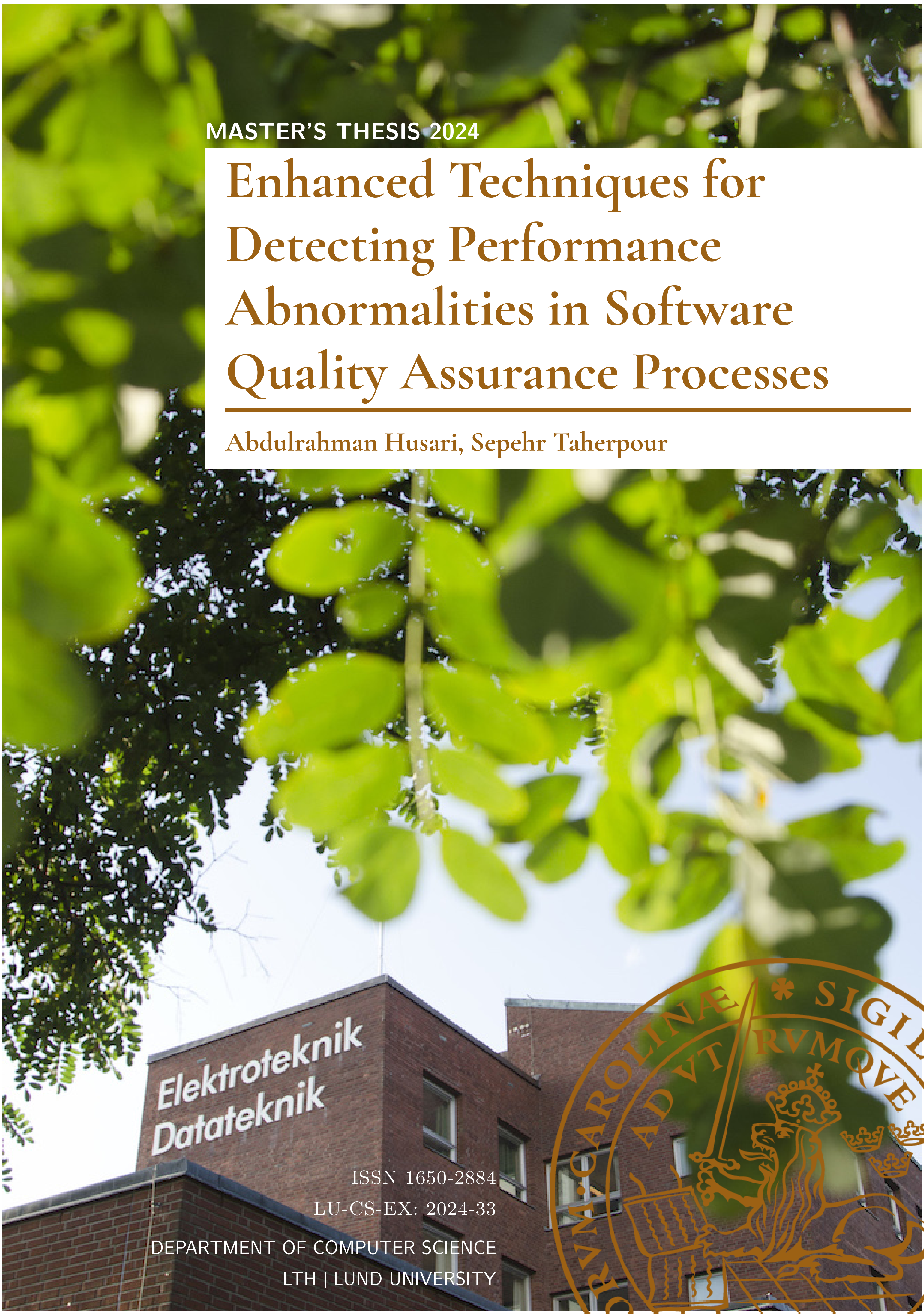
MASTER'S THESIS 2024

Enhanced Techniques for Detecting Performance Abnormalities in Software Quality Assurance Processes

Abdulrahman Husari, Sepehr Taherpour



ISSN 1650-2884
LU-CS-EX: 2024-33

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-33

Enhanced Techniques for Detecting Performance Abnormalities in Software Quality Assurance Processes

Förbättrade tekniker för att upptäcka prestandaavvikelser i processer för mjukvarukvalitetssäkring

Abdulrahman Husari, Sepehr Taherpour

Enhanced Techniques for Detecting Performance Abnormalities in Software Quality Assurance Processes

Abdulrahman Husari
abdulrahman.husari@outlook.com

Sepehr Taherpour
sepehr.taherpour@outlook.com

June 19, 2024

Master's thesis work carried out at Axis Communications AB.

Supervisors: Filip Nilsson, filip.nilsson@axis.com
Jacob Lindquist, jacob.lindquist@axis.com
Per Runeson, per.runeson@cs.lth.se

Examiner: Björn Regnell, bjorn.regnell@cs.lth.se

Abstract

Analyzing performance test data for network camera devices presents significant challenges due to the vast amount of data generated. The lack of efficient methods for detecting performance abnormalities necessitates studying this area. The analysis often requires manual work by quality engineers, which becomes increasingly complicated with the expansion of test variety and product variants, creating a bottleneck in the development and Quality Assurance (QA) processes. This study aimed to develop effective and adaptive techniques for detecting abnormal behavior in performance test results.

We investigated approaches based on statistical thresholds, unsupervised learning, and supervised learning. In addition, we implemented a feedback mechanism used for improving the detection over time. The proposed solution was designed to assist quality engineers and reduce their workload. The efficiency of the solution was evaluated using machine learning metrics and validated through live testing sessions. The results indicated that supervised learning was the most suitable approach in this study, with the RandomForest model achieving an average AUC score of 0.94 and an average recall value of 0.98 across three selected products. The evaluation of the proposed solution demonstrated the effectiveness of the detection in the QA process at the case company.

Keywords: Performance, Data Analysis, Quality Assurance (QA), Software Testing, Machine Learning (ML), Anomaly, Regression, Automation

Acknowledgments

We would like to thank Per Runeson for his guidance throughout the study and his supervision in the formation of this report. We would also like to thank Filip Nilsson, Jacob Lindquist, and Antonios Skourtis for their supervision during this study at Axis Communications AB, as well as Konstantin Malysh and Willie Betschart for their input and tips on how to improve our machine learning solution. Special thanks to Björn Regnell for examining this study. Lastly, we are grateful to friends and family for their support during this journey.

Contents

Chapter 1
Introduction

The ever-growing technological landscape and expanding product portfolios present significant challenges in analyzing the vast amounts of data generated from software tests. Efficiently identifying potential issues in the test data is particularly critical for industries where product quality and reliability are paramount. In software engineering, a robust Quality Assurance (QA) processes is essential to deliver high-quality products while maintaining efficient development cycles. Continuous changes require frequent monitoring and testing of various aspects, including performance. However, analyzing the data from performance tests becomes a bottleneck when dealing with a wide range of products and use cases.

Evaluating hardware resource utilization and service quality is crucial during software performance testing. Performance testing helps determine whether new software meets the defined criteria for a successful release. Analyzing performance test results presents a resource-intensive challenge, as it often requires manual work. This manual burden creates bottlenecks in both the software development and the QA processes.

This master's thesis project, conducted at Axis Communications AB, one of the leaders in the network camera solutions, seeks to address the named bottlenecks. Axis encounters the challenge of efficiently analyzing the performance data generated from its various camera products. This project proposes a solution based on a machine learning approach to streamline the analysis process. In this study, developing a user-centric solution that integrates seamlessly into existing workflows and offers ease of use is considered important. To ensure effectiveness, the proposed solution is evaluated in terms of both technical performance and practical use in real-world scenarios.

1.1 Problem Summary

Analyzing performance data to identify defects or abnormalities has been done mainly manually. Quality engineers at Axis have had to laboriously review graphs generated from performance test results, seeking to pinpoint problematic changes in product behavior. This manual approach is not only time-consuming but also becomes increasingly complicated as the variety of tests and product expands. Previous efforts by Axis to automate this process by defining static thresholds have not yielded the expected levels of efficacy, leaving room for significant improvement. Furthermore, these efforts have not been documented for reuse, making it difficult to build on this previous work.

1.2 Project Goals & Research Questions

This project aims to explore various methods for detecting performance abnormalities through analyzing performance test results and to develop a contextually appropriate solution based on similar case studies. The primary goal is to enhance the QA processes of the case company by creating a solution that efficiently identifies and highlights relevant cases for further review by quality engineers. By reducing the workload of quality engineers, the solution is expected to improve the timeliness and accuracy of the analysis, thus enhancing the overall QA processes. Positioned at the intersection of theoretical exploration and practical application, this study aims to provide valuable insights into the broader field of performance data analysis.

Research Questions

This study aims to answer the following questions in the context presented:

RQ1 How can efficient detection of performance abnormality be done through analyzing performance test results?

RQ2 In what ways can the proposed solution assist and affect current QA processes?

RQ3 How can the proposed solution be adaptable for evolving needs?

1.3 Proposed Solution

In the pursuit of solutions to the addressed problem, relevant case studies were reviewed, and the knowledge gained was applied to the specific context of this problem and finally evaluated and validated. Generally, incorporating machine learning has proven to be an effective approach for solving the presented problem. Since the problem requires automated data analysis and abnormality detection, it is similar to anomaly detection in machine learning. Due to the variability of the data and the evolving quality expectations, the proposed solution needed to be able to adapt in order to overcome these circumstances. This adaptability is achieved by incorporating feedback from quality engineers to continually refine its efficacy and accuracy.

1.4 Case Company Background

Axis Communications AB, one of the global leaders in network video surveillance, was established in 1984 in Sweden. Initially focusing on protocol converters, the company shifted its attention to networked video products in the mid-1990s. The groundbreaking introduction of the world's first network camera in 1996 marked a transformative moment for the surveillance industry, propelling the transition from analog to digital video surveillance. Axis offers high-resolution video monitoring and intelligent features, such as video analytics. Their product line includes network cameras, video encoders, video management systems, audio solutions, and access control systems.

With offices in more than 50 countries and partnerships in 179 nations, Axis maintains a vast global network. This extensive reach enables the company to deliver customized solutions to various sectors and meet global security requirements[1].

As a technology leader, Axis must maintain high-quality products in an industry characterized by rapid technological changes. Their wide range of products requires ongoing and rigorous testing, making automated testing and innovative quality assurance methods crucial for maintaining high standards. However, the increasing volume and complexity of data can make it challenging for quality engineers. This master's thesis project focuses on addressing these challenges at Axis Communications.

[1] `https://www.axis.com/about-axis/history`

Chapter 2
Method

This study is conducted according to the design science paradigm (DSP) [1]. The overall research method and activities are presented in this chapter.

2.1 Design Science Paradigm

The Design Science Paradigm (DSP) is a framework designed to address real-world problems and validate solutions using scientific literature. This framework often involves studying case studies to acquire knowledge and define a solution [1].

Design science aims to understand and improve human-made designs in practical contexts. The activities associated with this method include problem conceptualization, solution design, and validation, all linked to real-world scenarios. Contributions with a theoretical foundation are tailored to specific contexts, and recommendations are formed as 'technological rules.' These rules are practical solutions derived from general knowledge about problems and solutions, which are then validated in the context of the study [1].

DSP focuses on addressing real practical problems rather than adhering to specific methods. To propose solutions, an iterative process is undertaken between problem conceptualization and solution design. The proposed solutions are empirically validated to assess their feasibility, contributing to the general knowledge base and broadening the scope of validation in subsequent studies [1].

According to Figure 2.1, the DSP consists of two main dimensions: practice-theory and problem-solution. Research activities are links between constructs and instances, aiming to find solutions to defined problems and validate them. The research process begins in the lower-left corner of Figure 2.1 with a problem instance. Problem conceptualization forms a problem construct, which is then used in solution design. This design can then be implemented, instantiated, forming a solution instance targeted for validation.

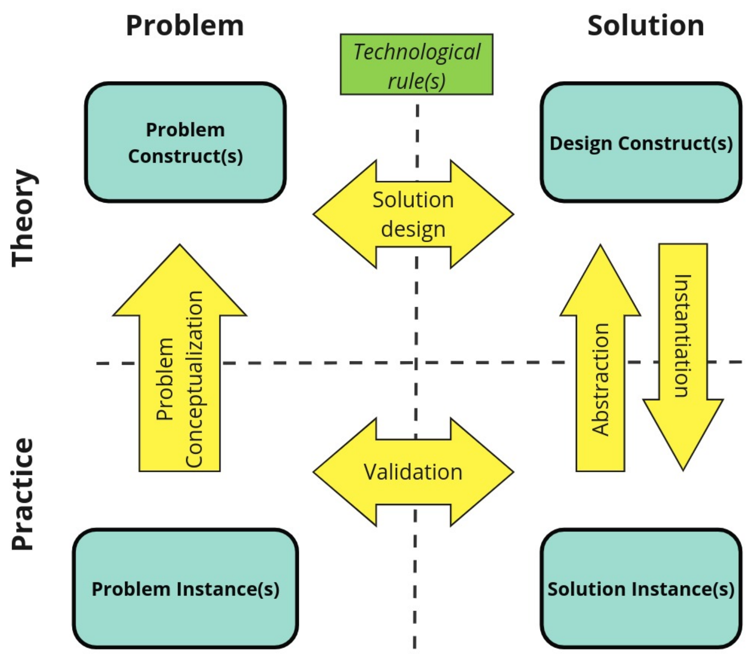


Figure 2.1: Design science paradigm, according to Runeson et al. [1]

2.2 Research Activities

This section outlines the research activities undertaken to address the identified problem. Our research follows a structured approach, including a thorough literature review, interviews, and a systematic process of problem identification, conceptualization, solution design, instantiation, abstraction, and empirical validation, as described by Runeson et al. [1]. Each step is designed to ensure an effective solution. Below is a detailed description of each research activity:

Literature review

The literature review provides the foundation for understanding current approaches that offer potential solutions to the identified problem. Our study followed established methodologies for a comprehensive literature review as described by Thiel [2]. The review process involved several steps:

- **Keyword Search**: Keywords such as "performance regression," "software performance," "anomaly detection," and "machine learning" guided our search in academic databases.
- **Filtering Papers**: We filtered publications based on criteria like publication date, peer-reviewed status, and relevance to performance testing.
- **Review of Abstracts**: Abstracts were reviewed to identify innovative techniques or frameworks relevant to our project.
- **Full Paper Review**: Promising papers were thoroughly read to analyze methodologies and findings.

- **Critical Analysis and Organization**: Key insights were organized into thematic areas to identify trends, gaps, and opportunities, providing a conceptual basis for our solution design.

This review ensures that our proposed solution is grounded in existing research and best practices, while also highlighting gaps that allow us to propose a tailored solution for the case company. The result of this review is detailed in Section 3.6.

Interviews

Interviews are crucial for gaining first-hand insights into challenges and practices present in the case company. The primary goals for us are to understand Axis' performance issue management and improvement aspirations to define the project requirements. Key focus areas include the quality assurance processes, performance data analysis workflows, data analysis criteria, and the challenges faced.

We used a semi-structured format [3], combining planned questions with spontaneous exploration. This format allows the interviewer to adapt questions based on the interviewee's responses. The interview structure followed a systematic approach [4], starting with broad open-ended questions to establish context, then narrowing to specific topics, and finally opening up again for a comprehensive discussion. During sessions, note taking was done to be able to use the information for further analysis and presentation.

Identifying Problem & Problem Conceptualization

During problem identification, the problem is studied and analyzed to gather information about real-world challenges. This involves methods such as interviews, observations, and surveys to capture detailed insights into specific issues faced by practitioners. Problem conceptualization bridges empirical findings with theoretical constructs, allowing for a structured understanding of the issues. It marks the transition from practice to theory by forming a structured problem statement.

In our study, problem identification was carried out through interviews and problem analysis, as detailed in Chapter 4. Through interviews and discussions with the quality assurance team, we mapped out the current process, identified its challenges, and established a clear problem conceptualization.

An extensive literature review and consultations with domain experts were conducted to frame the problem of automating detection. This activity aligns with the "Problem Conceptualization" quadrant in Figure 2.1.

Problem Construct & Solution Design

The problem construct involves defining the problem in general terms within the theoretical domain. Solution design refers to the activity of formulating one or more potential solutions to the defined problem. At this stage, alternative solutions, related research, and case studies are considered.

In this study, problem construct and solution design were achieved by reviewing related literature and theories connected to the problem, as presented in Chapter 3. By studying relevant terms and literature, we developed a sketch of potential solutions to the defined problem.

Instantiation

Instantiation refers to implementing the designed solutions in a real-world context to create tangible artifacts. These artifacts, also known as Solution Instances, result from instantiation. This process involves translating theoretical designs into practical implementations, which can then be validated. In our study, the instantiation and the resulting solution instance are presented in Chapter 5.

Abstraction

Abstraction involves generalizing validated findings into broader technological rules that can be applied across different contexts. This step transforms specific, contextual knowledge into generalized principles or guidelines. In our study, we formulate technological rules based on validated solutions, providing a design that is adaptable to similar problems in other settings. Abstraction involves moving from validated practical solutions to theoretical generalizations. This is provided in the context of this study in Chapter 7.

Empirical validation

Validation is the empirical assessment of the effectiveness of the solution in a real-world context. This involves systematic testing and feedback collection to evaluate whether the solution meets the intended goals and resolves the identified problem. In our study, we conducted evaluations using machine learning methods and user feedback. This process is detailed in Chapter 6, with the objective of ensuring that the implemented solution effectively addresses the identified problem, highlighting both the strengths and the areas for improvement.

2.3 Boundaries & Limitations

This project utilizes performance data from Axis network camera devices, focusing on key performance metrics such as video frame rate, bitrate, available memory, and CPU usage. The proposed solution specifically targets performance regressions in these devices, with a feedback mechanism designed for engineers familiar with camera technology.

