CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Applicability of Supervised Machine Learning for CI Configuration Selection

A Case Study in the Telecom Industry

Master's thesis in Computer science and engineering

ALBIN LÖNNFÄLT & VIKTOR TU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Applicability of Supervised Machine Learning for CI Configuration Selection

A Case Study in the Telecom Industry

ALBIN LÖNNFÄLT & VIKTOR TU

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Applicability of Supervised Machine Learning for CI Configuration Selection
A Case Study in the Telecom Industry
ALBIN LÖNNFÄLT & VIKTOR TU

Supervisor: Gregory Gay, Department of Computer Science and Engineering
Advisor: Sahar Tahvili, Ph.D., Ericsson AB
Examiner: Lucas Gren, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Applicability of Supervised Machine Learning for CI Configuration Selection
A Case Study in the Telecom Industry
Albin Lönnfält & Viktor Tu
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This study introduces a novel supervised machine learning (ML) model for accurately assigning CI configurations to test specifications. Current solutions to optimize selection of CI configurations lack the ability to select CI configurations for individual test cases and assigning them into predefined CI configurations. The model employs an ensemble architecture with three sub-models and a rule-based component, each focusing on specific aspects of the problem. Extensive model analysis reveals important features that contribute to the assignment process. A decision support system based on the ML model is developed to evaluate the applicability of supervised ML in CI configuration assignment, validated through a survey study involving domain experts. The study demonstrates that supervised ML can exceed the performance requirements of domain experts. Certain features in test specifications are found to be influential in the assignment outcome. Implementing supervised ML brings business value, reducing misassignments, saving time, and reducing fault slip through. Proposed future research includes exploring fully automated CI configuration assignments and investigating more complex ML models, such as neural networks, for enhanced performance and exploring the potential for fully automated adaptation.

Keywords: Software testing, continuous integration, continuous integration configuration, and supervised machine learning.

Acknowledgements

We express our gratitude to Gregory Gay, our academic supervisor, for his guidance and feedback during this thesis. We also thank Sahar Tahvili, our industrial supervisor, and all the Ericsson employees involved. Lastly, we thank our families and friends for their support throughout this journey.

Albin Lönnfält & Viktor Tu, Gothenburg, June 2023

Contents

List of Figures

List of Tables

1

Introduction

Software testing is a critical process in software development that involves the evaluation of a software product or system to identify defects and ensure that it meets specified requirements. Testing is crucial since it helps ensure software quality and reliability, reduces the risk of costly errors and failures [1, 2], and improves user satisfaction and trust. Effective software testing can also increase efficiency and productivity by identifying and addressing issues early in the development cycle, reducing rework, and minimizing the time and resources required to deliver a high-quality product [2]. Due to the importance of rigorous software testing, it is estimated that approximately 30% to 50% of the development effort is constituted of testing [3, 4].

As software products grow and become increasingly complex, integrating and testing software has become increasingly challenging [5, 6]. In response to this, continuous integration (CI) has risen in popularity both in industry [7, 8, 9, 10] and academic research [9, 10] the last decade. CI is a methodology in software development that emphasizes frequent and seamless integration of code changes. The CI process includes automatically compiling, testing, packaging, and deploying the code. In CI, developers integrate their modifications into the code base daily, promptly addressing potential issues. As a result, CI ensures that the code base remains up to date and ready for release at any given moment [11]. This approach eliminates the necessity for large-scale integration events, making identifying and resolving bugs significantly more efficient. Case studies have indicated that adopting CI can reduce the overall cost of integration and testing by 40% [7, 12].

CI brings numerous benefits to software development but also presents internal challenges that require careful attention [13]. As the code base grows, scalability issues arise, resulting in longer build times. Longer build times reduce the frequency of commits and hamper the overall efficiency of the development process. To address these scalability concerns, tailored CI configurations have been introduced. By configuring the build to test only the parts of the code affected by the changes and only including relevant test levels for the affected stakeholders, it is possible to reduce build time without compromising quality [14].

However, selecting the correct CI configuration is not a trivial task. This selection poses a significant challenge for development teams, and misconfigurations can have severe consequences. Incorrect CI configurations often lead to prolonged de-

velopment cycles and delayed product releases [15]. The time and effort wasted in diagnosing and resolving frequent build failures can harm the project's success [16]. This highlights the importance of addressing scalability issues in CI effectively.

Given the critical importance of software integration and testing [4, 17, 12], optimizing the CI process becomes highly relevant. Previous research has proposed various methods to enhance CI efficiency by optimizing CI configurations. However, thus far, no solution has explored the potential of utilizing supervised machine learning (ML) to improve the selection process of appropriate CI configurations. This thesis aims to explore the mentioned gaps and shed light on the applicability of ML methods to suggest the choice of CI configurations automatically.

This study is conducted in collaboration with the CloudRAN division at Ericsson, a Swedish telecom company. CloudRAN is a radio access network architecture that utilizes cloud computing to centralize and virtualize the baseband processing functions of mobile networks. Collaborating with Ericsson CloudRAN as a case study for this thesis carries several notable advantages. Firstly, it benefits from a clearly defined CI workflow with distinct configurations. Additionally, Ericsson has accumulated significant relevant data over an extended period, resulting in a relatively large annotated dataset that can be leveraged for comprehensive research.

