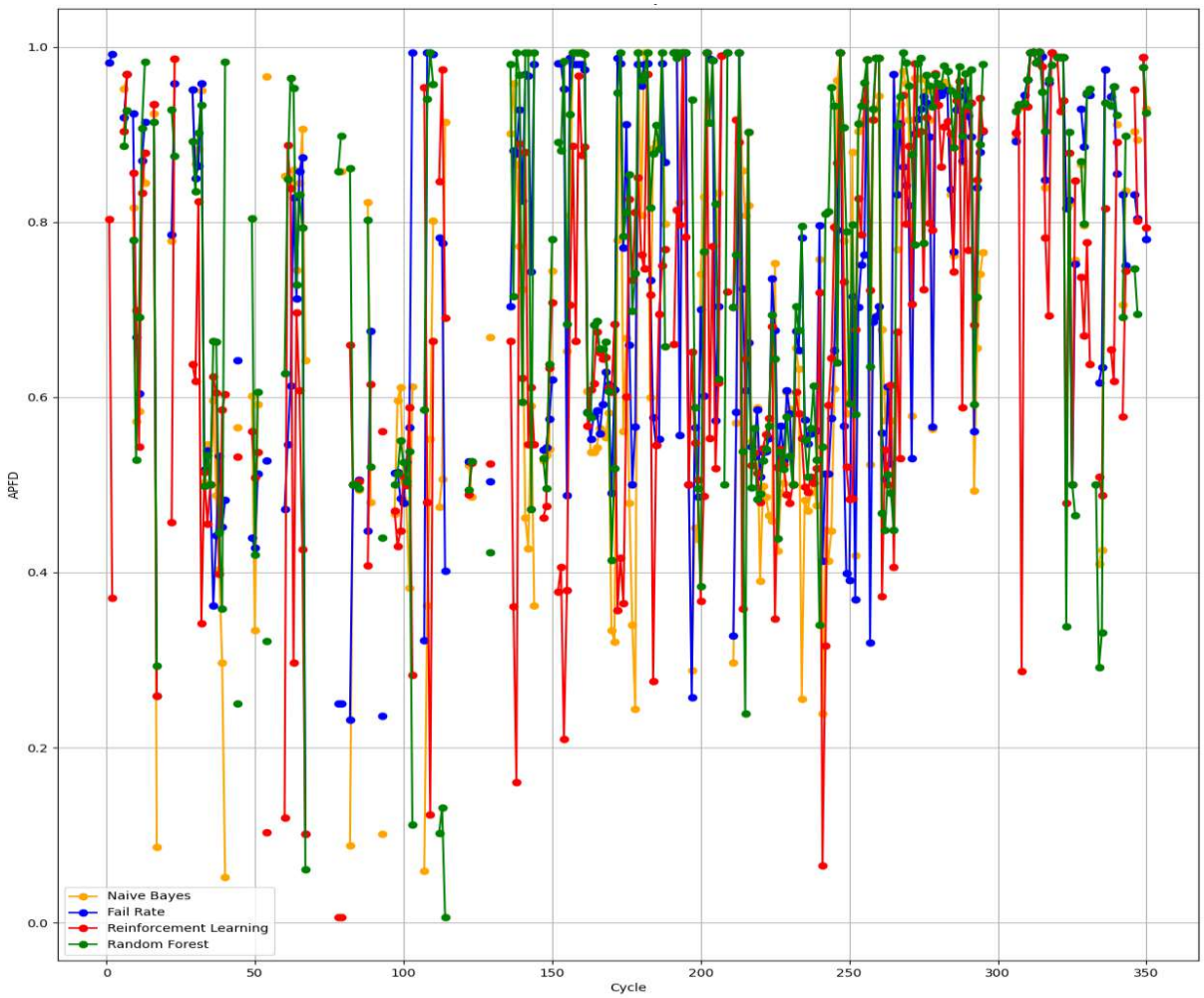

Figure 22: APFD on paint control Dataset

5.4.8 APFD on IOF/ROL dataset

Figure 23 illustrates the distribution of APFD values across different models on the IOF/ROL dataset. A notable trend is that most of the values tend to cluster around an APFD value of approximately 0.5, reflecting a moderate level of performance for all models.

Note that the failure rate model again cannot be calculated for this dataset, as the tests in the IOF/ROL dataset do not repeat, making it impossible to compute a failure rate. The Naïve Bayes model, while also showing some fluctuation, performs slightly better overall. Its APFD values range between 0.4 and 0.8, indicating that this model is more effective at ranking test cases

compared to the failure rate model, with an ability to detect faults more consistently across different test cases.

The Random Forest model shows the most pronounced fluctuation, with APFD values spanning from 0.1 to 0.95. Despite the higher level of fluctuation, this model demonstrates the highest overall APFD values, signalling that it is the most effective at prioritizing tests in a way that maximizes fault detection early in the testing process. The wide range of values suggests that the model is capable of adapting to a variety of test cases and contexts, achieving high performance in certain instances.

The Reinforcement Learning model, like the Random Forest model, exhibits considerable fluctuation, with values ranging from 0.2 to 0.8. Although the APFD value is similar to that of the other models, the Reinforcement Learning model shows more variability, which could indicate a less consistent performance across different test cycles.

The overall performance differences across these models indicate that while there is some fluctuation in all the models, the Random Forest model stands out due to its consistently high APFD values, despite its greater variability. In contrast, the failure rate and Naïve Bayes models offer more stable but slightly less effective performance, and the Reinforcement Learning model tends to be more volatile, suggesting potential issues with stability and fault detection across different test cases. The Random Forest model offers the best trade-off between high fault detection and adaptability to dataset characteristics, highlighting the importance of balancing stability and performance when selecting a model for test case prioritization.