Reliance on historical data and specific lab configurations may limit the solution's effectiveness for new or significantly different devices and environments. The project's time frame and available resources constrain the depth of analysis and the extent of prototype development and testing. To manage scope, we focused on three products using distinct chipsets, meaning the study is limited to data from these selected products. Furthermore, the focus on machine learning assumes the availability of high-quality data. Any gaps or errors in the data could impact the solution's effectiveness, underscoring the importance of robust data management and preprocessing.

Chapter 3
Background & Related Work

This chapter provides the background and terminology essential to this study. Further, the related work section summarizes relevant literature.

3.1 Software Engineering and Performance

This section covers terms and topics related to software engineering and testing, focusing on performance and performance measurements. The aim is to provide an understanding of these concepts and establish a foundation for further discussions in this work.

Software development encompasses various activities that together form a software solution. This is usually done through different stages which can be also called a lifecycle. According to the International Software Testing Qualifications Board (ISTQB), software development lifecycle is defined as *"the activities performed at each stage in software development, and how they relate to each other logically and chronologically"*[1].

In this study, testing refers specifically to software testing, which ISTQB defines as *"the process within the software development lifecycle that evaluates the quality of a component or system and related work products"*[2]. Software testing encompasses various types, each with its own objectives and practices. Of particular relevance to this study is performance testing, which will be elaborated upon in the following.

3.1.1 Performance Efficiency

In this study, the term *performance* refers specifically to performance efficiency, which ISTQB defines as *"the degree to which a component or system uses time, resources, and capacity when ac-*

[1] `https://glossary.istqb.org/en_US/term/software-development-lifecycle`
[2] `https://glossary.istqb.org/en_US/term/testing-9`

complishing its designated functions"[3]. This definition is based on ISO 25010[4], a standard issued by the International Organization for Standardization (ISO) that defines a product quality model for Information and Communication Technology (ICT) and software products. The model includes nine characteristics, each divided into sub-characteristics regarding the quality properties of the products. Performance efficiency is one of these characteristics and is divided into three sub-characteristics:

- **Time behavior:** The capability of a product to perform its specified function under specified conditions so that the response time and throughput rates meet the requirements.
- **Resource utilization:** The capability of a product to use no more than the specified amount of resources to perform its function under specified conditions.
- **Capacity:** The capability of a product to meet requirements for the maximum limits of a product parameter.

To determine performance efficiency, various parameters are measured, such as hardware resource utilization. Hardware resource could be CPU, memory, storage, etc. Throughput and capacity/bandwidth are specially important parameters for network devices. Additionally, energy consumption and material usage can be measured as well. The measurement part leads us to the next term, performance testing.

3.1.2 Performance Testing

Performance testing is defined by ISTQB as *"a type of testing to determine the performance efficiency of a component or system"* [5]. Essentially, it involves examining the performance efficiency characteristic mentioned above through various measurements. The tools used for this purpose are known as *performance testing tools* [6], defined as *"test tools that generate load for a designated test item and measure and record its performance during test execution"*. These tools form the performance testing framework employed by the case company in this study. The results produced by these tools constitute the data analyzed in this study.

3.1.3 Regression Testing

Regression testing is defined by ISTQB [7] as *"a type of change-related testing to detect whether defects have been introduced or uncovered in unchanged areas of the software"*. In other words, it assesses how new changes affect other components of the software, ensuring that previously existing functionalities continue to work as expected. The case company in this study uses this testing method to ensure the changed software meets the requirements.

[3]`https://glossary.istqb.org/en_US/term/performance-efficiency-2-2`
[4]`https://www.iso.org/obp/ui/#iso:std:iso-iec:25010`
[5]`https://glossary.istqb.org/en_US/term/performance-testing-2-2`
[6]`https://glossary.istqb.org/en_US/term/performance-testing-tool`
[7]`https://glossary.istqb.org/en_US/term/regression-testing-1-3`

3.1.4 Performance Regression Testing

Performance regression Testing (PRT) is directly related to the case studied as it the results of these tests which forms the data analyzed in this study. PRT is a subset of regression testing that focuses on verifying performance criteria after any changes. As part of the software development life cycle, performance regression testing (PRT) is crucial to ensure that new integrations and features do not degrade existing functionality and performance. By identifying and resolving performance issues before release, PRT aims to improve product quality and reduce the need for post-release fixes.

To determine if there has been a regression or improvement, measurements of key performance metrics based on the system's use case are compared to expected values. Examples of performance metrics include response time, duration, throughput, stability, and hardware resource usage. The process involves establishing a baseline performance from previous software versions and continuously monitoring new releases under varying conditions to identify any deviations from this baseline. The results of new tests are compared with the established baseline, requiring analysis to identify and address unwanted or unexpected deviations. Performance monitoring and analysis tools can provide insights into performance bottlenecks, helping to ensure optimal system performance.

3.2 Data Analysis

Data analysis is the process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, informing conclusions, and supporting decision-making[8]. As this study works with data, terms and methods related to analyzing this data are presented.

Correlation

Correlation analysis measures the relationship between two quantitative variables[9]. For example, in testing network cameras, understanding the correlation between frame rate and bitrate under normal operating conditions helps to establish expected performance benchmarks. A significant shift in this relationship can alert engineers to hidden issues, such as software inefficiencies or unexpected hardware behavior, which may not be immediately evident through other testing methods.

Additionally, if performance metrics analysis indicates a strong correlation between two or more metrics across different test cases and conditions for various products, it suggests that a shift in one metric may lead to a corresponding shift in the correlated metrics. This understanding is highly useful, as it allows engineers to monitor a single metric instead of multiple ones, simplifying the detection process and potentially speeding up troubleshooting.

[8]https://en.wikipedia.org/wiki/Data_analysis

[9]https://en.wikipedia.org/wiki/Correlation

Interquartile Range (IQR)

IQR is defined as the difference between the 75th and 25th percentiles of the data and is used to measure statistical dispersion. This method is particularly effective for identifying outliers, as it is robust against extreme values that could skew the analysis. In practice, data points falling outside a specified multiple (X) of the IQR from the upper or lower quartile are flagged as outliers, indicating potential operational irregularities or misconfigurations [10].

Z-Score

Z-score, standard score, is a statistical measure that describes the relative position of a value within a data set by expressing its distance from the mean in terms of standard deviations [11]. This method helps identify outliers by quantifying how many standard deviations a data point is from the mean, making it useful for detecting anomalies in normally distributed data.

The z-score, denoted by z, is calculated using the following formula: $z = \frac{x-\mu}{\sigma}$ where:

z, x, μ, σ represent respectively the z-score of the data point, the value of the data point, the population mean and the population standard deviation.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a type of data analysis that aims to summarize, visualize, and understand the characteristics of a dataset. This step helps analysts gain insights into the data beyond traditional hypothesis testing by uncovering patterns, outliers, and underlying structures. EDA employs various techniques such as visual methods, including histograms, box plots, and scatter plots, to provide a quick, intuitive understanding of distributions and relationships. Statistical summaries, including measures such as mean, median, and standard deviation, describe the central tendency and variability of the data. Correlation analysis and dimensional reduction techniques, such as Principal Component Analysis (PCA), can also be used as tools[12].

3.3 Performance Anomaly Detection

In the context of our study, anomaly detection refers to identifying unusual patterns or deviations in performance data from network camera solutions that differ from expected behavior, the established baseline. These anomalies may indicate potential defects, performance regressions, or other issues that could negatively impact the quality and functionality of the products. Effective detection in this context is crucial as it facilitates quicker troubleshooting, enhances product quality and customer satisfaction.

Static Thresholds involve setting predefined limits on specific metrics. Any data point out of the limits is considered an anomaly. This method is straightforward and effective in simpler contexts. Static thresholds can be used for metrics that are expected to remain

[10] `https://en.wikipedia.org/wiki/Interquartile_range`

[11] `https://www.machinelearningplus.com/machine-learning/how-to-detect-outliers-with-z-score/`

[12] `https://www.ibm.com/topics/exploratory-data-analysis`

within a narrow range during normal operations. Any deviation out of these limits is then immediately flagged as an outlier requiring investigation.

Statistical Thresholds, unlike static thresholds, are defined according to the characteristics and patterns in the data. These characteristics can be identified through data analysis and statistical methods, as discussed in Section 3.2. The insights gained from this analysis can either directly inform threshold definitions or provide a foundation for more nuanced threshold setting. This method is more adaptive and can be suitable for metrics that naturally fluctuate or for environments where test conditions or performance criteria may change over time[13].

3.4 Machine learning

Machine learning (ML) is a subfield of Artificial Intelligence (AI) that involves algorithms that learn from data without explicit programming. This learning process includes identifying patterns and relationships within the data, enabling the model to develop a "rule set". There are three main machine learning approaches: supervised, unsupervised, and reinforcement learning. Once trained, the model can make predictions on unseen data points.

Within the domain of performance regression testing, ML techniques can be leveraged to analyze historical performance metrics. By training on past measurement data across different metrics, the model can learn the system's typical performance behavior. This established baseline allows the ML model to identify anomalies in subsequent test results, potentially uncovering performance regressions that might be missed by other means.

Classification problem in machine learning is to predict the category of which any new data point belongs to based on past observations. Binary classification is a specific type of classification where there are only two possible classes or labels. For instance, an abnormalities detection system that labels test executions as either *anomaly (1)* or *normal (0)* is an example of binary classification [14].

3.4.1 Unsupervised Learning

Unsupervised learning is a type of machine learning that involves training algorithms on unlabeled data, focusing on identifying patterns, structures, or relationships within the data itself. These methods are particularly valuable in environments where defining explicit rules or labels is impractical due to the complexity or evolving nature of the data [15]. By analyzing data without predefined categories, unsupervised learning can uncover patterns, clusters, or outliers that indicate normal or abnormal behavior. Unsupervised learning can be categorized into different types or groups including:

Clustering

Clustering algorithms group objects so that those in the same cluster are more similar to each other than to those in other clusters. This is useful in anomaly detection, as anomalies often

[13]`https://www.sinch.com/blog/dynamic-threshold-estimation-for-anomaly-detection/`
[14]`https://en.wikipedia.org/wiki/Binary_classification`
[15]`https://www.ibm.com/blog/anomaly-detection-machine-learning/`

form smaller, less dense clusters than normal data.

K-means partitions data points into K predefined clusters based on similarities. It iteratively assigns points to the nearest cluster centroid and recalculates centroids. While efficient for grouping similar data, K-means struggles with outliers, potentially leading to missed anomalies despite high overall accuracy [15].

Density-Based Spatial Clustering (DBSCAN) identifies clusters of varying shapes and sizes by examining point density. It does not require predefined clusters and classifies points with sufficient neighbors as core points forming clusters, while isolated points are classified as noise (potential anomalies). This makes DBSCAN useful for detecting anomalies in data with varying densities [16].

Local Outlier Factor (LOF) measures the local deviation of density of a given data point with respect to its neighbors. It is effective in detecting anomalies that may not fit into the global distribution but are outliers in their local contexts [17].

Decision-tree

Decision-tree algorithms split data into branches at the decision nodes, where each node represents a test on an attribute, and each branch represents the result of the test.

Unlike clustering techniques, the **Isolation Forest** algorithm isolates anomalies instead of profiling normal data points. It randomly selects features and split values, isolating anomalies that tend to have shorter paths in the tree structure because of their rarity and distinctiveness. This makes Isolation Forest efficient for detecting anomalies [15].

Copula-Based

Copula-based algorithms use copulas to describe the dependence between random variables, joining multivariate distribution functions to their one-dimensional marginal distributions.

Copula-Based Outlier Detection (COPOD) utilizes copulas to model dependencies between variables, effectively detecting outliers in cases where relationships between performance metrics are non-linear or not normally distributed [18].

Histogram-based

Histogram-based algorithms analyze data distributions by constructing histograms, which are bar charts representing the frequency distribution of variables.

Histogram-based Outlier Score (HBOS) assumes independence between features and calculates anomaly scores by building histograms. It is faster than multivariate models, making it suitable for high-dimensional datasets, and can quickly highlight unusual distributions in performance metrics[19].

[16]`https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html`

[17]`https://en.wikipedia.org/wiki/Local_outlier_factor`

[18]`https://paperswithcode.com/paper/copod-copula-based-outlier-detection`

[19]`https://www.dfki.de/fileadmin/user_upload/import/6431_HBOS-poster.pdf`

Support Vector Machines

Those are a popular machine learning model used primarily for classification and regression tasks. They work by finding the hyperplane that best separates different classes in the data with the maximum margin, thereby minimizing classification errors. SVMs can also be extended for use in unsupervised learning, particularly for anomaly detection through the One-Class Support Vector Machine (OC-SVM).

OC-SVM fits a decision function around normal data to detect outliers. It is useful for learning boundaries around normal data points, effectively distinguishing them from anomalies [15].

Neural network

Deep learning and neural networks are fundamental concepts in ML, distinguished primarily by their complexity. Neural networks consist of interconnected nodes, inspired by biological neurons, that transmit and process information. These networks can learn and adapt based on the data on which they are trained.

Autoencoder is a type of neural network used to learn efficient codings of unlabeled data. The network is trained to ignore "noise" by learning to reconstruct the most important features from the input data. In anomaly detection, autoencoders can accurately reconstruct typical data but will fail to do so for anomalies, making them useful for detecting unusual data patterns [20].

3.4.2 Supervised learning

Supervised learning is a type of machine learning in which the model is trained using labeled data. In this approach, each input data point is paired with an output label. The model learns the relationship between input and output during training, which allows it to predict output labels for new unseen data based on the learned relationships [15].

Support Vector Machines (SVMs) are classifiers that work by finding the hyperplane that best separates different classes in the feature space with the maximum margin. This makes them particularly effective in high-dimensional spaces. SVMs are robust against overfitting, especially in cases where the number of dimensions exceeds the number of samples [21].

RandomForest is an ensemble learning method based on multiple decision trees, providing high accuracy and robustness by averaging multiple deep decision trees trained on different parts of the same training set. Random Forest performs well on large datasets and can handle thousands of input variables without variable deletion, providing estimates of what variables are important in the classification [22].

ExtraTrees stand for Extremely Randomized Trees, this algorithm builds multiple trees like Random Forest but with random splits of all observations and features rather than searching for the best split among a random subset of the features. This method typically yields even more diversified trees than Random Forest and can result in better performance [5].

[20]`https://www.ibm.com/topics/autoencoder?mhsrc=ibmsearch_a&mhq=Autoencoder`

[21]`https://scikit-learn.org/stable/modules/svm.html`

[22]`https://www.ibm.com/topics/random-forest`

K-Nearest Neighbors (KNN) is a non-parametric method used for classification. In KNN, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small) [15].

Synthetic Minority Oversampling Technique (SMOTE) presented by Chawla et al. [6], addresses unbalanced data sets which are a common challenge in machine learning. When one class (often the interesting or anomalous one in our case) has significantly fewer examples than others, models can become biased towards the majority class. SMOTE tackles this issue by creating synthetic data points for the minority class. It analyzes existing minority class examples and their nearest neighbors in the feature space, generating new data points along the lines connecting them. This helps balance the data set and potentially improves the performance of the model for the minority class [23].

Semi-supervised Learning

Semi-supervised learning is a machine learning approach that combines a small amount of labeled data with a large amount of unlabeled data during training. Positioned between unsupervised learning (which uses no labeled data) and supervised learning (which only uses labeled data), this method is particularly valuable when acquiring a fully labeled dataset is prohibitively costly or labor-intensive[15].

[23]Analytics Vidhya - SMOTE

3.5 Machine Learning Evaluation

To assess the performance of different ML models, several metrics are used based on the outcomes classified into a confusion matrix. These metrics help to understand the effectiveness of the model in predicting different classes.

Confusion Matrix is a table used to describe the performance of a classification model on a set of test data for which the true values are known. It allows visualization of the model's predictions, including true positives, true negatives, false positives, and false negatives, providing insight into the types of errors made by the model. In the context of anomaly detection, if an anomaly case corresponds to a 1 and a normal case to a 0, a positive prediction corresponds to detecting the case as an anomaly [24]. The metrics can be described as follows:

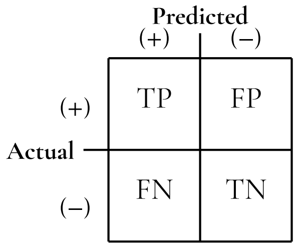


Figure 3.1: Confusion Matrix

- **True Positive (TP):** Correctly identified as a positive case. (An abnormal case correctly predicted as anomaly).
- **True Negative (TN):** Correctly identified as a negative case. (A normal case correctly predicted as normal).
- **False Negative (FN):** Incorrectly identified as a negative case. (An abnormal case is missed and predicted as normal mistakenly).
- **False Positive (FP):** Incorrectly identified as a positive case. (A normal case predicted as anomaly mistakenly).

Recall (Sensitivity) measures the ability of a model to find all the relevant cases within a dataset. High recall indicates that the model is good at detecting the positive class [24].

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Precision reflects the accuracy of the positive predictions made by the model. High precision indicates that a higher percentage of positive identifications was actually correct [24].

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

F1-Score is the harmonic mean of precision and recall. It is a way of combining both precision and recall into a single measure that captures both properties. It is particularly useful when the class distribution is uneven. High F1 score means that the model has low false

[24]`https://www.ibm.com/topics/confusion-matrix`

positives and low false negatives, so it's correctly identifying real threats and not disturbing users with false alarms [24].

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy is the most intuitive performance measure and it is the overall ability of the model to correctly identify both positive and negative outcomes [24].

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Observations}}$$

False Positive Rate (FPR) measures the proportion of actual negatives that are incorrectly identified as positives by the model. A lower FPR indicates fewer instances of normal cases being misclassified as anomalies [24].

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

False Negative Rate (FNR) measures the proportion of actual positives that are incorrectly identified as negatives by the model. A lower FNR indicates fewer instances of anomalies being missed by the model [24].

$$\text{FNR} = \frac{\text{False Negatives}}{\text{False Negatives} + \text{True Positives}}$$

Receiver Operating Characteristic (ROC) is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The area under the ROC curve **(AUC)** is a measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve[25].