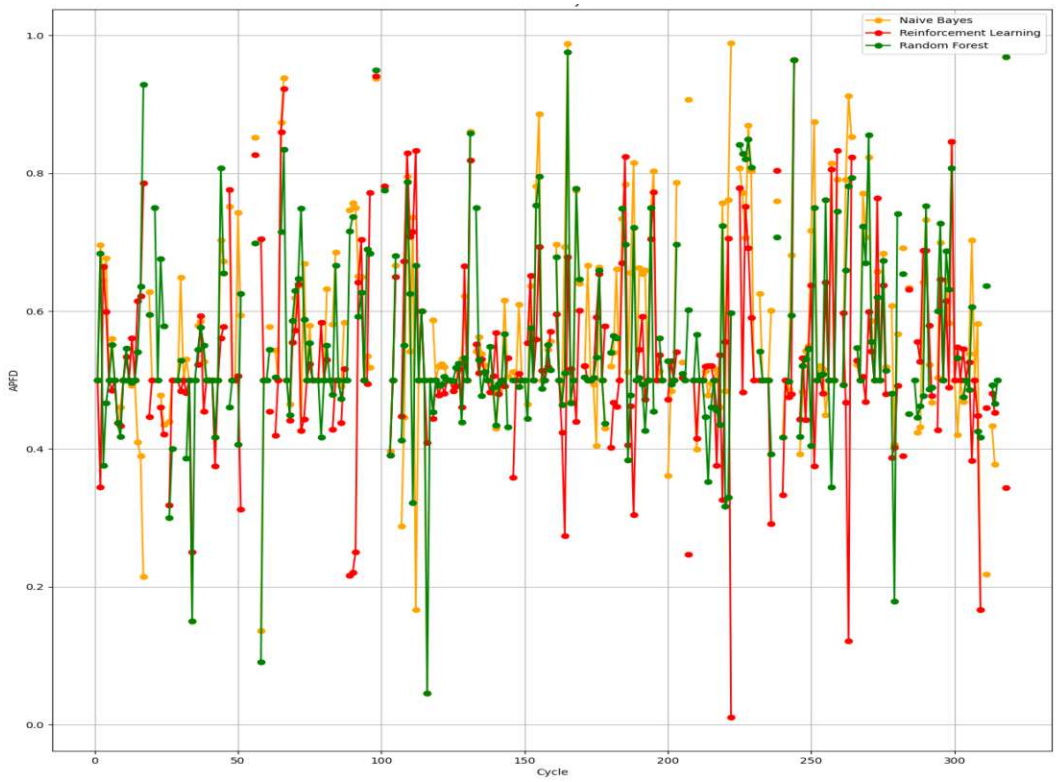

Figure 23: APFD on IOF/ROL Dataset

5.5 Timing Analysis

In this paragraph our four different models are compared in terms of runtime. As the goal of ranking test cases is to reduce test time this parameter is crucial to compare the different algorithms.

The training and test are done on a Windows PC with AMD Ryzen 9 5900X processor and NVIDIA GEFORCE GTX 1050 graphic card. The measured times include preprocessing of data and training of the model, for the APFD calculation it also includes the ranking of the test cases at each cycle.

The results are shown in two different tables
Table 7 shows the timing for the calculation of recall and precision.
Table 8 shows the average timing results for the calculation of the APFD value for 50 runs. Note that these take longer as a ranking of the test cases is required at each cycle.

Larger dataset like the GSDTSR dataset generally take more time compared to the smaller ones like the Bosch dataset. But overall the time for the calculation is reasonable considering that machine learning is included in the calculation process and each model needs to be trained after each cycle.

The naïve bayes model is the fastest model, for the largest dataset it only takes about 178 seconds. The fail rate analysis is also very fast, it takes only about 604 seconds for the largest dataset. Reinforcement learning is a bit slower (1260 sec) and random forest is the slowest model (4119 sec). The other datasets show similar results.

	Fail rate	Naïve Bayes	Random forest	Reinforcement learning
GSDTSR	604 sec	178 sec	4119 sec	1260 sec
Bosch	4 sec	1 sec	1 sec	2 sec
Paint control	23 sec	7 sec	142 sec	38 sec
IOF/ROL	-	7 sec	104 sec	51 sec

Table 7: Timing analysis for Recall and Precision Calculation

	Fail rate	Naïve Bayes	Random forest	Reinforcement learning
GSDTSR	152 sec	264 sec	6329 sec	1291 sec
Bosch	1 sec	1 sec	7 sec	1 sec
Paint control	5 sec	9 sec	249 sec	47 sec
IOF/ROL	-	11 sec	127 sec	56 sec

Table 8: Timing Analysis for APFD Calculation

6 Conclusion

The final chapter summarises the key findings that have emerged from the research and analysis undertaken within the thesis. It includes a thorough discussion of the findings, their implications and their relevance within the broader context of the field. In addition, this chapter aims to outline potential follow-up projects that could extend or deepen the scope of the current work.

6.1 Results and Insights

In this thesis four different models for test prioritization are executed and compared to each other. The used metrics are recall, precision and the APFD value. The different models are also compared in a timing analysis as the runtime is a crucial factor in the field of test case prioritization.

The first model is not based on machine learning, instead, it prioritizes test cases based on their failure rate. It serves as a baseline for comparison with the other models. This approach performs well across most datasets in terms of recall, precision, and the APFD value. The recall remains relatively stable across the datasets. However, for datasets where test cases appear only once, the failure rate model cannot be applied. In terms of APFD, the failure rate model also delivers strong performance. Furthermore, the calculation of this model is very fast.

The second model, the Naïve Bayes machine learning model, is relatively simple and requires minimal training time. It is the fastest among the four models compared. In terms of recall, it yields good results, for instance, on the GSDTSR dataset, it performs best with recall values reaching up to 95%. However, the APFD values for the Naïve Bayes model show significant fluctuations, although they remain generally good overall.

The third model, the Random Forest model, excels in the precision category, achieving the highest precision across many datasets. In terms of recall, it lags behind some of the other models on several datasets. The Random Forest model requires more computational time and needs more time to build the model. However, in terms of APFD, it outperforms most other models, with the best APFD values on the Paint Control dataset, for example.

Lastly, the reinforcement learning model produces the most stable outcomes in terms of both recall and precision. While its values are slightly lower compared to other models, it maintains consistent recall and precision across all datasets. In terms of APFD, it performs slightly worse than the other models, exhibiting lower and more unstable APFD values. Additionally, this model requires significantly more time to train.

Numerous machine learning models have been explored for test case prioritization, and while some models perform better on specific datasets, no universally optimal model has yet been identified. Further analysis, including datasets with more test cases and more standardized datasets, is needed to achieve a more comprehensive solution.

Nevertheless, this work provides a valuable overview and comparison of the main machine learning techniques used for test case prioritization.

6.2 Future Work

New developments in test case prioritization using reinforcement learning are highly anticipated. This approach has high potential to significantly reduce the testing time by optimizing the order in which test cases are performed. To get clearer and more general results, it would be very useful to have larger and more consistent sets of data. The collection and analysis of such datasets should be the focus of future research in order to allow for more in-depth exploration and more robust findings.

In terms of future work, hyperparameter tuning is another area worth exploring in more depth. By trying out different settings for hyperparameters across different models, researchers could compare how well they worked and find the best settings for specific data sets or testing situations. This method could improve ML models, helping them to be more efficient and adaptable. Also, using Natural Language Processing (NLP) techniques to decide which test cases to prioritize is a new area to research. Test case descriptions often contain important information that could help to decide which cases to focus on first, based on how likely they are to find problems. NLP can help us find and use this information to make test case prioritization better.