Cross Validation is an essential technique in machine learning for assessing a model's generalizability and mitigating overfitting by partitioning the dataset into multiple subsets. When data is limited, methods like K-Fold Cross Validation are employed, where the dataset is divided into K parts, and the model is trained and validated K times, each time using a different part as the validation set [26]. This technique ensures each data point is used for both training and validation. By averaging the performance across all folds, cross-validation provides a more robust estimate of how well the model is likely to perform on unseen data [7].

[25]`https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc`

[26]`https://www.coursera.org/articles/what-is-cross-validation-in-machine-learning`

3.6 Related Work

Literature relevant to this study is presented in this section.

3.6.1 Software Performance Testing

The goal of software performance testing is to verify that the software under test meets performance expectations or requirements. Many quality metrics and attributes, such as response time and resource utilization, can be used for this purpose, depending on the type of product and the testing context [8, 9].

Performance testing is crucial for many companies before releasing a product to the market or to customers. It helps to identify issues that cannot be detected during lower-level functional testing. The selection of test cases and goals is critical in performance testing and varies based on the company's specific needs. Common performance parameters measured include throughput, response time, number of tasks, bandwidth, and hardware resource usage (such as CPU, disk, and memory). These parameters are compared with the expected or required values in each case [10].

Patel and Gulati [11] propose a method to identify critical factors affecting software performance and determine ideal ranges for those factors. They analyze historical data to find correlations between different performance measures, using statistical techniques like linear regression and confidence intervals to create a performance measurement matrix. This matrix helps developers identify potential performance problems early in the development process, ensuring that the software meets expected performance benchmarks.

3.6.2 Detection of Software Performance Regression

Fagerström et al. [12] investigated the challenges of automating performance regression test analysis in continuous integration and deployment. Their qualitative case study at Ericsson highlighted the difficulties in deriving verdicts from non-functional testing due to unclear goals, insufficient requirements, and unreliable data. They suggested improvements such as standardization, a testing map, and intelligent tools that can leverage historical data to enhance efficiency and accuracy in analysis.

A study by Daly et al. [13] examined the detection of software performance regressions in a continuous integration system at MongoDB. This study introduced a new approach based on change point detection, moving away from manual graphs monitoring and threshold-based detection. The primary data consisted of time/speed measurements for performance tests, which included noise complicating automation and monitoring. The change-point detection algorithm provides a list of problematic cases to a reviewer, who then selects true regressions from the suggestions. The new approach positively impacted the process, significantly reducing the number of false positive cases.

Building on the previous work, Fleming et al. [14] focused on enhancing the change point detection algorithm and developed a tool named, *Hunter*. This tool aimed to eliminate the need for dedicated engineers to review performance test results by providing an intuitive and user-friendly interface suitable for non-experts. Hunter was evaluated against two well-known algorithms on time series data with injected regressions and demonstrated superior performance.

Bauer et al. [15] proposed an automated method for classifying and detecting change points in performance test data using machine learning. Their approach involved training supervised models on pre-labeled change points, achieving a 94.3% accuracy in distinguishing true from false change points. Additionally, a real-time detection method using a windowing technique achieved an AUC of 92.0%. This work underscores the potential of machine learning to automate change point detection, thereby reducing the need for manual analysis.

Nguyen et al. [16] explored the challenges in analyzing performance test results to identify the causes of performance regressions. Their method used historical performance data and hardware utilization measurements (e.g., CPU usage) to detect regression causes with up to 80% accuracy without requiring extensive historical data. They employed statistical methods and machine learning, along with synthetic problem injection, to bootstrap the approach, demonstrating its effectiveness on both open-source and commercial projects.

Liao et al. [17], in conjunction with [18], addressed challenges similar to those identified in other studies. They proposed using black-box machine learning models, which require no knowledge of the system's internal behavior, to detect performance regressions. These models were trained on historical and regressions in new performance data were identified even under varying workloads. The method was effective in both open-source and large industrial systems, proving to be a viable alternative to traditional performance testing, especially in resource-constrained environments.

3.6.3 Anomaly Detection in Similar Context

In a study by Mendes et al. [19], two different unsupervised machine learning algorithms were deployed to distinguish anomalous energy usage patterns in hotel units, leveraging a comprehensive sensor network to monitor electricity, water, and gas consumption, along with vital environmental parameters. This research underscored the significance of Exploratory Data Analysis (EDA) as a foundational step before using ML algorithms in anomaly detection, and showed that Isolation Forest (IF) performs better in their case.

Mejri et al. [20] explored unsupervised anomaly detection methods for time-series data, evaluating not only precision and recall but also model size, stability, and efficacy across various anomaly types. Their analysis covered five datasets and included both machine learning (ML) and deep learning (DL) methodologies, with DL demonstrating superior performance in standard metrics. This study highlighted the importance of considering multiple aspects when assessing the effectiveness of anomaly detection models.

Vávra et al. [21] proposed an adaptive ML-based anomaly detection system tailored for industrial control environments. The study pointed out the limitations of traditional methods and emphasized the potential of both unsupervised and supervised learning approaches to identify abnormal operations. The system utilized historical data to learn patterns and flag deviations as anomalies, stressing the importance of data preprocessing and feature selection for optimal performance.

Shahid et al. [22] investigated deep learning unsupervised models for anomaly detection in school buildings' electricity and district heating consumption. They compared various models that capture sequential data and effective for general data representation. The study found that the Long Short-Term Memory Variational Autoencoder (LSTM-VAE) model achieved superior results in capturing normal consumption patterns. LSTMs are a type of Recurrent

Neural Network (RNN) that particularly are effective in tasks requiring understanding the entire context, highlighting their suitability for this type of anomaly detection [27].

Aronsson [23] proposed an unsupervised machine learning tool for detecting anomalies in multivariate data collected over time in industrial settings. The solution achieved high accuracy in identifying anomalies. The study implemented specific preprocessing techniques and used particular metrics to assess model performance, which proved especially useful in dealing with datasets where anomalies are rare.

A study by Jernbäcker [24] evaluated various unsupervised learning models for anomaly detection in time-series data. This research compared models adept at handling sequential information with those suitable for general data representation. The findings suggested that combining different types of unsupervised models might be necessary to capture the wide range of anomalies in time-series data, as different models have strengths that make them suitable for real-time anomaly detection.

Finally, Zhao et al. [25] presented the Scalable Unsupervised Outlier Detection (SUOD) framework, which addresses the scalability challenges of machine learning by employing a modular and parallelizable architecture. SUOD integrates multiple detection algorithms to enhance robustness, utilizes distributed computing for efficient processing of large datasets, and offers flexibility through its modular design. This framework ensures efficient and effective outlier detection even as data size and dimensionality increase, making it a valuable tool for practical applications.

3.6.4 Incorporating Feedback in Detection

Improving the accuracy and relevance of detected anomalies has been the focus of several studies. Das et al. [26] focused on incorporating user feedback during the anomaly detection process to decrease the number of false positive cases. They modified IsolationForest algorithm by integrating binary feedback, which re-weighted the detection based on user input. This modification significantly enhanced the algorithm's performance, with some instances showing double the accuracy compared to the original version.

Vercruyssen et al. [27] proposed a framework combining unsupervised and supervised learning techniques for monitoring water usage in supermarkets. Initially, the framework operates in an unsupervised mode, organizing data through clustering methods. As human experts provide feedback, the system transitions to a semi-supervised mode, refining its detection capabilities by spreading labels throughout the data. This approach is particularly effective in scenarios where normal behaviors are diverse and less frequent than anomalies, addressing a common challenge in anomaly detection.

Building on previous work [26, 27], Zha et al. [28] explored an alternative method to improve the ranking of detected anomalies using reinforcement learning. Instead of relying on re-weighting or greedy algorithms, this method trains a policy for selecting the detected anomalies. The proposed approach demonstrated superior performance in benchmark settings compared to baseline methods, highlighting the potential of reinforcement learning in enhancing anomaly detection processes.

An example of integrating feedback into a performance monitoring system is presented by Hrusto et al. [29]. This study presented an example of integrating feedback into a per-

[27]`https://developer.ibm.com/tutorials/iot-deep-learning-anomaly-detection-1/`

formance monitoring system to tackle alert fatigue experienced by developers. The study proposed an approach to filter excessive alerts and improve the feedback loop between developers and the deployment process. By using unsupervised anomaly detection together with feedback-based filter, the system created a more sophisticated mechanism that considered the context and learned from past experiences, thereby reducing unnecessary alerts and improving developer efficiency.

Building on their previous work, Hrusto et al. in [30] proposed an approach for anomaly detection in microservices. Microservices are an architectural style that structures applications as a collection of loosely coupled services [28]. This study focused on a microservice-based back-end system for public transportation ticket and payment management. The proposed approach used unsupervised deep learning to analyze multivariate time series data. Upon detecting anomalies, developer feedback was collected to evaluate the model and identify the optimal method. Initially employing unsupervised learning, the study planned to explore supervised learning models trained on feedback labels as a future improvement, aiming to bridge the gap between monitoring performance metrics (operations) and detecting issues (development) in a DevOps context.

[28]`https://microservices.io/`

Chapter 4
Case Description

In this chapter, the results of the interviews conducted are presented, followed by an overview of the current process, the relevant data, and a detailed case description.

4.1 Problem Conceptualization

As described in Section 2.2, interviews were conducted to gather detailed information and define the problem. Three semi-structured interviews were held with engineers involved in performance testing and regression detection, with the questions listed in Appendix B.1. Additionally, two informal discussions were conducted with the senior test framework developer and the leading test expert. Each discussion was held in person to encourage open conversation and was tailored to the interviewee's role and responsibilities.

Using the results of these interviews and discussions, we were able to understand the challenges and the current workflow. This process also provided an introduction to the existing test cases, available data, and the quality assurance process. This foundational understanding allowed us to further study these aspects and ultimately define the problem in the context of this study. Detailed presentations of these areas are provided in the following sections.

4.1.1 Problem Overview

The interviews identified key performance metrics for further investigation, provided insight into how the participants perceive the problem and helped us to study different aspects of the problem further. This resulted in a problem definition and an initial process towards answering **RQ1**.

All participants found the current process limiting, hindering timely performance issue detection and extending feedback loops in development. This results in longer troubleshooting times, increased resource needs, lower product quality, and extended lead times. The organization aims for a more robust performance testing strategy with increased frequency, scale, and automation. However, challenges include a lack of understanding of significant performance factors and difficulties in establishing criteria for identifying anomalous performance after code changes. Test results can include noise from changes in the lab environment, test framework, tools, or issues regarding product hardware. These uncertainties hinder effective automation and scaling of the testing process.

Technical questions from the interviews revealed different perspectives: developers are concerned about longer feedback loops from quality engineers; quality engineers face high monitoring demands and time constraints; the lead test architect sees resource limitations as a bottleneck for more testing, making the process vulnerable and unsustainable for maintaining high quality measures and future growth.

A summary of these points is presented in Table 4.1, showing important performance metrics, the scope of investigation, level of detail, and desired elements in a potential solution, according to the interview participants.

Table 4.1: Summary of interview results

	Senior Test Framework Developer	**Quality Engineer**	**Lead Test Architect/Expert**
Important metric(s)	Frame rate, bitrate, memory, CPU.	Datasheets as hard requirement when using the default setting. Frame rate, bitrate, CPU, memory	Frame rate, bitrate, memory, CPU.
Relation between metrics	Frame rate - bitrate, chipsets - overall behavior.	Chipset - overall behavior, frame rate - bitrate, CPU - frame rate (when using apps).	(Question not applicable)
Size of historical data	Not so long back, focus on recent.	Selected weekly releases, longer periods second priority.	Months, focus on longer periods.
Level of detail in data	Builds for a developer, focus on weekly firmwares in start.	Compare with previous release, weekly.	On every build.
Desired features/solution	Present anomaly with link to graph, choose feedback and get information about which test and possibly time.	Better & easier visualization, indication on a potential issue, provide firmware version and test case.	Find an optimal degree of alerting and decrease the analytical work burden.

4.1.2 Quality Assurance Process for Performance

The quality assurance process related to performance testing at the case company involves several steps. Measurements from automated performance tests are collected in connection with the development of the camera firmware. The tests are executed using streaming tools within the performance test framework for each product under test, and results are logged in JSON format with timestamps. The JSON files are stored in the case company's artifactory.

Figure 4.1 illustrates an overview of the quality assurance processes for performance testing in connection with development. The development process begins with changes or commits made to the codebase by developers. These changes are integrated into the system and compiled into builds. Selected builds are then tagged as firmware, representing a new revision of the software operating the camera.

Daily performance tests are executed automatically on many different camera products in a lab environment using the latest builds. During these tests, measurements are logged in JSON format. This data includes various performance metrics related to video and device

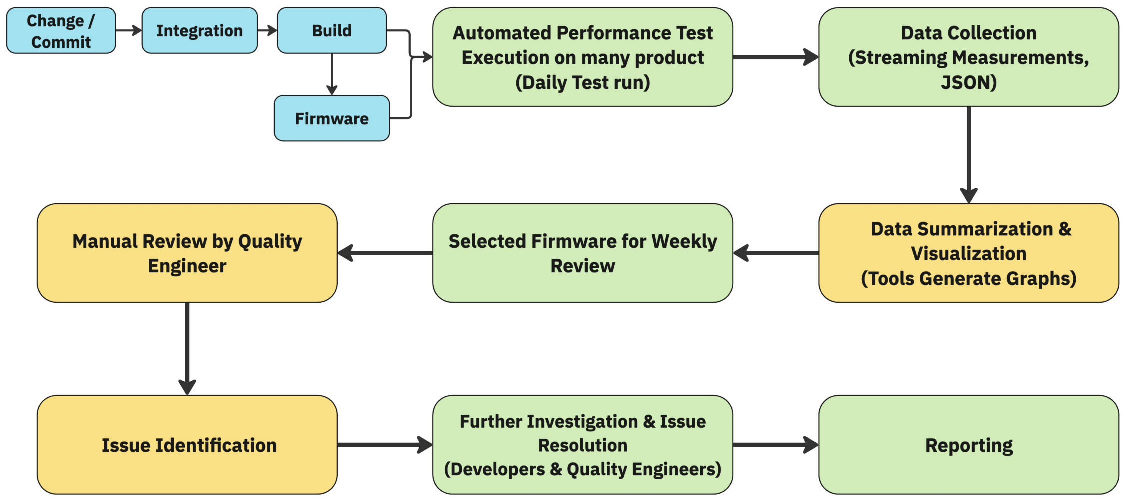

Change / Commit
Integration
Build
Firmware
Automated Performance Test Execution on many product (Daily Test run)
Data Collection (Streaming Measurements, JSON)
Data Summarization & Visualization (Tools Generate Graphs)
Selected Firmware for Weekly Review
Manual Review by Quality Engineer
Issue Identification
Further Investigation & Issue Resolution (Developers & Quality Engineers)
Reporting

Figure 4.1: Quality Assurance Process Related to Performance Test Analysis

performance quality. The collected data is summarized and visualized using tools to generate graphs. These graphs are used by a quality engineer to conduct a manual review of the selected firmware for weekly analysis. The selection criteria for these firmware versions are based on the company's testing and quality assurance strategies.

The manual analysis includes examining the graphs, comparing the current performance test results with the established baseline to identify irregular behaviors, potential defects, and regressions. Any identified issue or notable finding is investigated further by the quality engineer and developers. They work together to identify the cause and resolve the problem, ensuring the new firmware meets the required standards. If a suspected issue is confirmed as a defect at this stage, it is reported to relevant roles for further investigation and resolution.

How is this study related to QA processes? This study specifically focuses on the manual analysis and detection parts, highlighted in orange in Figure 4.1. This includes enabling the engineers to review the results and assist them in a more automated way. By focusing on these stages, the study aims to enhance the effectiveness and improve the overall quality assurance process.

What can be identified as a problem? The company prioritizes delivering video stream quality values, specifically frame rate and bitrate, as stated in the product datasheets. These values serve as performance requirements when monitoring results. However, this applies only to the default setting tests, as stated in Section 4.2. In general, any significant deviation from the established baseline is investigated further. Adverse or unexpected deviations over longer periods are usually considered more serious. However, the level of scrutiny varies depending on the product, the use case, and the performance metric (feature).

Challenges & Motivation

The reliance on manual analysis for performance data creates bottlenecks. As product complexity and the product pool grow, this method becomes unsustainable. Training new staff for this specialized role is hindered by the extensive knowledge required, further limiting scalability for broader quality assurance initiatives. Additionally, the engineer(s) tasked with this analysis often lack the time to thoroughly review data across various features for most products, further exacerbating the issue.

These challenges underscore the need for new approaches to simplify and support the manual analysis process. As mentioned in Section 1.1, previous attempts by the case company to set up a rule-based system with static thresholds for automating analysis have proven to be unsustainable, inflexible, and unscalable. Therefore, this study aims to explore alternative approaches to address these problems effectively.

4.2 Test Cases

All registered test results are related to a test case which in turn is related to a product. The test cases are based on the use case and specifications of the product. In addition, test cases can be defined to address specific quality issues, allowing focused monitoring of those concerns. In this context, the performance tests are primarily video streaming tests. These involve sending requests with specified settings to the camera under test, monitoring the response stream provided by the camera, and recording values for various video, network, and hardware parameters.

The tests can be divided into three main categories: default setting tests, single-channel tests, and multiple-channel tests. A channel refers to an individual logical path for video data within the camera, and it can handle one or multiple streams of video. Tests can include various settings and conditions which are presented in the following.

- **Default Setting Tests**: This is the most basic use case for each camera, defined as a test on one channel and one stream with no other special settings.
- **Single Channel Tests**: These tests evaluate the camera's performance when operating a single channel with one or multiple streams over this channel.
- **Multiple Channel Tests**: These tests assess the camera's performance when handling multiple channels with one or multiple streams over each channel.

All tests, other than default setting tests, may include special conditions. These conditions or settings can involve streaming through various protocols, containers, or formats, additional features such as audio, and different applications such as video analytics. For example, single-channel tests can incorporate multiple streams to evaluate the camera's performance while handling multiple streams alongside more advanced features. Multiple-channel tests involve multiple streams and can include scenarios like group view, quad stream, mixed encoders, and varying view areas with different stream counts. These subcategories represent the test setting combinations depending on the camera's functionalities and use cases.