7 List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Networks
APFD	Average Percentage of Faults Detected
CI	Continuous Integration
DTs	Decision Trees
e.g.	Example
GUI	Graphical User Interface
GSDTSR	Google Shared Data Set of Testing Suite Results
HMM	Hidden-Markov Model
KNN	K-Nearest Neighbors
LTR	Learning to Rank
ML	Machine Learning
NAPFD	Normalized Average Percentage of Faults Detected
NLP	Natural Language Processing
RPA	Rank Prevention Average
RETECS	Reinforced Test Case Selection
RL	Reinforcement Learning
RTL	Ranking to Learn
SVM	Support Vector Machine
TCP	Test Case Selection and Prioritization
i.e.	id est (that is)
at.al.	et al. (and others)

8 List of figures

9 List of tables

10 References

[1] R. Gopinath, R. ·Ajay, and C. Sanjay, *An introduction to machine learning*. New York NY: Springer Science+Business Media, 2019.

[2] J. Alzubi, A. Nayyar, and A. Kumar, "Machine Learning from Theory to Algorithms: An Overview," *J. Phys.: Conf. Ser.*, vol. 1142, p. 12012, 2018, doi: 10.1088/1742-6596/1142/1/012012.

[3] V. Wittpahl, Ed., *Künstliche Intelligenz: Technologie, Anwendung, Gesellschaft*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019.

[4] C. Atkinson, *ChatGPT and computational-based research: benefits, drawbacks, and machine learning applications.* [Online]. Available: https://www.researchgate.net/publication/376219446_ChatGPT_and_computational-based_research_benefits_drawbacks_and_machine_learning_applications (accessed: Nov. 19 2024).

[5] E. K. Mece, H. Paci, and K. Binjaku, "The Application Of Machine Learning In Test Case Prioritization - A Review," *EJECE*, vol. 4, no. 1, 2020, doi: 10.24018/ejece.2020.4.1.128.

[6] H. Spieker, A. Gotlieb, D. Marijan, and and M. Mossige, *Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration: Conference: ISSTA 2017: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis.* [Online]. Available: https://hspieker.de/files/Spieker_et_al._-_2017_-_Reinforcement_Learning_for_Automatic_Test_Case_Prioritization_and_Selection_in_Continuous_Integration.pdf (accessed: Feb. 10 2024).

[7] P. Ammann and J. Offutt, Eds., *Introduction to Software Testing*: Cambrige University Press, 2017. Accessed: Mar. 12 2025.

[8] S. Abele and P. Göhner, *Improving Proceeding Test Case Prioritization with Learning Software Agents: Conference :ICAART2014-InternationalConferenceonAgentsandArtificialIntelligence.* [Online]. Available: https://www.scitepress.org/papers/2014/49200/49200.pdf (accessed: Feb. 10 2024).

[9] R. Lachmann, S. Schulze, and C. Seidl, *System-Level Test Case Prioritization Using Machine Learning | Request PDF: Conference: 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA).* [Online]. Available: https://www.researchgate.net/publication/313469509_System-Level_Test_Case_Prioritization_Using_Machine_Learning (accessed: Nov. 19 2024).

[10] R. Lachmann, *Machine Learning-Driven Test Case Prioritization Approaches for Black-Box Software Testing: Conference: ettc2018 - European Test and Telemetry Conference.* [Online]. Available: https://www.researchgate.net/publication/347743375_124_-_Machine_Learning-Driven_Test_Case_Prioritization_Approaches_for_Black-Box_Software_Testing (accessed: Feb. 10 2024).

[11] D. Marijan, A. Gotlieb, and M. Liaaen, "A learning algorithm for optimizing continuous integration development and testing practice," *Softw Pract Exp*, vol. 49, no. 2, pp. 192–213, 2019, doi: 10.1002/spe.2661.

[12] A. Lawanna, "An effective test case selection for software testing improvement," in *Computer Science and Engineering Conference (ICSEC), 2015 International*, Chiang Mai, Thailand, 2015, pp. 1–6.

[13] O. Masmoudi, M. Jaoua, A. Jaoua, and S. Yacout, "Data Preparation in Machine Learning for Condition-based Maintenance," *Journal of Computer Science*, vol. 17, no. 6, pp. 525–538, 2021, doi: 10.3844/jcssp.2021.525.538.

[14] I. Goodfellow, Y. Bengio, and A Courville, "Deep learning," (in En;en), *Genet Program Evolvable Mach*, vol. 19, 1-2, 2018, doi: 10.1007/s10710-017-9314-z.

[15] L. Mariani, M. Pezzè, O. Riganelli, and M. Santoro, "AutoBlackTest," in *2011 33rd international conference on software engineering (ICSE 2011): Honolulu, Hawaii, USA 21-28 May 2011*, Waikiki, Honolulu HI USA, 2011, pp. 1013–1015.

[16] O. Ali Sadek Ibrahim and D. Landa-Silva, *ES-Rank: Evolution Strategy Learning to Rank Approach.* [Online]. Available: https://www.researchgate.net/publication/385985292_Study_of_Supervised_Logistic_Regression_Algorithm (accessed: Dec. 10 2024).

[17] A. Kurani, P. Doshi, A. Vakharia, and M. Shah, *A Comprehensive Comparative Study of Artificial Neural Network (ANN) and Support Vector Machines (SVM) on Stock Forecasting.* [Online]. Available: https://link.springer.com/article/10.1007/s40745-021-00344-x (accessed: Mar. 12 2025).

[18] G. Webb, *Naïve Bayes.* [Online]. Available: https://www.researchgate.net/publication/306313918_Naive_Bayes (accessed: Oct. 19 2024).

[19] T. Ahmad, A. Ashraf, D. Truscan, and I. Porres, *Exploratory Performance Testing Using Reinforcement Learning: 2019 – 45th Euromicro Conference on Software*.

[20] S. S. Emam and J. Miller, "Test Case Prioritization Using Extended Digraphs," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, 2015, doi: 10.1145/2789209.

[21] J. Jordan, "Evaluating a machine learning model," *Jeremy Jordan*, 2017, 2017. https://www.jeremyjordan.me/evaluating-a-machine-learning-model/ (accessed: Feb. 10 2024).

[22] T. Shi, L. Xiao, and K. Wu, *Reinforcement Learning Based Test Case Prioritization for Enhancing the Security of Software: Conference: 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA).* [Online]. Available: https://www.researchgate.net/publication/347086385_Reinforcement_Learning_Based_Test_Case_Prioritization_for_Enhancing_the_Security_of_Software (accessed: Dec. 19 2024).

[23] M. Zurek-Mortka, C. K. Chanda, and P. K. Mondal, *Advances in Energy and Control System: Estimation of Prioritization of Test Cases Using Machine Learning Algorithms.* [Online]. Available: https://link.springer.com/book/10.1007/978-981-97-0154-4 (accessed: Jan. 5 2025).