4.3 Available Data

The available data consists of data points from test measurements recorded during daily test executions according to defined test cases, stored in JSON format. These measurements include values for various performance metrics such as video frame rate, bitrate, bandwidth, and hardware resource utilization on CPU and memory.

Figure 4.2 illustrates the structure of the available data in a tree view. The figure demonstrates various data configurations based on different components in a potential test case. The levels in the tree are interconnected, forming various settings for a test case.

Starting from the top of the tree in Figure 4.2, products built on specific chipset generations operate on particular firmware (build). Performance test cases are first categorized according to channel settings (single or multiple channels). Test cases can then include various options, such as different codecs, protocols, and applications, as shown under *With options*. Each execution of a test case generates detailed measurements for the features presented in the bottom of the figure. Measurements on these features form the test data used in this project.

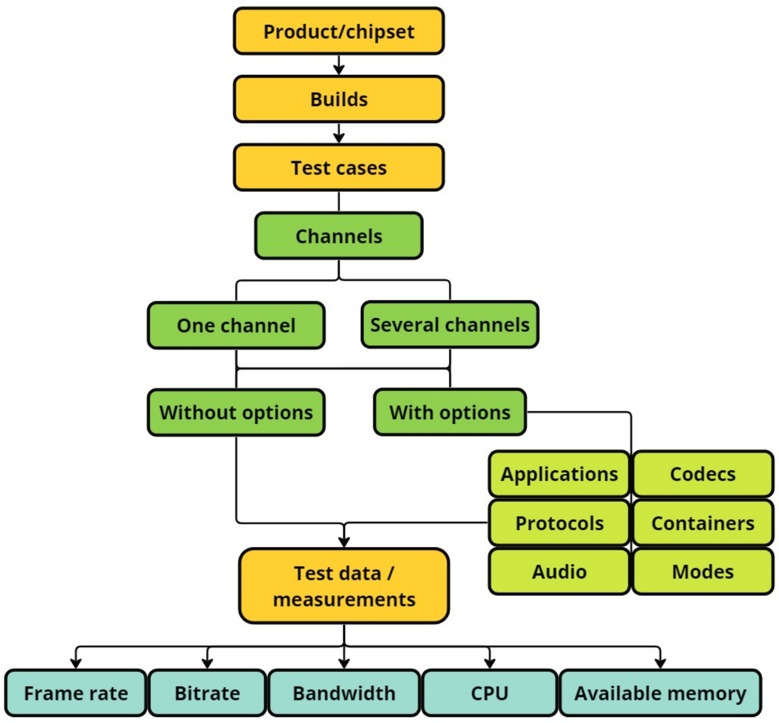


Figure 4.2: The structure of available data regarding each product

Regarding measurements, the sampling occurs every other second during each execution regarding each test case, resulting in approximately 30 measurement points for each feature, assuming an execution lasts about one minute. In other words, a test execution corresponds to 30 data points on each feature multiplied by the number of test cases for a product. Products and test cases may be added or removed over time, influenced by changes in the company's test plan and testing priorities. As mentioned in Subsection 4.1.2, the data are then translated into graphs to be reviewed.

4.4 Alerting Cases

Defining regressions or anomalies is challenging, especially in the studied case. Performance changes can depend on various factors, and understanding these factors is necessary to determine if the change is a true regression. To evaluate different methods for detecting regressions, we define a **regression** or an **anomaly** as *a measurement or behavior that deviates from the established baseline*. Essentially, the goal is to detect abnormal values or trends in the data and alert about them.

Alerting cases in this problem can be divided into two main categories: local and global. The following descriptions and examples illustrate these categories. Note that Figure 4.3 and Figure 4.4 do not depict real data but are intended to demonstrate what anomalies might look like in summarized data. These mock figures have been cleared of noise and do not cover the natural variations within the test cases.

Local Case

A local case refers to a single measurement point of a feature that deviates from the baseline behavior. This deviation could be caused by changes in the system or external factors affecting the measured parameter. An example of a local case is shown in Figure 4.3, which displays a deviating value at a specific timestamp. In this example, the value drops to an abnormal level, assuming that the baseline behavior for this particular test case and feature has been a stable value at 30.

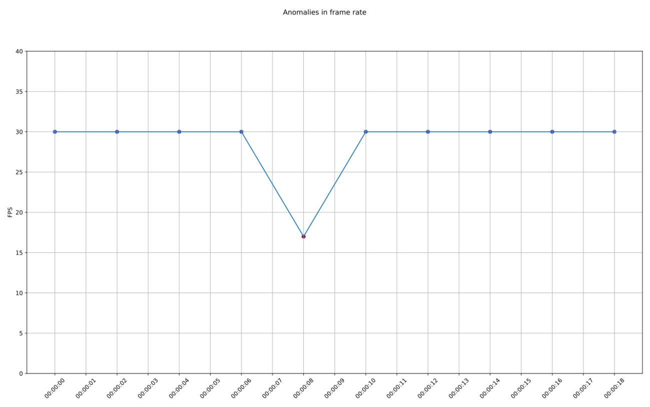
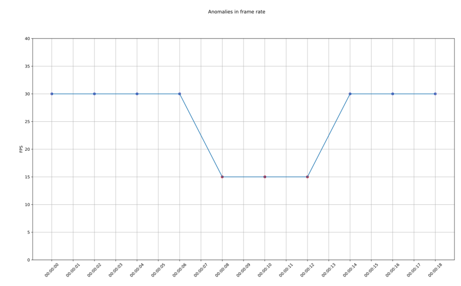


Figure 4.3: An abnormal value

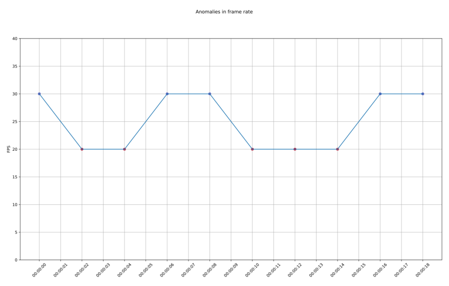
Global Case

A global case can manifest itself in the following patterns, assuming that the corresponding baseline does not contain such a pattern:

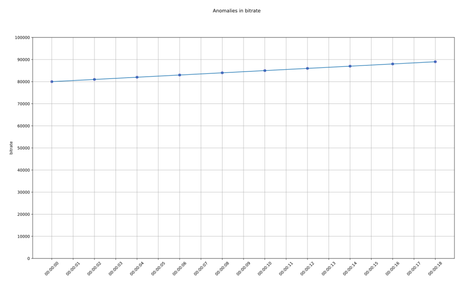
1. Several consecutive measurements deviating from the baseline, suggesting a persistent problem, as shown in Figure 4.4a.
2. A repeated pattern of abnormal values, for example, in a periodic or cyclical manner that deviates from the baseline, as shown in Figure 4.4b.
3. A changed trend or behavior over time compared to the baseline, as seen in Figure 4.4c and Figure 4.4d.



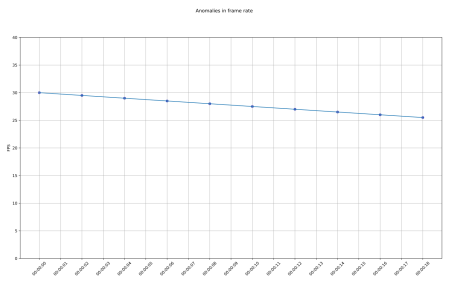
(a) A series of abnormal values



(b) Repeated abnormal values (periodic)



(c) An upward trend over a longer period of time



(d) A downward trend over a longer period of time

Figure 4.4: Illustrating global regression cases

Global cases might indicate a more serious or sustained problem within the system. In some instances, the abnormality depends strictly on the context of the test and use case. Identifying this type of abnormality requires analyzing data for patterns over a longer period.

As seen, the regression cases are context-based and have variation in their pattern that introduces complexity in detection. This underscores the need for a nuanced approach that considers specific occurring characteristics and contexts, meaning an adaptive and flexible solution.

4.5 Data Exploration

To understand the problem and define the named alerting cases in Section 4.4, we began with exploring the available data. The main purpose and general findings of this exploration are presented in this section, while more detailed method and results are provided in Section 5.3.

The main purpose of this exploration was to get familiar with the data and gain a better understanding of different scenarios. This involves identifying and defining alerting cases through simple data exploration. Additionally, the exploration aimed to investigate recurring patterns in the data by examining test cases, test settings, and different products. Finally, it explored the existence of correlations between various features.

Findings

A **feature** here refers to performance metrics such as video frame rate, bitrate, and others, as previously discussed. The general findings derived from the conducted data exploration are as follows:

- **Stability:** The video frame rate typically appears more stable compared to other features, especially in lighter test settings such as default setting tests. Overall, the stability and value ranges of features vary by test case and product.
- **Defining normal behavior for a feature:** Normal behavior is context-specific, defined for each feature and test case.
- **Performance depending on codecs:** Some codecs exhibited lower performance compared to those used in the default settings.
- **Observing deviations in box-plots:** Box-plots provided a clear view of deviations between different builds. These plots helped identify potential performance regressions in some cases.
- **Correlation analysis & relationships:** A strong positive correlation was observed between bitrate and bandwidth. In some cases, there was a correlation between memory and CPU usage. When deviations in video frame rate occurred, bitrate sometimes followed a similar pattern. However, these correlations were not consistent across all cases.

These findings guided us through further investigation and the formation of solutions.

4.6 Requirements

Through problem analysis and conceptualization, we identified key requirements and desired criteria for the solution. These requirements were established as objectives for this project, guiding the development process. They are listed below:

Outcome & Development

The outcome and development category of requirements is designed to help us cover the aspects related to **RQ1**. The solution shall:

1. Detect and report cases of interest regarding each new test execution.
2. Achieve a high detection rate, meaning minimizing missed cases.
3. Ensure more sensitive detection regarding default setting tests, as required by the company.
4. Provide reliable results using minimal input data, ensuring high data efficiency.
5. Be implemented in Python using common libraries to ensure compatibility with the existing code base at the case company.
6. Use technical components that are as explainable as possible to avoid unnecessary complexity.
7. Be evaluated through consultation with domain experts and data labeling.
8. Deliver a solution that is as general as possible, facilitating use across different products and test cases.

Usage

The usage category of requirements is designed to help us cover the aspects related to **RQ2** and **RQ3**. The solution shall:

1. Simplify the monitoring and detection work to assist the quality engineer and reduce their workload.
2. Be easy to integrate into the current process.
3. Allow adjustment of the detection parameters as needs evolve.
4. Be user-friendly and intuitive to operate.

Chapter 5

Solution

This chapter outlines the steps taken to propose a solution to the problem defined in Chapter 4. It provides details about the implementation and evaluation of the proposed solution, along with the rationale behind these decisions.

5.1 Solution Overview

The objective is to propose a solution that simplifies manual analysis and assists quality engineers in reducing their workload, while addressing the research questions presented in Section 1.2. The solution involves developing a tool that meets the criteria and requirements outlined in Chapter 4. This tool aims to detect new test executions related to a product that exhibit unexpected behavior or regressions. The detected cases are then presented to a quality engineer through a visualization tool developed during this project, allowing them to provide feedback that can be used to improve future detections.

As described in Section 2.3, the study focuses on three selected products, referred to as *A*, *B*, and *C*. The initial data exploration, mentioned in Section 4.5, was conducted on these products and is analyzed further in Section 5.3. This analysis helped us identify performance metrics that could serve as input features for this study. With a thorough understanding of the available data, we began by exploring the feasibility of a solution based on statistical thresholds. Subsequently, we investigated solutions using unsupervised machine learning and later transitioned to supervised learning. To evaluate these approaches, datasets for each product were labeled using a tool developed during this project. Continuous feedback from the quality engineer, gathered through the visualization tool, will be incorporated into future training and detection.

5.2 Solution Motivation

The primary motivation behind this project is to address the research questions outlined in Section 1.2 and to solve the defined problem in Chapter 4. Our main research question, **RQ1**, aims to explore how detection can be done more efficiently. This involves developing a solution that assists quality engineers in detecting regression cases, as described in Section 4.4, by analyzing data retrieved from performance test results.

Providing a solution in the form of a tool to achieve this objective is central to this study. Python was chosen for the tool's implementation, as required in Section 4.6, due to its widespread use in data analysis and machine learning [31], and to ensure compatibility with the case company's extensive existing Python codebase. Various potential solutions were sketched by reviewing related literature presented in Section 3.6 and the problem construct in Chapter 4, including inspirations from existing quality assurance processes, as discussed in Subsection 4.1.2. A more detailed motivation, according to the research questions and requirements of this study, is presented in the following sections, grouped into two categories: development and outcome-related, and usage-related.

Development of the Tool & Outcome

The following points address the requirements in Section 4.6 which are related to **RQ1**.

Based on the insights gained from problem identification and data analyses, the focus was initially narrowed down to investigating two critical performance metrics for the case company while focusing on single-channel tests. These metrics were bitrate and video frame rate, which also corresponds to frames per second (FPS) further in this study. Later, the scope was extended to include additional metrics such as CPU and memory, as well as more advanced test cases like multi-channel tests.

Our primary goal for the detection mechanism was to minimize missed cases and achieve a high detection rate, guiding our evaluation and selection of solutions. We began by exploring *statistical thresholds* as an initial solution, which involves defined rule sets based on contextual data, as detailed in Section 5.4. This approach advocates for simplicity to avoid unnecessary complexity. Although effective in certain scenarios, statistical thresholds did not provide the results, flexibility, and adaptability required for this study. This highlighted the need for further investigation into a more suitable solution.

After experimenting with thresholds, we considered *unsupervised learning* methods. This approach does not require labeled data, making it a compelling choice for initial detection efforts. To compare the results from different approaches, we evaluated them, including unsupervised methods, against labeled data. We further investigated optimizing the approach based on unsupervised learning. However, to achieve more precise and reliable detection, we transitioned to a solution based on *supervised learning*.

In supervised learning, models are trained on a dataset where each entry is labeled as either an anomaly (1) or normal (0). This method allows the model to predict the class of new data points, specifically data from new test executions. While unsupervised learning was initially explored, the potential for greater control and performance with supervised learning, despite its time-consuming data labeling process, led us to prioritize this approach for a potentially better overall solution. To enhance data efficiency and ease of use, we investigated methods to minimize the required labeled data size and simplify annotation. Furthermore, to reduce computational requirements, we adopted a multi-stage approach, exploring simpler methods first to find the most efficient solution that addresses the problem effectively.

Usage & Integration

The following points address the requirements in item 4.6 which are connected to **RQ2** and **RQ3**.

By addressing our research questions, this project aimed to develop a tool that not only improves detection mechanisms but also enhances existing work processes and adapts to evolving needs, providing a comprehensive and effective solution for analyzing performance test results. This was achieved by designing the annotation tool to be simple and user-friendly, and by visualizing the results in a manner familiar to users, thereby allowing the solution to be easily integrated into the existing process.

We ensured the solution's adaptability to evolving needs by allowing users to provide feedback on the results. To assess the impact of the solution on the process and engineers' workflow, we conducted usability testing and observations.

5.3 Data Analysis

The purpose and general findings of the data exploration conducted in this study are outlined in Section 4.5. This section presents the detailed method and results of the analysis of the exploration.

Analysis Method

Exploratory data analysis (EDA) was conducted on performance data collected over 20 weekly test executions for the selected products. Video frame rate and bitrate was analyzed as the primary features due to their importance. The analysis included the listed activities regarding each feature, test case and product under study:

- Plotting values over time.
- Reviewing statistical metrics, including mean, median, standard deviation, minimum value, maximum value, and percentiles.
- Plotting rolling standard deviation.
- Reviewing box-plots across consecutive weekly test executions.

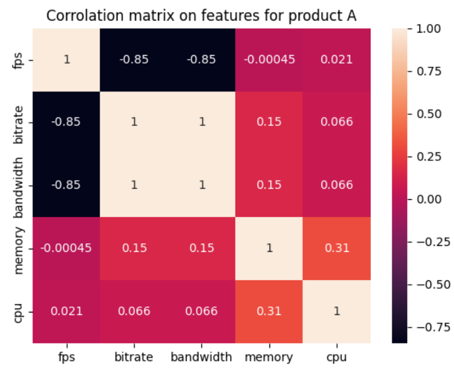
Additionally, a correlation analysis was performed on features such as FPS, bitrate, bandwidth, CPU usage, and available memory. This involved calculating the correlation between each feature to assess relationships between them. This analysis helps to identify redundancy in the data, reduce data dimensions, and uncover more characteristics of the data.

Analysis Result

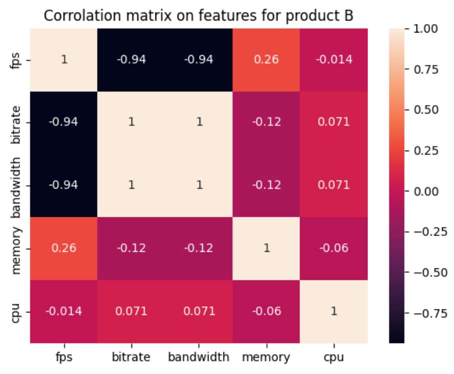
Plotting the measurements of each feature over time for each test case and product allowed us to observe the level of variation and value range over time, providing insights into common and uncommon behavior. Statistical metrics gave a general overview of the data's characteristics and variations. The rolling standard deviation highlighted the variation of values over time, potentially indicating problematic cases. Box-plots revealed deviations in test executions, helping us investigate how detection based on deviations could be implemented.

One key observation was that data from default setting tests, compared to those using custom settings, showed more stable values, usually following the same pattern and value range. Values were typically more stable in lighter test settings compared to more advanced test settings (e.g., using applications). When comparing data from test cases using different codecs, specific codecs resulted in lower performance compared to others or the default ones. Overall, the analysis revealed significant variation in data characteristics across test cases and products, which was expected.