[24] K. Antti, M. Mika, P. Tuula, and K. Mika, *Model-Based Testing Through a GUI: Conference: Formal Approaches to Software Testing, 5th International Workshop, FATES.* [Online]. Available: https://www.researchgate.net/publication/221366262_Model-based_testing_through_a_GUI (accessed: Feb. 20 2025).

[25] A. Bertolino, A. Guerriero, B. Miranda, R. Pietrantuono, and S. Russo, "Learning-to-rank vs ranking-to-learn: Corference: ICSE '20: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering," in *Seoul, South Korea*, pp. 1–12.

[26] A. Swain, K. Swain, and S. K. Swain, Eds., *Meta Heuristic Techniques in Software Engineering and Its Applications: Automated Test Case Prioritization Using Machine Learning; International Conference on Metaheuristics in Software Engineering and its Application*: Springer, Cham, 2022.

[27] J. Thorsten, *Learning to Classify Text Using Support Vector Machines.* [Online]. Available: https://link.springer.com/book/10.1007/978-1-4615-0907-3 (accessed: Mar. 23 2025).

[28] T. Zhu, "Analysis on the Applicability of the Random Forest," *J. Phys.: Conf. Ser.*, vol. 1607, no. 1, p. 12123, 2020, doi: 10.1088/1742-6596/1607/1/012123.

[29] SciKit learn, *User Guide.* [Online]. Available: https://scikit-learn.org/stable/user_guide.html (accessed: Feb. 10 2024).

11 Appendix A – Figures

In this appendix different figures are displayed showing the recall and APFD values of the different models performing on the different datasets in more detail. Figure 24 to Figure 41 showing the recall and the percentage of test classified as failing. Figure 42 to Figure 57 are showing the APFD values.