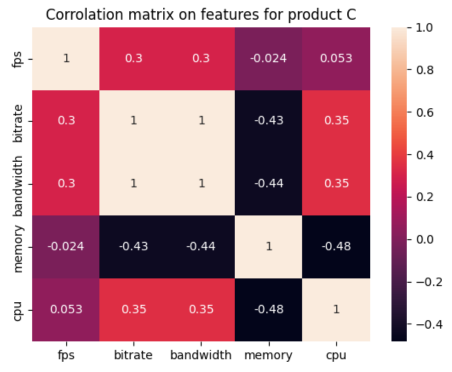
The key findings regarding the comparison of data characteristics between features indicated that the nature of an alerting case differs based on the feature. For example, bitrate measurements, in megabits per second (Mbps), are generally stable with fluctuations within the expected range. However, when regressions occur, the bitrate values deviate significantly from the norm, indicating potential issues that require attention. This is not always the case with FPS values. In some instances, a change in FPS values was mirrored by changes in bitrate values at approximately the same timestamp.



(a) Correlation Matrix for Product A



(b) Correlation Matrix for Product B



(c) Correlation Matrix for Product C

Figure 5.1: Figures on correlation matrices for each product

The results of the correlation analysis are presented as heat maps in Figure 5.1. A value of 1 represents the highest positive correlation, while -1 represents the highest negative correlation. The correlation study revealed that bitrate and bandwidth metrics consistently have a strong positive correlation, as shown by the value 1 in the cells representing bitrate-bandwidth correlation. FPS and bitrate generally showed a partially strong negative correlation, as seen in Figure 5.1a and Figure 5.1b. Some correlation was observed between available memory and CPU usage, mostly in the case of product C, as shown in Figure 5.1c, indicating an anticipated relationship between these two features. However, apart from the strong correlation between bitrate and bandwidth, other correlations were not consistent across all test cases and products. For instance, a strong correlation between frame rate, bandwidth, and bitrate was present except in Figure 5.1c for product C.

5.4 Statistical Thresholds

The initial step in developing our solution was to assess the feasibility of implementing a solution based on statistical thresholds using statistical methods. The aim was to determine whether the identified problem could be addressed with adaptable, data-driven thresholds that would allow for simple and explainable detection of alerting cases.

The thresholds were established based on the results from the data analysis and in consultation with the quality engineer, considering the alerting cases described in Section 4.4. These thresholds were not static; instead, they adapted to the character of the data through statistical measures such as quartiles, representing the expected value range for both FPS and bitrate. The bounds of these thresholds were statistically calculated using the interquartile range (IQR) and the mean value, following the formula $\boldsymbol{bound = mean \pm factor * IQR}$.

Since using only statistical measures was not sufficient for detecting alerting cases, various factors such as the bound factor, the local factor, and the quartile factor were introduced to refine the thresholds depending on specific features and test settings. For instance, tests involving a specific codec that often exhibited higher variability used thresholds adjusted to accommodate a wider range, apart from the thresholds being statistically set.

Thresholds Results

Implementing and evaluating these statistical thresholds provided insights into the complexity of the problem. Table 5.1 summarizes the results of the statistical threshold evaluation for products A, B and C. The evaluation metrics include AUC, recall, precision, F1 score, precision, false positive rate (FPR), and false negative rate (FNR).

Table 5.1 shows that using FPS alone generally provided better performance compared to using bitrate alone. For Product A, FPS alone achieved an AUC of 0.91, while bitrate alone achieved 0.60. Combining FPS and bitrate resulted in an AUC of 0.72. For Product B, FPS alone achieved an AUC of 0.68, bitrate alone achieved 0.65, and combining both features resulted in an AUC of 0.65. For Product C, FPS alone achieved an AUC of 0.71, bitrate alone achieved 0.77, and combining features resulted in an AUC of 0.66.

Notably, the FNR and FPR were relatively high in several cases. For instance, in Product A, using both features resulted in an FNR of 0.26 and an FPR of 0.30, indicating that many true anomalies were missed. Similarly high FNR and FPR values were observed for Prod-

Table 5.1: Evaluation Results of Statistical Thresholds on Test Runs for Products A, B, and C. Test set size indicate the number of weekly test executions used in the evaluation.

Product	Feature	Test Set Size	AUC	Recall	Accuracy	F1 Score	Precision	FPR	FNR
A	fps	4,924	0.91	0.92	0.91	0.77	0.67	0.10	0.08
	bitrate	4,924	0.60	0.48	0.68	0.32	0.24	0.29	0.52
	fps & bitrate	4,924	0.72	0.74	0.71	0.58	0.48	0.30	0.26
B	fps	4,557	0.68	0.38	0.83	0.54	0.95	0.01	0.62
	bitrate	4,557	0.65	0.53	0.76	0.25	0.17	0.22	0.47
	fps & bitrate	4,557	0.65	0.50	0.71	0.51	0.51	0.20	0.50
C	fps	1,815	0.71	0.45	0.84	0.58	0.81	0.03	0.55
	bitrate	1,815	0.77	0.82	0.73	0.38	0.25	0.28	0.18
	fps & bitrate	1,815	0.66	0.60	0.68	0.51	0.44	0.29	0.40

ucts B and C. These high FNR and FPR values suggest that while the statistical thresholds were useful for identifying anomalies in some cases, they lacked the precision and robustness needed for reliable detection.

Given these results, we recognized the need for a more advanced solution. Therefore, we investigated whether transitioning to machine learning-based approaches would address the problem better compared to statistical thresholds.

5.5 Data Preprocessing

Data preprocessing was performed as a preliminary step to implement a solution based on machine learning. This step was essential for preparing the data to be fed into the models and for improving their performance. It ensured that the data was consistent and in a suitable format for optimal model results. The input to machine learning models is typically a matrix of numerical data, with features as columns.

When fetching the data from the case company's servers, the dataset included performance measurements along with a timestamp for each entry. Additionally, it contained the name of the test case for each entry as a categorical feature among all other features as columns. To identify which test case each entry belonged to, we used the test name column.

Feature Encoding for Test Differentiation: Since test case names are categorical, we applied one-hot encoding to transform these names into numerical binary values. In the input matrix, each unique test name is represented by a binary feature; a *1* in a given test name column denotes that the entry is related to that test name, while a *0* indicates its absence. This transformation is crucial for enabling the model to differentiate effectively between various test cases[1].

[1] `https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/`

Data Cleansing: Data cleansing was performed to remove any incomplete, insufficient, and corrupted entries. This step increases the quality and reliability of the data fed into the model, avoiding inconsistencies that could impair the model's performance.

Time Alignment: Due to discrepancies in the timing of recorded performance metrics, such as frame rate and bitrate noted to the precise millisecond versus slightly delayed timestamps for CPU usage and available memory, it was necessary to align and match all entries by the second instead of the millisecond. This step ensured that each set of measurements shared consistent timestamps, allowing for accurate mapping of CPU and memory measurements to the corresponding frame rate and bitrate measurements.

Normalization: The features' data were normalized using scikit-learn's StandardScaler. This technique ensures all features have a similar scale, preventing features with larger ranges from dominating the model's learning process. By doing so, the model can focus on the relative importance of each feature, regardless of its unit or scale [2].

Dataset

For each product *X*, a general dataset was created using a chosen number of test executions. The processed data were organized into training and test datasets before use in models. The data for each product included performance metrics presented earlier along with their timestamps. As a result of the data analysis conducted in Section 5.3, the final features used in the machine learning approaches are detailed in Table 5.2.

Table 5.2: Overview of the Dataset

Feature/Column Name	Description
Datetime	The timestamp of a measurement point
FPS	The video frame rate value
Bitrate	The bitrate value
CPU	The total CPU usage value
Memory	The available memory value
Test Names (One-Hot Encoded)	The test names transformed into binary features using one-hot encoding.

5.6 Annotation Tool

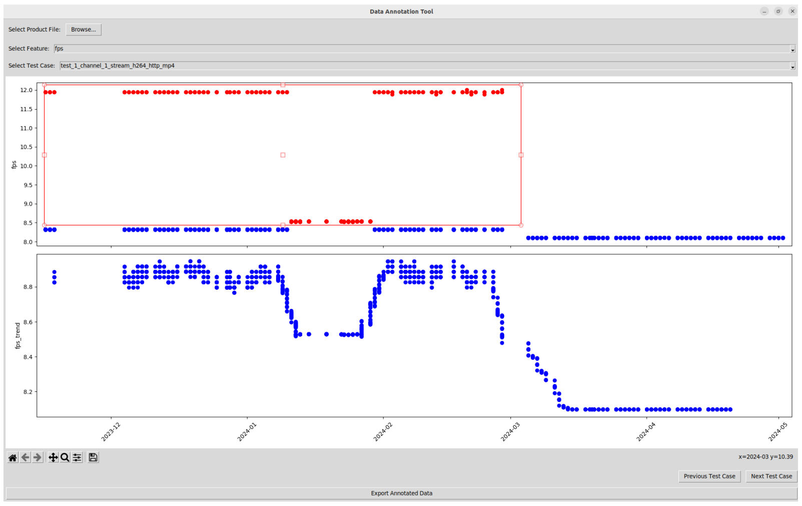
The development of a reliable detection tool required labeling the dataset to evaluate the results of the detection tool. Given the lack of pre-existing labeled data and the large size of our dataset, an efficient and user-friendly annotation tool was crucial. The annotation tool was developed using Python's Tkinter library[3], providing a straightforward graphical user interface (GUI) for ease of use as shown in Figure 5.2.

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

[3] `https://docs.python.org/3/library/tkinter.html`

The workflow of the tool begins with the user loading the dataset of a product as a CSV file. Once the data is loaded, the user selects the specific feature they wish to annotate for each test case. The GUI of the annotation tool is designed to be intuitive, featuring two main plots for data visualization. The first plot displays the measurement points for the selected feature and test case, allowing users to view individual data points in detail. The second plot provides an overview of data trends over time, aiding in the identification of patterns and potential anomalies.

Users can interact with the plots by drawing a box around points they wish to mark as anomalies, see the red box in Figure 5.2. This interactive approach simplifies the process of labeling large datasets by visually assisting the user in identifying abnormalities. The tool also includes options for zooming and adjusting the view to closely inspect different sections of the data. After completing the annotation, the user can export the labeled data to a CSV file. This annotated dataset is then used for model training and/or evaluation. This user-centric design ensures that the annotation process is both efficient and accurate.



Select Product File:

Select Feature: fps

Select Test Case: test_1_channel_1_stream_h264_http_mp4

x=2024-03 y=10.39

Figure 5.2: The Developed Annotation Tool: The first plot displays measurement points for a selected feature in a test case. The second plot shows data trends over time to help identify patterns. Users can easily draw a box around points in the first plot to mark them as anomalies.

The development of the annotation tool followed an iterative process in which quality engineers assessed its usability and modifications were made accordingly. This iterative development ensured that the tool met the practical needs of engineers and improved its overall effectiveness. The tool was tested in collaboration with quality engineers to label datasets for the selected products. This collaborative effort ensured that the annotations were accurate and aligned with detection criteria.

5.7 Unsupervised Learning as Approach

In this stage of our project, we focused on exploring various anomaly detection techniques using unsupervised learning methods. This approach was particularly motivated by the lack of officially labeled data. Given the complexity and variability of our data, we initially constrained our experiments to simpler scenarios involving single channel single stream test cases. This reduction allowed us to thoroughly evaluate each model's capability in identifying anomalies before scaling up to more intricate datasets.

We selected 10 different unsupervised models for our initial analysis: K-means, DBSCAN, Autoencoder, One-Class SVM (OC-SVM), Isolation Forest, Local Outlier Factor (LOF), Copula-Based Outlier Detection (COPOD), Histogram-Based Outlier Score (HBOS), Gaussian Mixture Model (GMM), and K-Nearest Neighbor (KNN). We configured the models to apply stringent detection criteria, aiming to identify as many anomalies as possible while minimizing the risk of missing true anomalies.

5.7.1 Investigation Setup

We used data from 110 test executions for training and tested the models on the 5 most recent executions for each product. We first focused on two primary features: FPS and bitrate. To evaluate the impact of different training and test set configurations on the models' performance, we used three distinct strategies:

1. Single Test Case Training: Models were trained and tested on individual test cases.
2. Aggregated Training with Individual Testing: A model was trained using data from all test cases of a product but tested on individual test cases.
3. Aggregated Training and Testing: A model was trained on all test cases of a product and tested on all test cases collectively.

The initial results indicated that training models on individual test cases for each product yielded the most reliable performance. K-means consistently underperformed in our experiments due to its assumption that clusters are spherical and isotropic, which does not hold true for the complex and varied shapes of clusters in performance data. This limitation made K-means less effective for our specific detection requirements, leading us to exclude it from further tests.

Our observations also showed that DBSCAN required a significant amount of computing resources, particularly in terms of memory usage. The high resource demand is due to the computational complexity of DBSCAN, especially when dealing with large datasets and high-dimensional data, as it requires calculating the distances between all points in the dataset. For this reason, DBSCAN was also excluded from further experiments.

Initial Results

We evaluated the performance of each model using several metrics: recall, AUC, accuracy, F1-Score, and precision. Given the importance of minimizing false negatives in the detection process, recall was prioritized as the primary metric, followed by AUC and accuracy respectively. The results summarize the performance of the models using FPS and bitrate as features while training and predicting on individual test cases of each product.

In the first part of the results, specifically in Table 5.3 for Product A, five models achieved a recall score of 1.0. Although HBOS had a slightly lower recall of 0.94, it excelled in other metrics. For Product B, as shown in Table 5.4, KNN demonstrated best performance in all metrics. Similarly, for Product C, as seen in Table 5.5, KNN led the metrics except recall.

Table 5.3: Initial Evaluation for Product A

Metric	AutoEncoder	OC-SVM	IF	LOF	COPOD	HBOS	GMM	KNN
AUC	0.51	0.54	0.53	0.68	0.59	**0.86**	0.53	0.70
Recall	**1.00**	**1.00**	**1.00**	0.89	**1.00**	0.94	**1.00**	0.60
Accuracy	0.20	0.25	0.23	0.55	0.33	**0.81**	0.24	0.77
F1 Score	0.32	0.33	0.33	0.43	0.36	**0.65**	0.33	0.49
Precision	0.19	0.20	0.20	0.28	0.22	**0.50**	0.20	0.42
False Positive Rate	0.99	0.93	0.95	0.53	0.83	**0.22**	0.93	0.19
False Negative Rate	**0.00**	**0.00**	**0.00**	0.11	**0.00**	0.06	**0.00**	0.40
Test set size	185	185	185	185	185	185	185	185
Train executions	110	110	110	110	110	110	110	110
Test executions	5	5	5	5	5	5	5	5

Table 5.4: Initial Evaluation for Product B

Metric	AutoEncoder	OC-SVM	IF	LOF	COPOD	HBOS	GMM	KNN
AUC	0.54	0.65	0.51	0.64	0.76	0.79	0.61	**0.88**
Recall	0.84	**1.00**	**1.00**	**1.00**	**1.00**	0.95	**1.00**	**1.00**
Accuracy	0.38	0.47	0.25	0.45	0.63	0.71	0.40	**0.81**
F1 Score	0.39	0.47	0.39	0.46	0.56	0.61	0.44	**0.72**
Precision	0.26	0.31	0.24	0.30	0.39	0.45	0.28	**0.56**
False Positive Rate	0.77	0.70	0.98	0.72	0.49	0.37	0.79	**0.25**
False Negative Rate	0.16	**0.00**	**0.00**	**0.00**	**0.00**	0.05	**0.00**	**0.00**
Test set size	185	185	185	185	185	185	185	185
Train executions	110	110	110	110	110	110	110	110
Test executions	5	5	5	5	5	5	5	5

Table 5.5: Initial Evaluation for Product C

Metric	AutoEncoder	OC-SVM	IF	LOF	COPOD	HBOS	GMM	KNN
AUC	0.66	0.66	0.67	0.67	0.74	0.83	0.69	**0.83**
Recall	**1.00**	0.92	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.85
Accuracy	0.44	0.49	0.45	0.45	0.57	0.72	0.48	**0.83**
F1 Score	0.38	0.39	0.39	0.39	0.45	0.55	0.40	**0.63**
Precision	0.24	0.24	0.24	0.24	0.29	0.38	0.25	**0.50**
False Positive Rate	0.68	0.60	0.66	0.66	0.52	0.34	0.63	**0.18**
False Negative Rate	**0.00**	0.08	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.15
Test set size	75	75	75	75	75	75	75	75
Train executions	110	110	110	110	110	110	110	110
Test executions	5	5	5	5	5	5	5	5

Table 5.3, Table 5.4 and Table 5.5 also show a very high false positive rate, meaning that the models marked most of the cases as anomalies.

Despite being computationally intensive, the Autoencoder did not perform as well as expected, as evidenced by the evaluation results for different products. Consequently, it was excluded from further experiments. These findings guided our subsequent steps, allowing us to focus on the most promising models for further development.

5.7.2 Towards Generalized Approach

The results of the initial unsupervised learning experiments varied between different products. This was expected as the data analysis, discussed in Section 5.3, showed that the data has unique characteristics for different test cases and products. A more generalized solution that could encapsulate better performance in a wider range of test cases and products would be beneficial. Following this aim, we investigated generalizing the solution regarding unsupervised models.

To develop a more generalized approach, we took inspiration from ensemble learning methods and combined the predictions from the remaining models forming a *Mode Ensemble*, as illustrated in Figure 5.3. By aggregating the predictions, we aimed to leverage the strengths of each model and mitigate their individual weaknesses. We collected the predictions from all models and calculated the mode (the most frequent prediction) among them. The mode value then became the final prediction. This approach ensured that anomalies were identified based on a consensus, thereby enhancing the robustness of the detection mechanism. The best results for this approach are presented in Table 5.6, in comparison to other ensemble approaches.

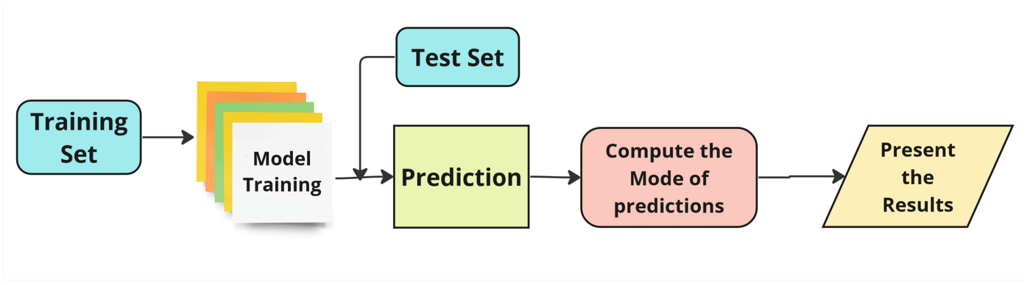


Figure 5.3: Simplified overview of the ensemble approach based on the mode value of the predictions

Another approach for generalizing the solution based on unsupervised learning was using *Scalable Unsupervised Outlier Detection (SUOD)* framework [25]. The SUOD framework is designed to parallelize the execution of multiple detection algorithms and combine their results. We specifically used OC-SVM, HBOS, and IF as the estimators within this framework. The results of using the SUOD framework are presented in Table 5.6.

Model Selection Algorithm

To experiment with optimizing the performance of unsupervised learning, we developed a custom algorithm designed to select the best-performing model for each test case for a product. We called this approach the *Model Selection* algorithm, which chooses the most effective model for each specific test case. An overview of how the algorithm is used is illustrated in Figure 5.4. The process begins with training seven unsupervised models using a training set. These models include OC-SVM, IF, LOF, COPOD, HBOS, GMM, and KNN.

Once trained, these models are used to predict anomalies in a separate tuning set. Simultaneously, a quality engineer labels the tuning set to provide ground truth in order to be used by the algorithm. The algorithm evaluates the prediction results for each model using the labeled tuning set. This is done by ranking according to evaluation metrics in the following order: prioritizing recall to minimize false negatives, followed by accuracy, F1 score, and precision. After this evaluation, the algorithm identifies the best-performing model for each test case across different products. The selected models are then employed to detect anomalies in upcoming test executions. The best results regarding this approach are presented in Table 5.6.

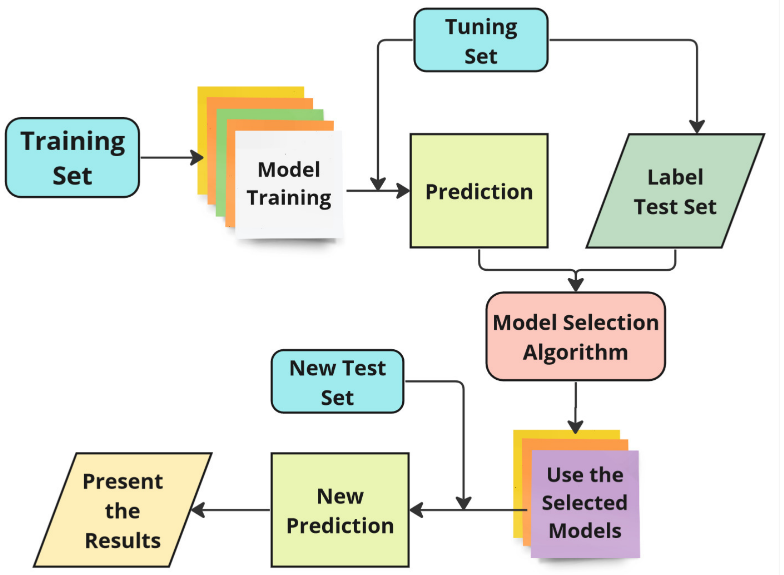


Figure 5.4: Simplified overview of using the Model Selection algorithm

Improvement Experiments

We used different methods for training the unsupervised models and incorporated additional feature engineering techniques, such as calculating averages, percentile changes, and differences between points in the dataset. These experiments were done to investigate if they could improve the performance of the unsupervised approach. These experiments used the previous generalization approaches and were conducted by training individually on each test case, as explained in Subsection 5.7.1. Below is a summary of the experiments that yielded the most positive results among all those conducted:

Exp1 **Direct Training and Prediction:** This straightforward approach involves training the machine learning models on the entire training dataset and then using them to predict anomalies in a test set. This method provides a direct assessment of the model's performance on unseen data.

Exp2 **Baseline Training:** In this approach, models are trained on a selected number of weekly test executions (five) that represent the expected data. This establishes a baseline for the behavior, which can then be used to train models.

Exp3 **Deviation Analysis:** This method calculates the average of each performance metric in a chosen baseline according to *Baseline Training*. The models are then trained and tested on the difference between these averages and the measurements in the dataset.

Exp4 **Relation Analysis:** Similar to *Deviation Analysis*, this method further includes the relation between the average of each performance metric in the baseline and the measurements by calculating $\frac{\text{avg for metric}}{\text{avg for metric in baseline}}$. Training on both differences and percentage deviations aims to provide a more nuanced feature set for detection.

Exp5 **Standardization with Z-Score:** Before training, this experiment standardizes the data using z-scores, which adjust each feature based on its mean and standard deviation. This standardization helps make the features more comparable with the aim of improving the model's ability to identify anomalies.

The results that made a significant difference in the performance of the models are presented among the best results achieved in Table 5.6. Other experiments did not result in expected performance thereby they are not studied further.

Results of Generalization

In Table 5.6, we present the best results obtained by using the generalization approaches with the different improvement experiments discussed above. We only include the top results from the five experiments. The table shows that the Model Selection algorithm outperformed others in most metrics when testing on all three products, except in recall when using the Direct Training and Prediction method. For the mode ensemble method, the best results were also obtained using the Direct Training and Prediction method across all three products. The ensemble method outperformed in recall for two of the products. Conversely, SUOD underperformed in all three products. Additionally, the Model Selection algorithm achieved the best FPR value in two of the three products. Based on these findings, we decided to continue with the Model Selection algorithm for further analysis with unsupervised learning, specifically using the Direct Training and Prediction method.

Table 5.6: Best results from Feature Engineering & Generalization experiments for products A, B, and C

Product	Method	Features	Experiment	AUC	Recall	Accuracy	F1 Score	Precision	FPR	FNR	Training Size	Tuning Size	Test Size
A	**Model Selection**	fps & bitrate	**Exp1**	**0.92**	0.97	**0.89**	**0.76**	**0.63**	**0.13**	0.03	110	5	5
	Mode Ensemble	fps & bitrate	**Exp1**	0.59	**1.00**	0.34	0.36	0.22	0.82	**0.00**	110		5
	SUOD	fps & bitrate		0.45	0.80	0.23	0.28	0.17	0.91	0.20	5		5
B	**Model Selection**	fps & bitrate	**Exp1**	**0.87**	0.98	**0.82**	**0.72**	**0.57**	**0.23**	0.02	110	5	5
	Mode Ensemble	fps & bitrate	**Exp1**	0.72	**1.00**	0.57	0.52	0.35	0.57	**0.00**	110		5
	SUOD	fps & bitrate		0.60	0.98	0.41	0.44	0.28	0.77	0.02	5		5
C	**Model Selection**	fps & bitrate	**Exp1**	**0.96**	**1.00**	**0.93**	**0.84**	**0.72**	0.08	**0.00**	110	5	5
	Mode Ensemble	fps & bitrate	**Exp1**	0.73	**1.00**	0.55	0.43	0.28	**0.00**	**0.00**	110		5
	SUOD	fps & bitrate		0.72	**1.00**	0.53	0.43	0.27	0.56	**0.00**	5		5

5.7.3 Extending the Scope

As explained earlier, we initially narrowed down the focus of the study to only FPS, bitrate and tests with single channel and stream. Next, we expanded our study to additionally include CPU usage and available memory as performance metrics. Further, we included all types of test cases, including those involving multi-channel and multi-stream setups. This extension was the final step in assessing the feasibility of an unsupervised learning solution. We utilized the Model Selection algorithm along with the Direct Training and Prediction approach to evaluate the method with all four performance features together and including all test cases.

In Figure 5.5 we illustrate the rolling evaluation method used to assess the performance of our unsupervised learning models with the model selection algorithm. This evaluation process is designed to mimic real-world scenarios where models are continually applied to new data. The evaluation begins by training the models on a training set consisting of the first 10 test executions. After training, a labeled tuning set, which includes the next execution (the 11th execution), is used by the model selection algorithm . The selected models then predict anomalies in the test set, which consists of the 12th execution. Following the evaluation on the test set, the test set rolls forward by two executions at a time. This rolling forward continues iteratively through the entire dataset. Upon each prediction on a new test set, the models' performance is evaluated, and the evaluation metrics are recorded. The average of these metrics across all iterations provides an overall assessment of the model's capability in a setting that reflects continuous, real-world usage. By averaging the evaluation metrics, we obtained a measure of the approach's performance over time.

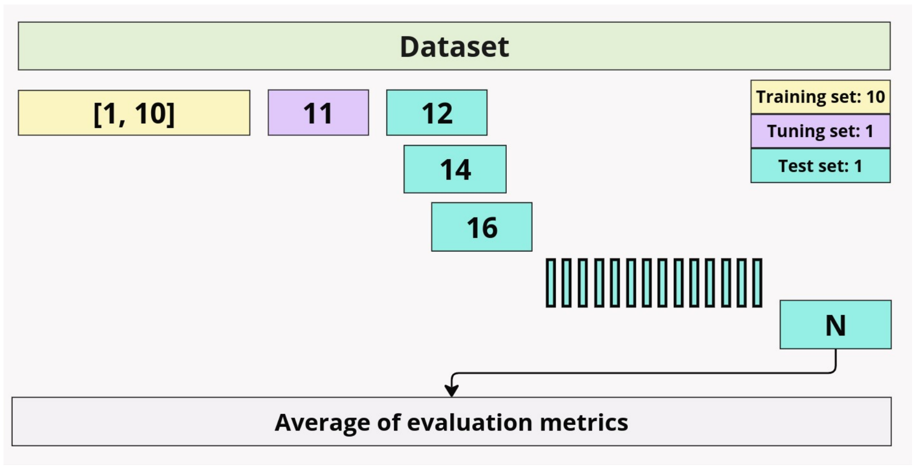


Figure 5.5: Overview of Rolling Evaluation for Unsupervised Learning with Model Selection Algorithm

The results of the rolling evaluation are summarized in Table 5.7. For Product A, the model achieved an AUC of 0.51, and a recall of 1.00. Product B showed an AUC of 0.56, and a recall of 1.00. Product C demonstrated an AUC of 0.57, and a recall of 1.00. These results indicate that the method were effective in detecting anomalies, as evidenced by the high recall values. However, having a high recall but relatively low AUC score means that the result includes positives even for negative cases, predicting anomaly when it is not an actual one. This resulted in a high portion of false positives as reflected in FPR and a low AUC.

Table 5.7: Rolling Evaluation Results for Products with 4 Performance Metrics/Features

Product	AUC	Recall	Accuracy	F1 Score	Precision	FPR	FNR
A	0.51	1.00	0.77	0.86	0.77	0.98	0.00
B	0.56	1.00	0.99	1.00	0.99	0.19	0.00
C	0.57	1.00	0.76	0.85	0.76	0.77	0.00

5.8 Supervised Learning as Approach

After evaluating the feasibility of the unsupervised solution on the expanded scope, we explored supervised learning techniques. This transition aimed to leverage the labeled data for more accurate anomaly detection through supervised classifiers. This was due to that the unsupervised solution resulted in many false positives, as indicated by the high *FPR* in Table 5.7. This meant that a quality engineer would need to review a large number of cases to rely on the unsupervised approach effectively.

Instead of reviewing all detected cases (both false and true positives), the engineer could spend the same amount of time labeling a portion of the data to investigate if a supervised approach would be more suitable. This shift would allow for the development of a more precise model, reducing the number of false positives and, consequently, the review workload for the engineer.

5.8.1 Evaluating Supervised Models

During our exploration of supervised learning, we utilized four different models: RandomForest, K-Nearest Neighbors (KNN), ExtraTrees, and Support Vector Machine (SVM). The models were trained on a labeled training set for each of the selected products. The process involved training models for each product using a training set, then testing the model on a test set. Our experiments revealed that to produce meaningful results with SVM, it required training a model for each test case of each product individually. In contrast, other models like Random Forest, KNN, and Extra Trees were able to generalize across different test cases within each product. Given the goal of proposing an efficient and general solution, as stated in Section 4.6, SVM was excluded from further analysis due to its higher computational intensity and lower generalization capability compared to the other used models.

To assess the capability of the remaining models (Random Forest, KNN, and Extra Trees), we followed a rolling evaluation method similar to the one used previously to evaluate the extended unsupervised solution. This evaluation process is illustrated in Figure 5.6. The process began by training the models on a training set consisting of the first 10 test executions and testing the models on the following test execution. We then moved both sets forward by two test executions in each step. For example, after the initial step, the training set included executions 3 to 12, and the test set was execution 13, as seen in Figure 5.6. This rolling process was repeated iteratively until the entire dataset was evaluated. By averaging each evaluation metrics across all iterations, we obtained a comprehensive assessment of the models' performance and generalization capabilities over time.

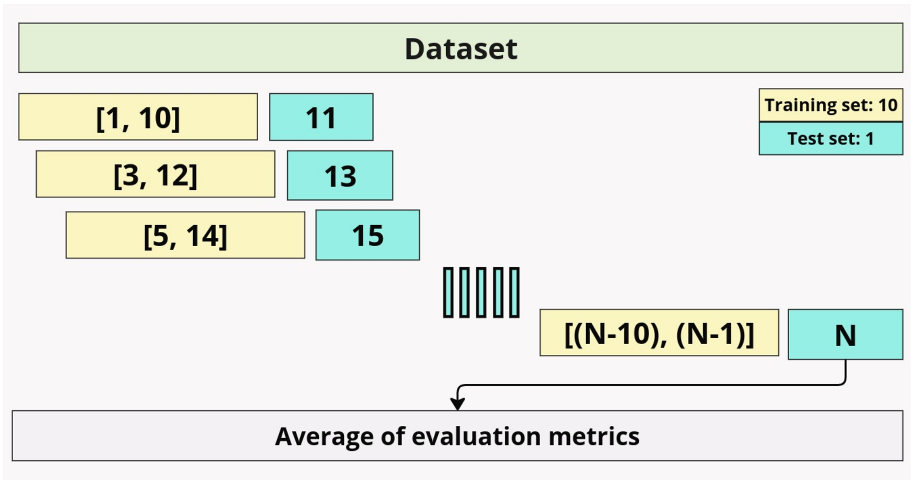


Figure 5.6: Overview of Rolling Evaluation on Supervised Learning

Evaluation Results

The results of the evaluation are shown in Table 5.8. For Product A, Random Forest and Extra Trees achieved AUC scores of 0.96, with recall values of 0.96 and 0.95, respectively. KNN showed slightly lower performance with an AUC of 0.93 and a recall of 0.89. For Product B, both Random Forest and Extra Trees achieved high recall and accuracy, with AUCs of 0.90. KNN performed comparably well with an AUC of 0.91. For Product C, Random Forest and Extra Trees performed well, each with an AUC of 0.95 and high recall values. KNN had an AUC of 0.94 and slightly lower recall.

The results in Table 5.8 indicate that Random Forest performed better in all evaluation metrics compared to the other two models across the three products.

Table 5.8: Rolling Evaluation Results for Supervised Learning on Products. The training set consist of 10 test executions

Product	Model	AUC	Recall	Accuracy	F1 Score	Precision	FPR	FNR
A	**RandomForest**	**0.96**	**0.96**	**0.97**	**0.97**	**0.99**	**0.03**	**0.04**
	KNN	0.93	0.89	0.91	0.93	**0.99**	**0.03**	0.11
	ExtraTrees	**0.96**	0.95	0.96	**0.97**	**0.99**	**0.03**	0.05
B	**RandomForest**	0.90	**1.00**	**0.99**	**1.00**	**1.00**	**0.05**	**0.00**
	KNN	**0.91**	0.99	**0.99**	0.99	**1.00**	**0.05**	0.01
	ExtraTrees	0.90	**1.00**	**0.99**	**1.00**	**1.00**	**0.05**	**0.00**
C	**RandomForest**	**0.95**	**0.96**	**0.95**	**0.96**	**0.96**	0.05	**0.04**
	KNN	0.94	0.93	0.93	0.94	**0.96**	**0.04**	0.07
	ExtraTrees	**0.95**	0.95	**0.95**	0.95	**0.96**	**0.04**	0.05

Labeling data for training supervised models is a time-consuming process. Therefore, we decided to minimize the amount of labeled data required while maintaining detection performance. To achieve this, we adopted an approach that involved using a relatively small training set consisting of 10 test executions.

5.8.2 Improvement Investigation

We aimed to enhance the performance of our initial supervised learning solution with the aim of reducing the required labeled training data and lowering the false negative rate. In the following, the conducted investigations and the results are presented.

Synthetic Anomaly Injection

Initially, we created a clean training set by removing all measurements labeled as anomalies (1). Next, we injected synthetic measurement points representing anomalies for all features (FPS, bitrate, CPU usage, and available memory) and labeled them as 1. This injection aimed to cover a wide range of potential anomalies that might occur in the test set. The injections were controlled by factors based on the interquartile range (IQR) of the cleaned dataset. These factors were then multiplied by various factors to represent different levels of anomalies. The number of injected points matched the number of data points in the cleaned training set, effectively doubling its size. By doing so, we ensured that the training set included a diverse range of anomalies.

Different features required varying sensitivity levels: frame rate (FPS) detection needed high sensitivity, CPU and memory usage required moderate sensitivity, and bitrate had the least sensitivity due to acceptable fluctuations in its original values. This differentiated sensitivity ensured that the models could accurately detect significant anomalies across all features without being overly sensitive to minor fluctuations.

The results representing the rolling evaluation for all products using synthetic anomaly injection are shown in Table 5.9. Comparing these results to those presented in Table 5.8 shows that almost all evaluation metrics exhibited lower performance. The injection appears to have negatively impacted the overall performance of the models, highlighting an increase in FPR and less efficient detection.