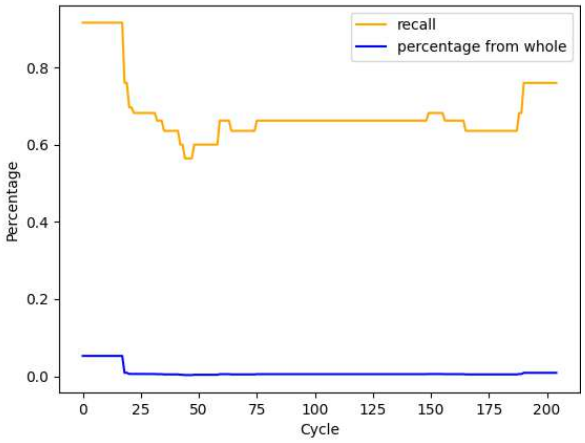

Figure 24: Fail Rate on GSDTSR

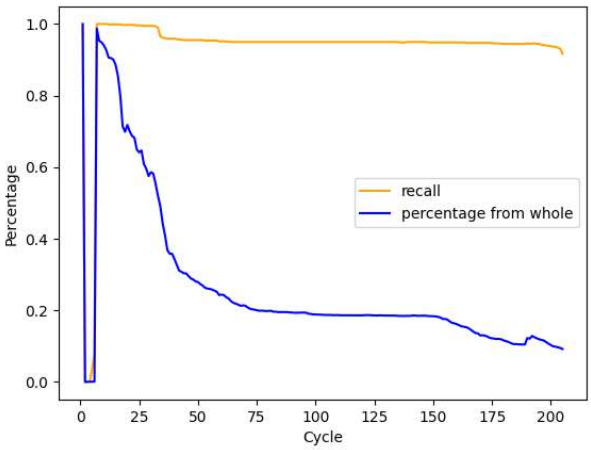

Figure 25: Naïve Bayes on GSDTSR

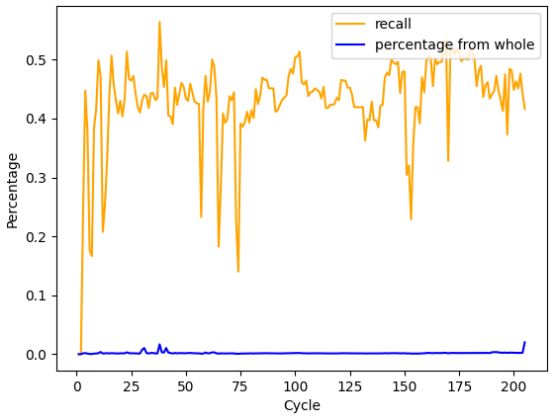

Figure 26: Random Forest on GSDTSR

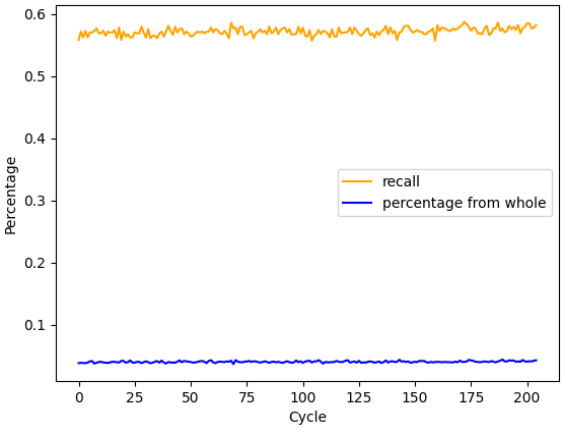

Figure 27: Reinforcement Learning on GSDTSR

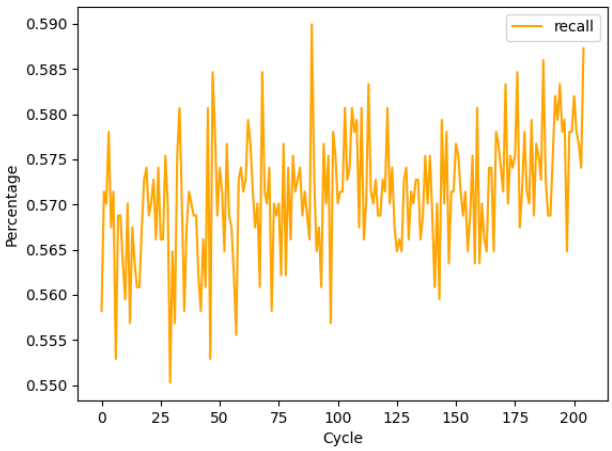

Figure 28:Recall of Reinforcement Learning on GSDTSR

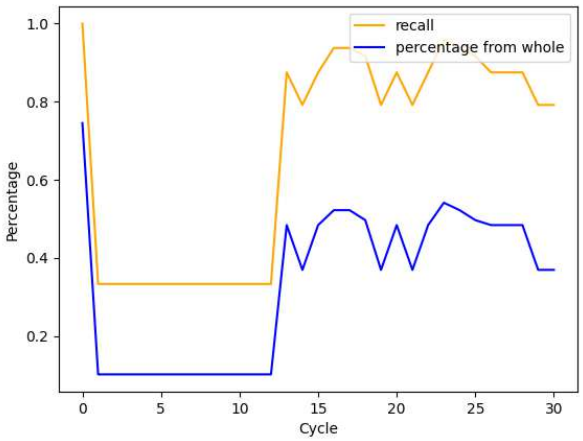

Figure 29: Fail Rate on Bosch

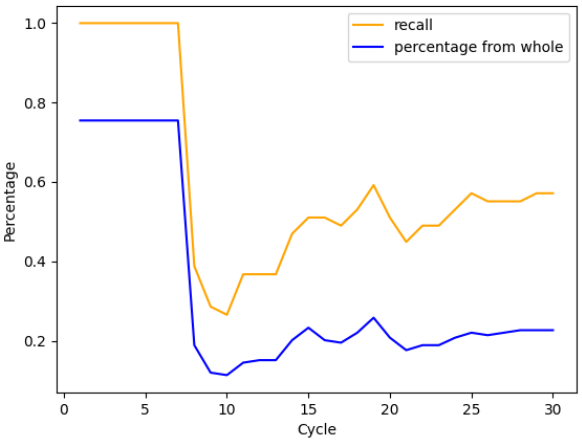

Figure 30: Naïve Bayes on Bosch

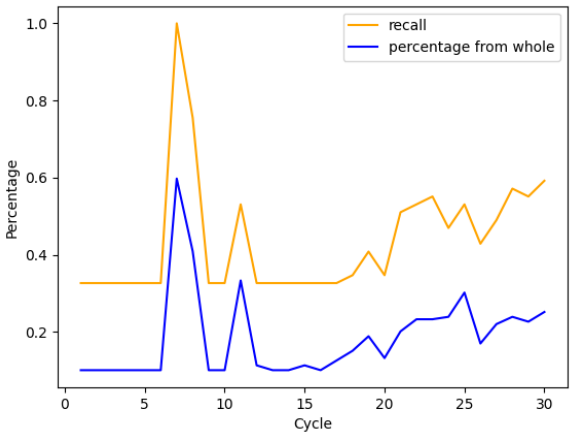

Figure 31: Random Forest on Bosch

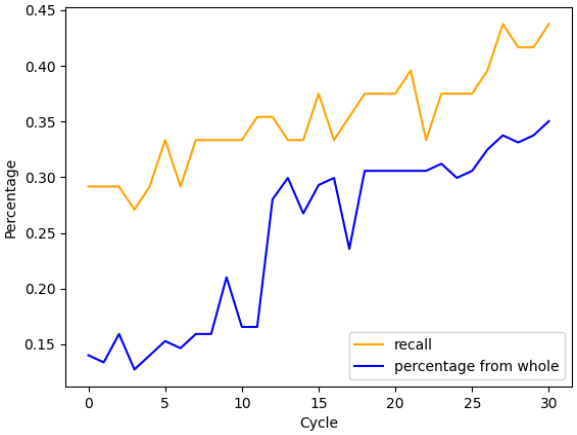

Figure 32: Reinforcement Learning on Bosch