Table 5.9: Rolling Evaluation Results for Supervised Learning on All Products Using Synthetic Anomaly Injection. The training set consist of 10 test executions

Product	Model	AUC	Recall	Accuracy	F1 Score	Precision	FPR	FNR
A	**RandomForest**	0.85	**0.96**	**0.91**	**0.93**	0.91	0.25	**0.04**
	KNN	0.82	0.93	0.87	0.91	0.90	0.28	0.07
	ExtraTrees	**0.86**	0.95	0.90	**0.93**	**0.92**	**0.22**	0.05
B	**RandomForest**	**0.83**	**0.99**	**0.99**	**0.99**	**1.00**	**0.09**	**0.01**
	KNN	0.79	0.98	0.98	**0.99**	**1.00**	0.10	0.02
	ExtraTrees	0.81	0.98	0.98	**0.99**	**1.00**	**0.09**	0.02
C	**RandomForest**	**0.88**	**0.96**	**0.92**	**0.94**	0.93	**0.17**	**0.04**
	KNN	0.86	0.93	0.90	0.93	0.93	**0.17**	0.07
	ExtraTrees	0.87	0.94	0.91	0.93	**0.94**	0.16	0.06

Balancing Classes

Another approach that could contribute to improved detection was balancing the training set between anomalies and normal measurement points. This approach involves generating synthetic data that match the characteristics of the minority class (anomalies) to increase their representation in the training set. For this, we utilized the Synthetic Minority Over-sampling Technique (SMOTE).

Synthetic samples were generated for each performance metric in the training set. Due to the differing nature of anomalies between test cases and performance metrics, SMOTE was applied individually to each test case. The generated samples were then added to the original training set, maintaining the same data distribution and characteristics as the expected anomalies. This led to a more balanced dataset.

Despite our hypothesis, the results of this approach were worse than the results obtained using synthatic anomaly injection, as presented in Table 5.9. Consequently, we did not see the results as valuable to present and decided not to continue investigating this approach.

5.9 Feedback Mechanism

One of the requirements defined earlier was to provide a mechanism for quality engineers to give feedback on the model's results, refining them over time based on this input, adding adaptability to the solution. This refinement process includes two scenarios:

- **True Positives:** If the results are of interest to the quality engineer and are indeed true positives or indicate a problematic behavior, the engineer outlines the specific measurement points they are interested in detecting in future executions.
- **False Positives:** If the results are not of interest and turn out to be false positives, they should not be detected in future executions. In this case, the engineer can proceed to review the next test case.

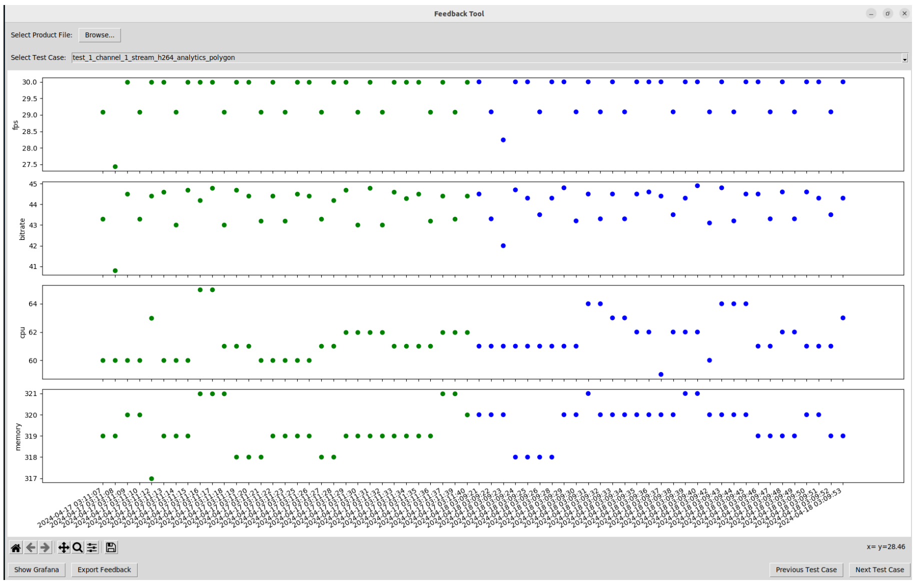
The main concept is that after the model presents its prediction results, the quality engineer reviews these results and provides feedback, indicating whether the alerted cases were interesting or not. This feedback is then incorporated into the training set, which is used to retrain the model. By gradually adding more accurate training data, the model aims to recognize the quality engineer's criteria for desired detection, reducing irrelevant and non-interesting detected cases over time.

To facilitate the feedback mechanism, we developed a simple user interface, seen in Figure 5.7, similar to the one presented in Section 5.6. The quality engineer can load the detection results as a CSV file and review the test cases flagged as problematic or containing anomalies in the latest test execution. Inspired by the current work process of quality engineers in analyzing results, the tool displays the detected cases together with the data for the previous test execution. This allows engineers to compare the detected cases with the baseline, helping them verify interesting cases. Another feature inspired by the current quality engineer work process was the addition of a button that links directly to the test cases on the Grafana[4] platform, which is currently used for manual analysis. This feature ensures that

[4]`https://grafana.com/oss/grafana/`

they can seamlessly switch between the feedback tool and their regular workflow to get a better perspective.

In general, this feedback loop and visualization interface were designed to facilitate an improved workflow for quality engineers, allowing them to refine the detection and ensure that the detection aligns with their current needs.



Select Test Case: test_1_channel_1_stream_h264_analytics_polygon

Figure 5.7: The Feedback Tool was developed to facilitate the feedback process. Green points represent data from the baseline, which are compared to the blue points representing the new execution.

Chapter 6

Evaluation

This chapter focuses on empirically validating the solution based on the supervised learning approach. It includes validation of the solution's performance, the effectiveness of the detection tool in the QA process, and its potential impact.

6.1 Supervised Learning Approach

To validate the supervised learning approach, we used a modified extended rolling window cross-validation technique designed to rigorously evaluate the model's ability detecting the alerting cases across different test executions. This validation is done to first examine the performance of the solution in a simulated setup of practical daily use and second the generalizability of the solution, as explained under *Cross Validation* in Section 3.5. In addition, performing this validation helps us to mitigate the threat of the solution being overfitted and limited. This threat is further discussed in Section 7.5. The validation strategy is illustrated in Figure 6.1.

It begins with an initial training set consisting of the first 10 test executions from a dataset of length N. Subsequently, we created $\frac{(N-10)}{2}$ different test sets, each containing data from a single test execution. The trained model predicted anomalies in these test sets, and evaluation metrics were calculated. The training set was then shifted forward by two test executions, and new test sets were created. This process was repeated until the entire dataset was used for training and testing. Finally, the average of each evaluation metric was calculated.

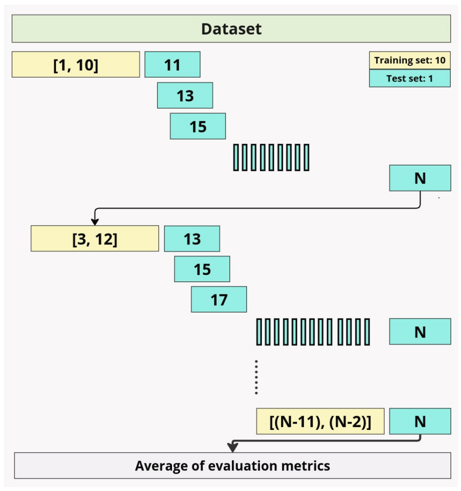


Figure 6.1: Overview of Cross-Validation Technique, *N: size of the dataset*

We also validated the impact of adding more data to the training set to simulate how incorporating feedback could affect performance. We used a validation technique similar to the previous one but with incremental training. The process of this technique is illustrated in Figure 6.2. Initially, we trained the model on a subset of the dataset and made predictions on the next test execution. After each prediction, we added the labeled data corresponding to the predicted test set, along with the next execution, to the training set. This process was repeated, with the training set expanding each time by including the new two executions, followed by making predictions on the subsequent execution. Once the entire dataset had been processed, we shifted the starting point of the training window forward by two executions and repeated the process. Evaluation metrics were calculated after each prediction, and the overall averages of these metrics were used to assess the model's performance.

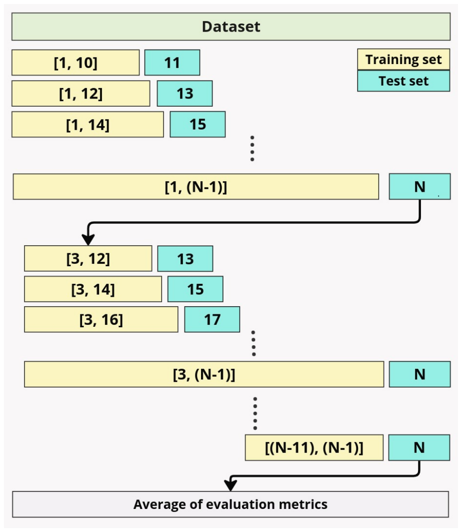


Figure 6.2: The Process of Incremental Sliding Window Cross-Validation

Validation results

In Table 6.1, we present the validation results for the solution using the RandomForest model. The results show strong performance across the three products, with particularly high recall scores. For Product A, the recall is 0.90, indicating the model effectively identifies 90% of true positives. Product B achieves a high recall of 0.99, highlighting a near-perfect detection capability for true anomalies. Product C also performs well with a recall of 0.93. These high recall values suggest that the RandomForest model is highly effective in identifying true anomalies across different products, making it a reliable choice for anomaly detection. However, there is an observed increase in the false positive rate compared to the results from the rolling evaluation in Table 5.8, indicating that a higher ratio of detections will include false positives.

Table 6.1: Results of Cross-Validation on RandomForest. $N = 111$ test executions

Product	Model	AUC	Recall	Accuracy	F1 Score	Precision	FPR	FNR
A	RandomForest	0.82	0.90	0.86	0.89	0.89	0.25	0.10
B	RandomForest	0.64	0.99	0.98	0.99	0.99	0.26	0.01
C	RandomForest	0.88	0.93	0.89	0.90	0.90	0.13	0.07

The results of validating the incremental training are presented in Table 6.2, showing that adding more labeled data to the training set improved the results over time. RandomForest was able to achieve a recall of 97% in two of the three products and the highest possible recall in the third product. It also achieved a minimum AUC of 90% across all products. Additionally, the false positive rate decreased to a more expected level, with a maximum of 0.08 across products, indicating that the addition of training data improves performance.

Table 6.2: Results of Incremental Cross-Validation on RandomForest. $N = 111$ test executions

Product	Model	AUC	Recall	Accuracy	F1 Score	Precision	FPR	FNR
A	RandomForest	0.96	0.97	0.97	0.97	0.98	0.04	0.03
B	RandomForest	0.90	1.00	1.00	1.00	1.00	0.08	0.00
C	RandomForest	0.95	0.97	0.96	0.96	0.94	0.05	0.03

6.2 Overall Solution Evaluation

The overall system, which combines supervised learning with a feedback loop, was validated to ensure it meets the project's goals. We conducted testing sessions with a quality engineer involved in the existing manual analysis process, who will potentially use the proposed solution. The goal was to evaluate the system's usability, effectiveness, and impact on the QA process.

The system was tested in a real-world environment to assess its performance under conditions similar to actual operations. This phase involved analyzing live performance test data from various products and making continuous adjustments based on real-time feedback and performance metrics. The testing session was divided into two smaller parts: one examining the annotation tool and the other focusing on the detection and feedback functionalities. The questions discussed during these sessions are presented in Appendix B.2 and Appendix B.3.

6.2.1 Evaluation of Annotation Tool

The evaluation of the annotation tool began by presenting the solution to the user and allowing them to get familiar with it. The user was then asked to experiment with the tool by annotating data from several test cases for a product. We observed the user's interaction with the tool and provided assistance and answers to questions as needed.

Some usability-related improvement points revealed during the session include:

- Better visualization for displaying the trend.
- Clearer navigation and zooming functionalities.
- Changing the names of the features to accurately represent the parameters they measure.

The results from the answers and discussions regarding the questions presented in Section B.2 indicate that the user interface is simple and easy to understand, including the functionalities needed for day-to-day annotation work by the engineer. However, there is room for improvement in terms of better zooming functionalities and the addition of more advanced features.

6.2.2 Evaluation of Detection & Feedback Tool

The evaluation of the detection and feedback tool followed a similar approach to the one used for the annotation tool, with the added step of guiding the user through the transition between different tools—from annotation to training, detection results, and feedback. The user was then asked to experiment with the visualization and feedback tool and share their opinions.

Potential improvements pointed out during the session include:

- Add more information about the product and test case.
- Clearer visualization of the baseline and new results.
- More automated integration of the tools for easier transition.

The engineer felt that the solution included all the necessary functionalities to serve as a result visualization and feedback tool in daily QA work. However, the engineer also noted that users need to understand how the solution works and how tasks should be performed, likely due to the low level of automation between different stages of the solution.

According to the engineer, the tool can effectively be used to review detection results on new test executions and compare them against the established baseline shown in the tool.

The engineer also mentioned that, unlike manual analysis, only the test results flagged as problematic need to be reviewed, making the workflow more efficient. Additionally, the engineer highlighted the benefit of being able to influence detection by submitting feedback on the detection results. This feedback, combined with previously labeled data, helps refine the detection process, providing greater efficiency, accuracy, and flexibility.

Overall evaluation: In general, the engineer noted that more initial work is required to get started with the solution, as 10 test executions need to be labeled beforehand to serve as the initial training set. However, the engineer also pointed out that the solution offers opportunities for easier use and reduced review time, and hopefully facilitate a more controlled, accurate, and efficient work process in the long run.

Chapter 7
Discussion

In this chapter, we discuss the results and outcomes from implementing and evaluating the presented solutions towards selecting the best solution to the defined problem. Further, we answer the study's research questions and present the conclusion.

7.1 Statistical Solution

Initially, we explored an approach based on statistical thresholds which offered limited flexibility and adaptability. The results showed that the statistical solution struggled with dynamic changes in data and varying test conditions. Consequently, it often generated false positives and missed an average of 38% of true anomalies across products. This necessitated frequent manual reviews and adjustments to the threshold factors. These limitations underscored the need for a more adaptive and automated approach, leading us to investigate machine learning solutions.

7.2 Analysis of Unsupervised Learning Results

The exploration of unsupervised learning models for detection began with evaluating 10 different models. Initial results showed that models such as K-means, DBSCAN, and Autoencoder were less effective due to their sensitivity to data distribution and high computational demands. Models like KNN and HBOS demonstrated better performance metrics, but the challenge of high false positive rates persisted across all models.

Further experiments with ensemble methods, such as combining predictions of different models and using the SUOD framework, aimed to enhance robustness. However, these approaches resulted in increased computational complexity without a significant reduction in false positives. The introduction of the Model Selection algorithm provided a more targeted approach by dynamically choosing the best-performing model for each test case. This improved overall detection performance but still faced challenges with generalizing across diverse test scenarios and the need for labeling a tuning set.

When we extended the study to include additional features like CPU usage and available memory, as well as more complex test cases, the unsupervised methods continued to struggle with high false positive rates, averaging 65% across the products, despite achieving high recall. This indicated that while unsupervised learning was effective in identifying potential anomalies, it lacked precision, leading to an increased review workload for quality engineers.

7.3 Analysis of Supervised Learning Results

Given the limitations of unsupervised learning, we transitioned to supervised learning methods to enhance detection accuracy. The rolling evaluation method ensured continuous evaluation against new data, simulating real-world usage. The results showed that RandomForest achieved the best performance across all evaluation metrics, averaging a recall of 97%, an AUC score of 94%, and a false positive rate of 4% across the three products.

To further improve this approach, experiments were conducted with synthetic anomaly injection and data balancing techniques. Despite the intention to enhance model performance, the results of synthetic anomaly injection were not promising. While it introduced a greater variety of anomalies into the training set, it also led to a decrease in overall performance metrics, decreasing the AUC score of RandomForest to 85% and increasing the false positive rate to 17% on average across the products. This indicates that the synthetic anomalies perhaps did not fully capture the characteristics of real-world anomalies. Another approach involved using the SMOTE which resulted in lower performance compared to the results from synthetic anomaly injection. These results highlights the challenge of generating synthetic data that accurately mimics real anomalies.

Supervised learning models, particularly Random Forest and Extra Trees, exhibited strong generalization capabilities. By training on labeled data generated by using the developed annotation tool, these models could effectively distinguish between normal and anomalous behavior across various performance metrics, test cases and products.

An essential requirement of this study was to implement a feedback mechanism to refine the model over time. The feedback is added by the quality engineer annotating the problematic data while reviewing the results on the detected cases. This feedback mechanism was facilitated through a user interface that allowed engineers to load detection results and review flagged test cases. This interface, inspired by the current work processes of quality engineers, included features for comparing detected cases with baseline data and linking directly to test cases in the Grafana platform. This tool resulted in a better workflow, allowing engineers to have the opportunity to refine the detection process continually to adapt to evolving QA requirements.

7.4 Unsupervised vs. Supervised

The previous findings directly address **RQ1**. The results highlight that while unsupervised learning can efficiently identify a wide range of anomalies with high recall, it struggles with precision, resulting in a high number of false positives. This necessitates significant manual review or labeling a tuning set for the Model Selection algorithm. On the other hand, the effort needed to review false positives or to label the tuning set are roughly the same as the effort needed to label the training set for the supervised approach. This makes supervised learning a more effective approach for detecting problematic performance test execution results in the context of the case company.

The results demonstrate that supervised learning, particularly with the feedback mechanism, assists quality engineers by reducing the number of false positives and providing a user-friendly interface for reviewing and refining detection results. This integration with existing workflows improves efficiency and effectiveness in reviewing performance test results, addressing **RQ2**.

The ability to integrate continuous feedback was an advantage of using supervised learning, enhancing its practicality and effectiveness. The findings underscore that the solution based on supervised learning is capable of adapting to changing detection criteria and new data characteristics, thereby providing adaptability and addressing **RQ3**.