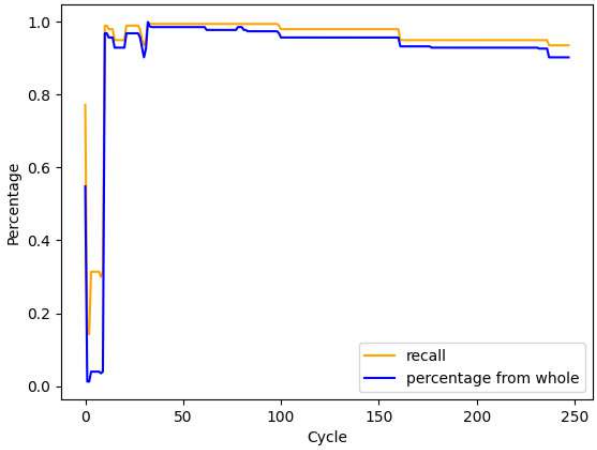

Figure 33: Fail Rate on Paint Control

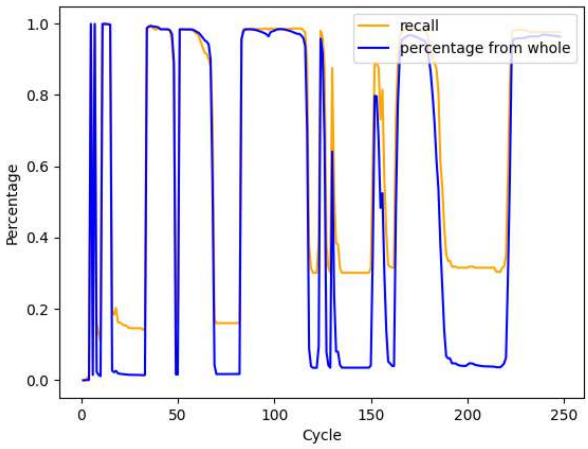

Figure 34: Naïve Bayes on Paint Control

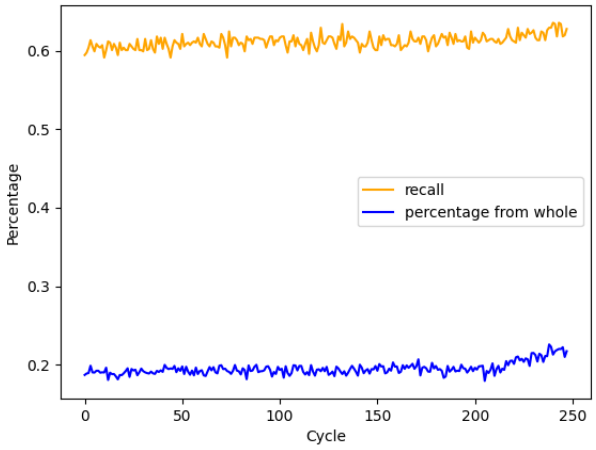

Figure 35: Random Forest on Paint Control

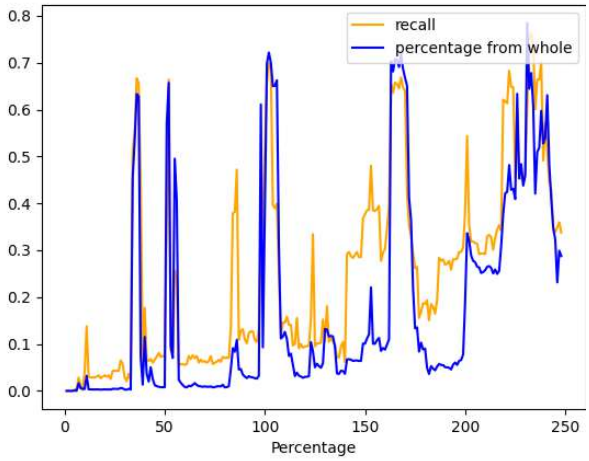

Figure 36: Reinforcement Learning on Paint Control

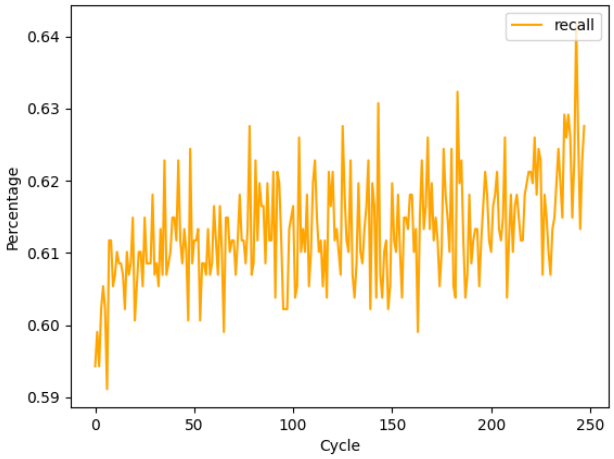

Figure 37:Recall of Reinforcement Learning on Paint Control

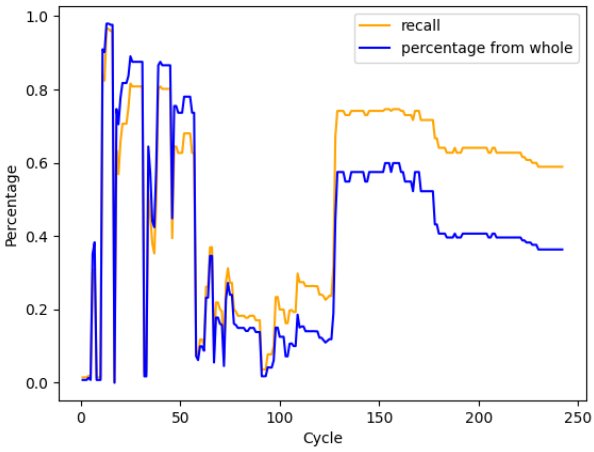

Figure 38: Naïve Bayes on IOF/ROL

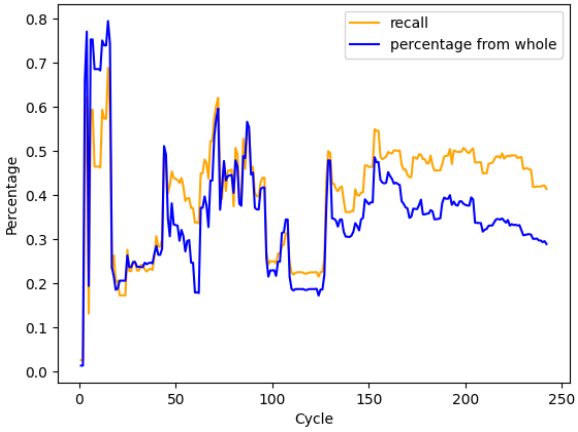

Figure 39: Random Forest on IOF/ROL

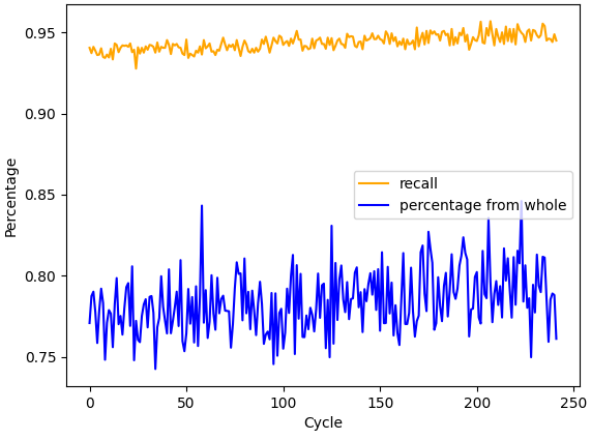

Figure 40: Reinforcement Learning on IOF/ROL

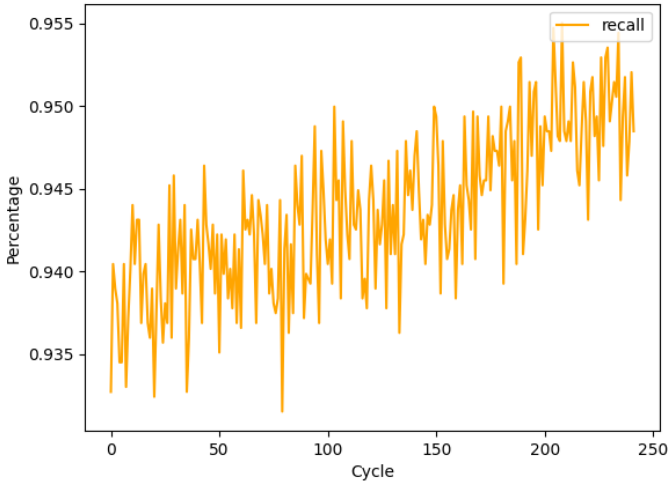