Unsupervised learning solutions were found to be computationally more intensive compared to supervised solutions. This high computational demand can limit their scalability and practical application in production environments. Conversely, supervised learning solutions were generally more efficient and scalable. Models like Random Forest and Extra Trees provided a good balance between computational efficiency and detection accuracy.

7.5 Threats to Validity

Several factors could threaten the validity of this study. The quality and representativeness of the labeled data play a crucial role in the performance of supervised models. If the labeling process is flawed or inconsistent, it can lead to incorrect model training and evaluation. Mislabeling anomalies as normal data or vice versa can significantly affect the performance metrics of the models, resulting in poor detection accuracy and increased false positive or false negative rates. Ensuring high-quality labeling requires validation procedures and possibly multiple reviews by domain experts to minimize errors and biases in the labeled dataset.

The generalizability of the models across different products and test cases is a critical concern. While supervised learning models demonstrated strong performance on the specific products and test cases included in this study, their ability to generalize to other products and new test scenarios remains uncertain. Differences in product characteristics, use cases, and performance metrics can affect the models' detection capabilities. Further validation on diverse datasets is essential to ensure the models remain effective across various conditions. Additionally, the applicability of the models in real-world scenarios is another important consideration. Although the solution showed promising results in controlled evaluation and validation settings, the performance in live environments may still differ. In order to keep the solution effective in real-world use, maintenance is important which requires robust monitoring and evaluating of the results from time to time to be able to identify and address any negative performance changes.

A significant concern regarding the results presented in this study and the performance of the solution lies in the potential for implementation errors that may have influenced the conclusions drawn from this work. Despite the authors' meticulous efforts to ensure correctness, there remains a non-trivial probability and a looming threat concerning the validity of the presented results and solution.

Overfitting is a potential threat, particularly in supervised learning, where models may become too specialized to the training data and fail to generalize to new data. This can occur if the models are overly complex or if the training dataset is not sufficiently diverse. Overfitting can lead to high accuracy on the training data but poor performance on unseen data, reducing the practical applicability of the models. The extended rolling window cross-validation conducted through this study had the aim to mitigate the threat of overfitting to some degree as this validation examines if the solution is general enough to be used on upcoming unseen data.

The feedback mechanism is a critical component of the supervised learning approach. However, human factors such as cognitive biases, inconsistencies in judgment, and fatigue can affect the accuracy and reliability of the feedback. Inconsistent feedback can lead to erroneous adjustments in the model, affecting its performance. Due to time limitations, we could not validate how the feedback mechanism would perform over time. Instead, we attempted to simulate this by conducting an extended rolling window cross-validation with incremental training. This approach provided insights into how the model might adapt to changes in detection criteria. However, it is important to note that this simulation is not equivalent to validating the feedback mechanism over time, as the feedback is only based on the already annotated data from two test runs in each increment.

These considerations address **RQ1**, **RQ2**, and **RQ3** collectively, as ensuring the validity of the models' performance directly impacts their efficiency, their ability to assist current work processes, and their adaptability to evolving needs. Addressing these threats through rigorous validation, continuous monitoring, and iterative improvement ensures that the proposed solution remains robust and reliable.

7.6 Social & Ethical Aspects

By streamlining the performance data analysis process, the study aims to alleviate the manual workload of quality engineers at the case company, thereby improving their work-life balance. Reducing the manual burden allows engineers to focus on more critical tasks, potentially increasing job satisfaction and reducing stress. While automation aims to reduce the workload of quality engineers, there is an ethical consideration regarding its impact on employment. It is important to ensure that the introduction of such tools does not lead to job losses but rather supports engineers in their roles. Offering training and development opportunities to help engineers adapt to using advanced tools can mitigate any negative organizational and employment impacts.

Quality engineers should be able to get a clear understanding of how the tool identifies anomalies and the criteria it uses as establishing clear accountability for the tool's outcomes is necessary. Engineers should be able to review and challenge the tool's decisions, ensuring that the final judgment is done by human experts.

By considering these aspects, the project attempts to contribute positively to the societal and ethical context beside the technical contributions.

7.7 Future Work

A critical aspect of future work involves evaluating and improving the feedback mechanism. A more systematic evaluation is needed to understand how this feedback mechanism performs over time and impacts the process. Future efforts should focus on developing more sophisticated methods for incorporating feedback, this could involve exploring advanced techniques for weighting feedback based on the engineer's confidence level or incorporating semi-supervised learning approaches to better utilize both labeled and unlabeled data.

Integrating the various components of the solution into a cohesive automated pipeline is a crucial next step. Currently, the solution involves multiple stages, including data collection, preprocessing, annotation, detection, and feedback incorporation. Automating these processes can significantly enhance efficiency and scalability. This integration should ensure seamless data flow between components. Additionally, developing an improved user interface for annotation and feedback tool can contribute to improvement as well.

Advanced feature engineering is important for improving model performance. Future work should focus on identifying and engineering features, together with the help of the organization, to capture more relevant aspects of the performance data and potentially define new features that summarize and represent these in a better way. Such features could provide more opportunities regarding using an unsupervised learning approach as a solution due to the reduced complexity of the data.

7.8 Conclusion

This study investigated various approaches for assisting quality engineers in detecting performance abnormalities through statistical thresholds, unsupervised, and supervised learning methods. After observing weak performance of statistical thresholds, unsupervised learning was investigated, which showed better detection capability. However, it faced significant challenges such as high false positive rates and computational demands. Despite enhancement attempts, the precision of the unsupervised methods remained insufficient for practical application without extensive manual review.

In contrast, supervised learning models, particularly RandomForest, demonstrated better performance across all evaluation metrics. The implementation of the feedback mechanism allowed for continuous detection improvement based on real-world insights, which showed a 15% reduction in the false positive rate, according to a comparison of validation results in Section 6.1. The solution, consisting of the RandomForest model and the feedback mechanism, presented potential generalization capabilities, effectively distinguishing between normal and anomalous behavior across various performance metrics, test cases, and products.

The findings address the research questions by highlighting the following:

- **Efficiency in Detection (RQ1):** A solution based on supervised machine learning with continuous feedback can provide an efficient and effective approach for detecting performance abnormalities through analyzing test data.
- **Assisting Current Work Processes (RQ2):** The integration of the solution has the potential to assist quality engineers by reducing their workload over time and improving the efficiency of the review process. This is achieved by reducing the false positive rate and only alerting engineers to problematic test cases that need to be reviewed.
- **Adaptability for Evolving Needs (RQ3):** The feedback mechanism allows the solution to adapt to evolving quality expectations and new data characteristics, ensuring ongoing relevance and effectiveness.

Future work is supposed to focus on further evaluating the feedback mechanism, integrating and automating the solution components into a cohesive pipeline, and employing advanced feature engineering refinements. These efforts aim to develop a more sophisticated and effective detection system that meets the evolving needs of the QA team and the broader requirements of the system.

References

[1] P. Runeson, E. Engström, and M.-A. Storey, “The Design Science Paradigm as a Frame for Empirical Software Engineering,” in *Contemporary Empirical Methods in Software Engineering*, M. Felderer and G. H. Travassos, Eds. Cham: Springer International Publishing, 2020, pp. 127–147.

[2] D. V. Thiel, *Literature search and review.* Cambridge University Press, 2014, p. 27–72.

[3] O. A. Adeoye-Olatunde and N. L. Olenik, “Research and scholarly methods: Semi-structured interviews,” *JACCP: Journal of the American College of Clinical Pharmacy*, vol. 4, no. 10, pp. 1358–1367, 2021.

[4] H. Kallio, A. Pietilä, M. Johnson, and M. Kangasniemi, “Systematic methodological review: Developing a framework for a qualitative semi-structured interview guide.” *Journal of Advanced Nursing*, vol. 72, no. 12, pp. 2954 – 2965, 2016.

[5] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002.

[7] H. Yoon, “Finding unexpected test accuracy by cross validation in machine learning,” *International Journal of Computer Science & Network Security*, vol. 21, no. 12spc, pp. 549–555, 2021.

[8] C. U. Smith and L. G. Williams, *Software performance engineering.* Springer, 2003.

[9] R. Pressman and B. Maxim, *Software Engineering: A Practitioner’s Approach.* McGraw-Hill Education, 2020.

[10] E. Weyuker and F. Vokolos, “Experience with performance testing of software systems: issues, an approach, and case study,” *IEEE Transactions on Software Engineering*, vol. 26, no. 12, pp. 1147–1156, Dec. 2000.

[11] C. Patel and R. Gulati, "Identifying ideal values of parameters for software performance testing," in *2015 International Conference on Computing, Communication and Security (ICCS)*, 2015, pp. 1–5.

[12] M. Fagerström, E. E. Ismail, G. Liebel, R. Guliani, F. Larsson, K. Nordling, E. Knauss, and P. Pelliccione, "Verdict machinery: on the need to automatically make sense of test results," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016. Association for Computing Machinery, 2016, pp. 225–234.

[13] D. Daly, W. Brown, H. Ingo, J. O'Leary, and D. Bradford, "The Use of Change Point Detection to Identify Software Performance Regressions in a Continuous Integration System," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '20. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 67–75.

[14] M. Fleming, P. Kolaczkowski, I. Kumar, S. Das, S. McCarthy, P. Pattabhiraman, and H. Ingo, "Hunter: Using change point detection to hunt for performance regressions," in *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '23. Association for Computing Machinery, 2023, pp. 199–206.

[15] A. Bauer, M. Straesser, L. Beierlieb, M. Meissner, and S. Kounev, "Automated triage of performance change points using time series analysis and machine learning: Data challenge paper," in *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '22. Association for Computing Machinery, 2022, pp. 29–32.

[16] T. H. D. Nguyen, M. Nagappan, A. E. Hassan, M. Nasser, and P. Flora, "An industrial case study of automatically identifying performance regression-causes," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, May 2014, pp. 232–241.

[17] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, J. Guo, C. Sporea, A. Toma, and S. Sajedi, "Using black-box performance models to detect performance regressions under varying workloads: an empirical study," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4130–4160, Sep. 2020.

[18] J. Chen, "Performance regression detection in DevOps," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 206–209.

[19] T. Mendes, P. J. S. Cardoso, J. Monteiro, and J. Raposo, "Anomaly Detection of Consumption in Hotel Units: A Case Study Comparing Isolation Forest and Variational Autoencoder Algorithms," *Applied Sciences*, vol. 13, no. 1, p. 314, Jan. 2023, number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[20] N. Mejri, L. Lopez-Fuentes, K. Roy, P. Chernakov, E. Ghorbel, and D. Aouada, "Unsupervised anomaly detection in time-series: An extensive evaluation and analysis of state-of-the-art methods," *arXiv preprint arXiv:2212.03637*, 2022.

[21] J. Vávra, M. Hromada, L. Lukáš, and J. Dworzecki, "Adaptive anomaly detection system based on machine learning algorithms in an industrial control environment," *International Journal of Critical Infrastructure Protection*, vol. 34, p. 100446, Sep. 2021.

[22] Z. K. Shahid, S. Saguna, and C. Åhlund, "Autoencoders for Anomaly Detection in Electricity and District Heating Consumption: A Case Study in School Buildings in Sweden," in *2023 IEEE International Conference on Environment and Electrical Engineering and 2023 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*, Jun. 2023, pp. 1–8.

[23] E. Aronsson, "Unsupervised Anomaly Detection in Multivariate Time Series Using Variational Autoencoders," *Master's Theses in Mathematical Sciences, Lund University*, 2023, iD: 9135876.

[24] C. Jernbäcker, "Unsupervised real-time anomaly detection on streaming data for large-scale application deployments," *Independent thesis Advanced level, KTH, School of Electrical Engineering and Computer Science (EECS)*, 2019.

[25] Y. Zhao, X. Hu, C. Cheng, C. Wang, C. Wan, W. Wang, J. Yang, H. Bai, Z. Li, C. Xiao, Y. Wang, Z. Qiao, J. Sun, and L. Akoglu, "Suod: Accelerating large-scale unsupervised heterogeneous outlier detection," in *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica, Eds., vol. 3, 2021, pp. 463–478.

[26] S. Das, W.-K. Wong, A. Fern, T. G. Dietterich, and M. A. Siddiqui, "Incorporating Feedback into Tree-based Anomaly Detection," *arXiv*, Aug. 2017, arXiv:1708.09441 [cs, stat].

[27] V. Vercruyssen, W. Meert, G. Verbruggen, K. Maes, R. Baumer, and J. Davis, "Semi-Supervised Anomaly Detection with an Application to Water Analytics," in *2018 IEEE International Conference on Data Mining (ICDM)*. Singapore: IEEE, Nov. 2018, pp. 527–536.

[28] D. Zha, K.-H. Lai, M. Wan, and X. Hu, "Meta-aad: Active anomaly detection with deep reinforcement learning," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 771–780.

[29] A. Hrusto, P. Runeson, and E. Engström, "Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations," *SN Computer Science*, vol. 2, no. 6, p. 447, Sep. 2021.

[30] A. Hrusto, E. Engström, and P. Runeson, "Towards optimization of anomaly detection in DevOps," *Information and Software Technology*, vol. 160, p. 107241, Aug. 2023.

[31] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence," *Information*, vol. 11, no. 4, p. 193, 2020.

Appendices

Appendix A

A.1 Author Contribution Statement

The authors, A.H. and S.T., contributed equally regarding all parts of the study e.g. exploration, design, implementation, evaluation and documentation (the report). This was done for instance through continuous collaboration and pair programming. A.H. focused more on the implementation of the presented annotation and feedback tools while S.T. only contributed with design ideas. S.T., on the other hand, focused more on the area of studying performance regression detection and its challenges.

Appendix B

B.1 Interview Questions

Here are the interview questions used in Section 4.1:

1. How do you perceive the problem, and how does it affect you personally and professionally?
2. What characteristics define a flagged regression in your experience?
3. Can you describe the different types of regressions you encounter?
4. How do you typically handle these regressions?
5. What is your process for addressing a flagged anomaly?
6. How many products are you currently managing? Are there specific products that frequently encounter issues?
7. How do you manage tests involving multiple streams? What factors are important in these tests? Do such tests usually encounter problems?
8. Which parameters or measurements do you consider the most critical?
9. Do you often notice relationships between different measurements?
10. How do you monitor data over longer periods? Do you use daily averages? How do you calculate these averages?
11. How do you review measurement values, and how precise are you in this review? Do you round off the values?
12. *(To Quality Engineer)* Do you ever revisit and review old test runs? If so, why and how often?
13. In your opinion, which of the testing levels are more critical? builds or firmwares?
14. How would you like the current process to be improved? What features or capabilities would you like to have?
15. What do you anticipate will happen when a new product is added to the system?

B.2 Annotation Evaluation Questions

1. What do you think about the annotation tool?
2. Describe how easy is it for you to use the tool.
3. Describe how it can be compared to the existing process/tools.

B.3 Detection & Feedback Evaluation Questions

1. Generally, what do you think about the detection and feedback functionalities?
2. Describe how the solution can be used by you.
3. Describe how the use of this tool can affect your work process.
4. How do you think you can adjust the detection upon evolving needs?
5. How adaptable is the solution?

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2024-06-12

EXAMENSARBETE Enhanced Techniques for Detecting Performance Abnormalities in Software Quality Assurance Processes
STUDENTER Abdulrahman Husari, Sepehr Taherpour
HANDLEDARE Per Runeson (LTH)
EXAMINATOR Björn Regnell (LTH)

Automatisera upptäckter av avvikande beteenden i testresultat

POPULÄRVETENSKAPLIG SAMMANFATTNING **Abdulrahman Husari, Sepehr Taherpour**

Att säkerställa mjukvarukvalitén i tekniska produkter som nätverkskameror kräver omfattande mjukvarutestning, vilket genererar stora mängder data. Detta arbete undersöker metoder för att detektera problem i testresultat på ett mer automatiskt sätt för en förbättrad process.

Den huvudsakliga frågan som vi letar svar på är hur man skulle kunna göra livet enklare för en kvalitetsingenjör som sitter dagligen och analyserar testresultat i form av grafer. Detta arbete är tidskrävande och tråkigt. Analysen av denna typ är en långsam och manuell process, vilket blir svårare när antalet tester ökar. Frågan är om datorbaserade metoder kan användas för att underlätta detta.

Målet med detta arbete var att skapa en lösning som hjälper kvalitetsingenjörer att snabbt och noggrant identifiera avvikande beteenden i prestandatestdata. Prestanda innebär hur bra något fungerar eller utförs, och i detta sammanhang handlar det om att testa hur väl en produkt, som till exempel en nätverkskamera, fungerar under olika förhållanden. Prestandatestning är avgörande för att hitta problem som kan påverka produktens kvalitet och tillförlitlighet. Förseningar i att identifiera prestandaproblem kan leda till långsammare utveckling och att potentiellt felaktiga produkter når marknaden.

Genom att använda datorbaserade tekniker kan upptäckter av problem automatiseras, vilket minskar det manuella arbetet och ökar effektiviteten i processen. Automatiseringen kan bland annat göras med hjälp av statistiska metoder, men även genom användning av maskininlärning. Våra experiment visade att vissa metoder fungerar bättre i detta arbete. Vägledd maskininlärning är en metod som fungerar igenom inmatning av märkta data, indata och förväntad utdata, med syfte av att träna datorn att upptäcka saker enligt det inlärda mönstret. Användning av denna metod visade bäst resultat i det här arbetet.

För att kunna använda en lösning baserad på vägledd maskininlärning, behövdes det andra verktyg för att märka datan, visualisera resultat och lämna återkoppling på resultat. Detta arbete har även tagit fram dessa tillhörande verktyg för att möjliggöra användning av lösningen på fallföretaget. Detta användarcentrerade tillvägagångssätt säkerställer att lösningen inte bara är tekniskt effektiv utan också praktisk och enkel att använda i verkliga scenarier.

Detta examensarbete representerar ett steg framåt i att automatisera kvalitetssäkringsprocessen vid fallföretaget. Det fungerar även som ett exempelarbete inom detta område för vidare utforskning. Genom att använda dessa datorbaserade tekniker kan granskning av prestandatestresultat bli mer effektiv, vilket i slutändan leder till bättre kvalitetssäkringsprocess.