Figure 41:Recall of Reinforcement Learning on IOF/ROL

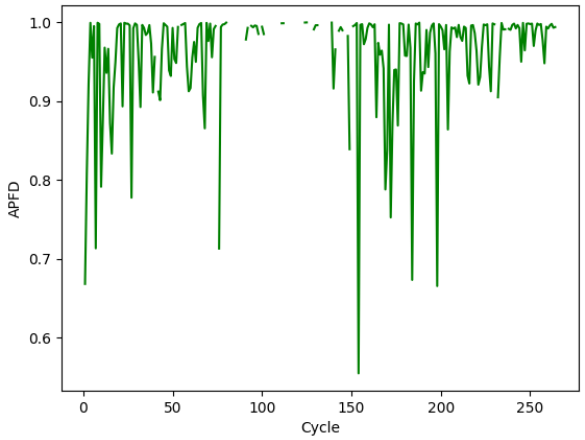

Figure 42: APFD Fail Rate on GSDTR

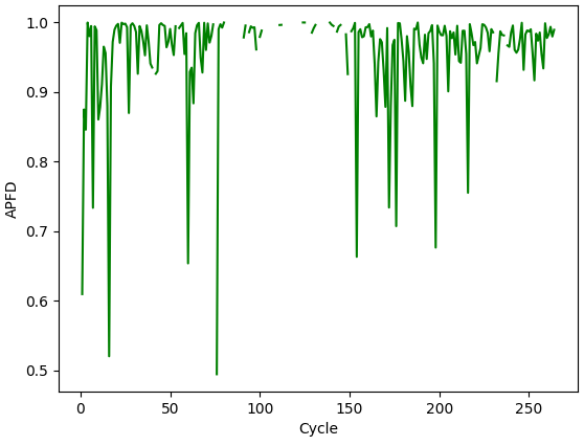

Figure 43: APFD Naïve Bayes on GSDTR

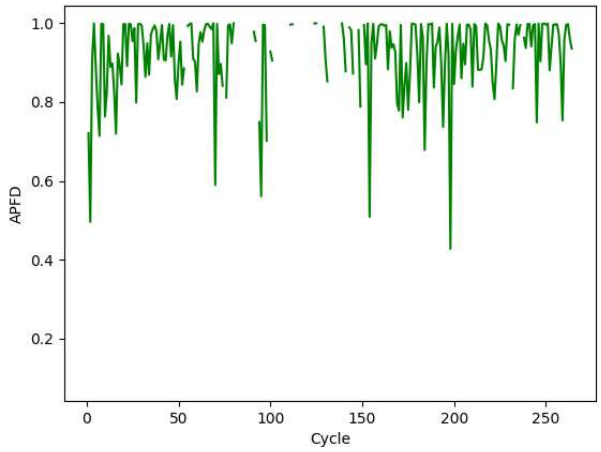

Figure 44: APFD Random Forest on GSDTR

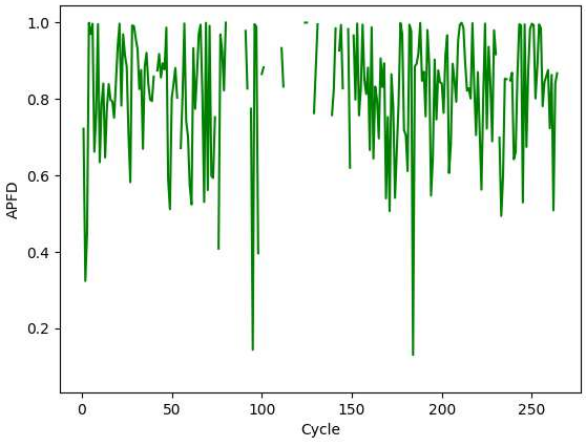

Figure 45: APFD Reinforcement Learning on GSDTR

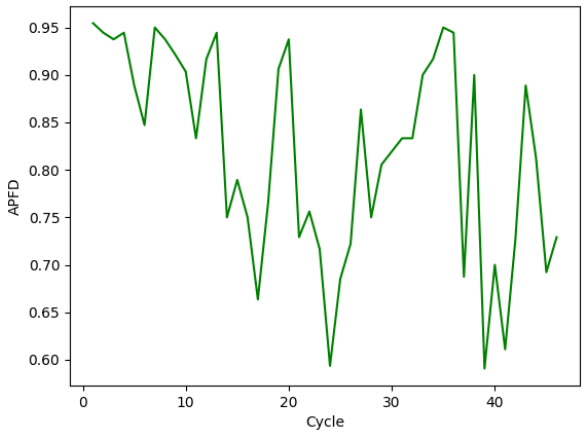

Figure 46: APFD Fail Rate on Bosch

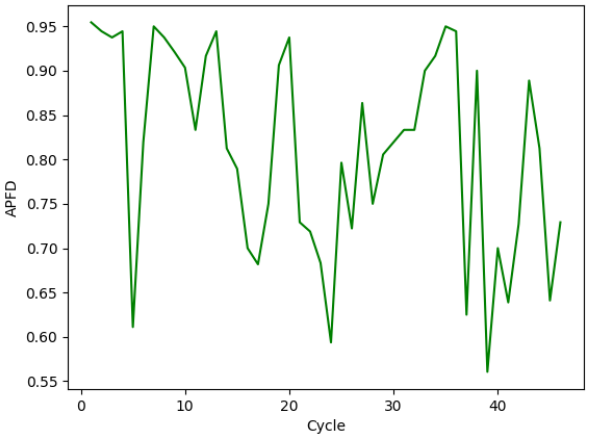

Figure 47: APFD Naïve Bayes on Bosch

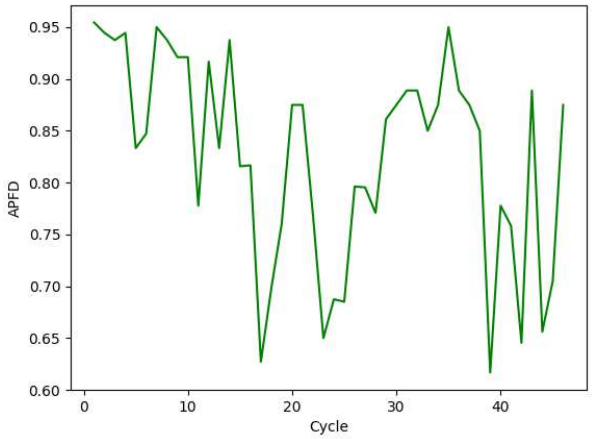

Figure 48: APFD Random Forest on Bosch

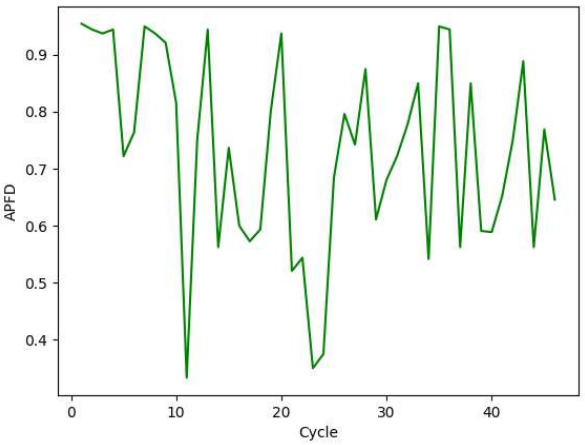

Figure 49: APFD Reinforcement Learning on Bosch

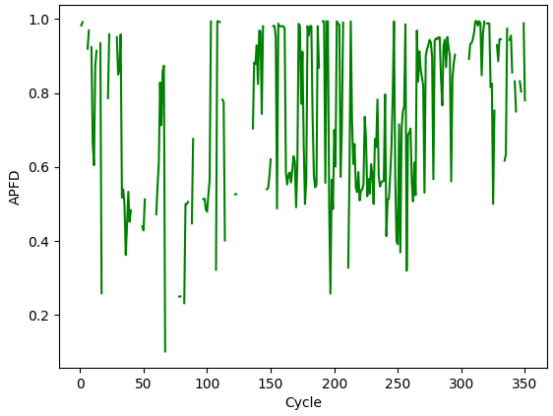

Figure 50: APFD Fail Rate on Paint Control

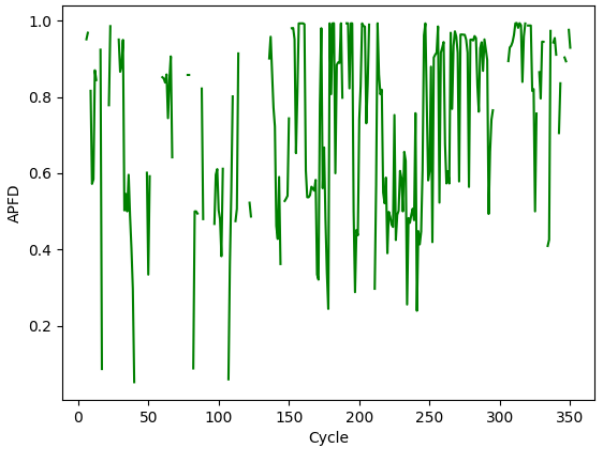

Figure 51: APFD Naïve Bayes on Paint Control

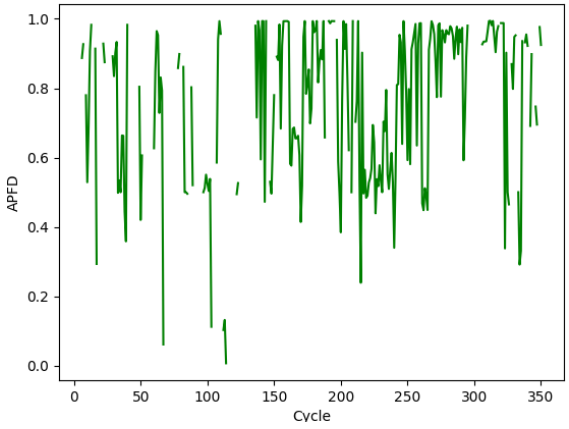

Figure 52: APFD Random Forest on Paint Control

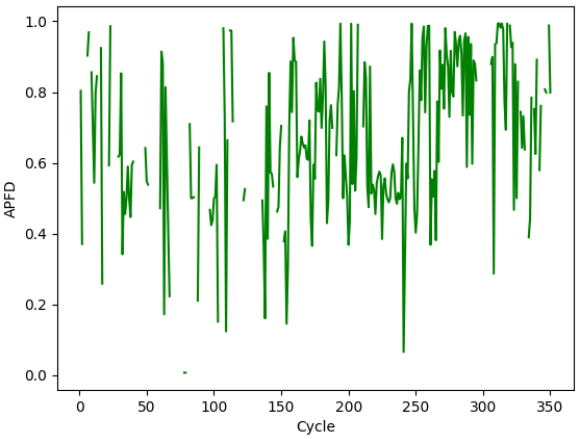

Figure 53: APFD Reinforcement Learning on Paint Control

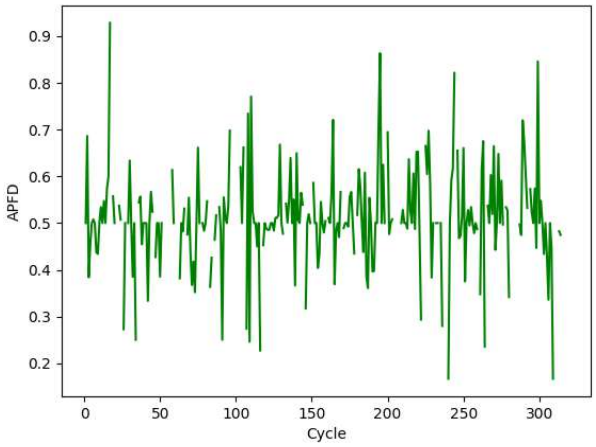

Figure 54: APFD Fail Rate on IOF/ROL

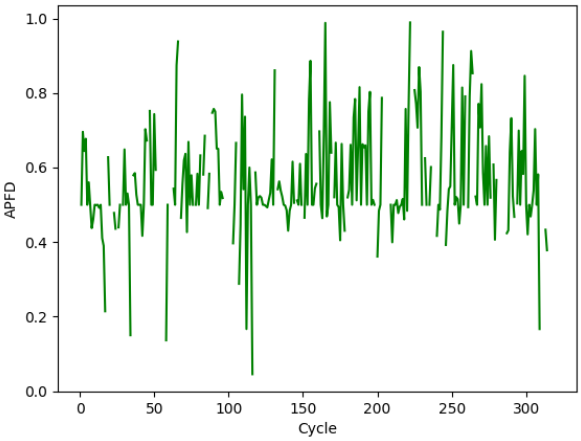

Figure 55: APFD Naïve Bayes on IOF/ROL

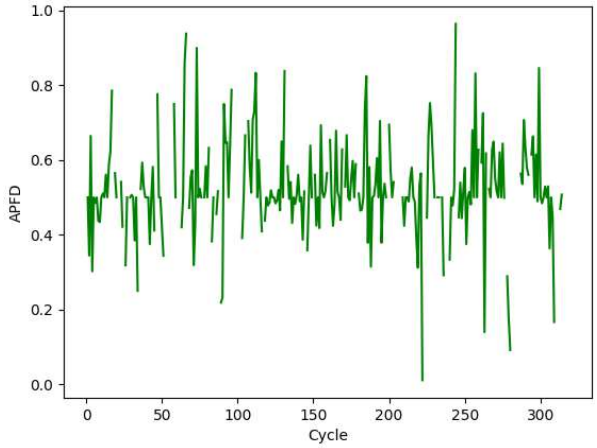

Figure 56: APFD Random Forest on IOF/ROL

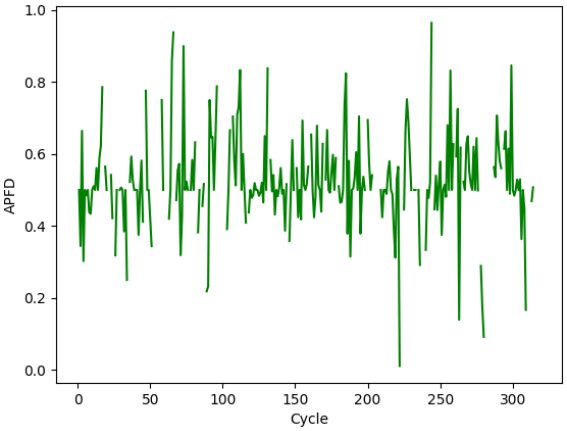

Figure 57: APFD Reinforcement Learning on IOF/ROL