1.1 Problem Description

Many well-known companies have successfully adopted the CI methodology since its inception. For example, CI adoption helped reduce development costs by 78% in a product group at HP and enabled Flickr to deploy production code more than ten times per day [18]. Despite these successful implementations [13], there are several pitfalls associated with the use of CI. One such challenge is determining the appropriate CI configuration to use.

The correct CI configuration is critical in achieving high efficiency in the CI implementation and realizing the financial and management benefits [16]. One of the most important aspects of the CI methodology is the build time since it requires frequent commits. Reducing build time enables more efficient use of the developer's time and thus improves the software development process [11]. One way of reducing the build time, and therefore the time between committing the code and receiving feedback, is by using correct CI configuration [19]. The CI methodology can also impact developer productivity through CI build failures. These failures often occur due to incorrect CI configurations and can waste valuable developer time. Debugging these issues can also be time-consuming, further adding to the ineffectiveness of the software development process [15]. The value of choosing the correct CI configuration and increasing the developer's efficiency increases as the scale of the software development grows. Therefore, it is crucial to select correct CI configurations to minimize the occurrence of build failures and maximize the productivity of the development team.

Another critical aspect related to CI is the selection of a subset of a test suite for execution during testing. As software projects grow in size or undergo frequent changes, testing can become time-consuming and encounter scalability challenges. One of the main culprits behind these challenges is test redundancy, which occurs when software functionalities are unnecessarily retested, despite remaining unaffected by the latest commits. This redundancy becomes more pronounced in large-scale software, leading to significant inflation in the size of test suites. Therefore, selecting an appropriate test suite is an important aspect when configuring a CI pipeline [19].

Many settings can be configured in a CI pipeline. In this study, a CI configuration relates to the following specific aspects of the CI process:

- **Execution Environment**: Individual microservices are tested in isolation in a decentralized environment. In contrast, in a centralized environment, multiple microservices are allowed to execute, and their interactions can be tested.
- **Test Suite**: Which test cases to execute in a build. Some test case types, e.g., unit versus integration tests, may be more appropriate in some configurations than in others.
- **Build Scheduling**: Build scheduling refers to when the build process occurs. It involves deciding whether multiple commits should be bundled together and built nightly or if a build should occur promptly after each commit is made.

Rather than configuring these options independently, we focus on pre-defined configurations of these four major aspects of the CI process in this study.

1.2 Purpose of the Study

This study aims to improve the efficiency of software testing in CI by offering valuable insights into the selection of a suitable CI configuration for a given test specification. In this study, a test specification refers to a written document outlining the steps and conditions required to execute a test case. It includes detailed test instructions written in natural language and categorical data describing the testing environment, such as the system under test, the tagged microservice, the traffic model, and the test framework. The primary objective of this thesis is to improve the software testing procedure by helping developers identify the appropriate CI configurations under which a particular test specification should be executed.

Moreover, this study focuses on the practical usability of a system built on supervised machine learning for this purpose. To achieve this, we develop a system based on supervised ML techniques and evaluate its real-world application in industry. The study aims to provide concrete evidence of its practical value and suitability for integration into existing CI workflows by assessing this system's performance, effectiveness, and feasibility in realistic scenarios. Additionally, this study hopes to contribute to academic research by filling a crucial gap in understanding how

supervised machine learning can be effectively employed in the context of CI configuration.

Specifically, the study aims to answer the following questions:

- **RQ1:** What performance can supervised ML achieve when classifying test specifications into the optimal pre-defined CI configurations?
- **RQ2:** What are the critical features for classifying test specifications into the optimal pre-defined CI configurations?
- **RQ3:** What is the practical applicability of ML methods for classifying test specifications to pre-defined CI configurations in practice?

1.3 Significance of the Study

This study makes dual contributions, both in the scientific and practical realms.

The study provides a scientific contribution by advancing the understanding of critical features for assigning CI configurations to test specifications. Moreover, this study fills the knowledge gap in the current literature regarding the design of supervised ML systems to assist decision-making in the CI process. The study also sheds light on the achievable performance of using supervised ML for this purpose.

This thesis makes a significant practical contribution by offering valuable insights that can assist practitioners in enhancing their testing procedures. The study serves as a comprehensive guide for practitioners looking to implement supervised machine learning techniques for assigning CI configurations. Furthermore, even for those who plan to utilize something other than supervised ML in their CI implementation, the study provides insightful information to improve the overall effectiveness of the integration and testing process.

By incorporating the recommendations and lessons learned from this study, practitioners can streamline their testing procedures, minimize inefficiencies, and ensure a more effective overall integration and testing process. This thesis serves as a valuable resource, equipping practitioners with practical knowledge to optimize their testing procedures, regardless of whether they choose to utilize supervised machine learning techniques in their CI implementation.

1.4 Thesis Outline

The organization of this thesis is laid out as follows:

Chapter 2 (Background) presents relevant terminology and concept for this study.

Chapter 3 (Related Work) presents other studies related to this study.

Chapter 4 (Methodology) describes the methodology of this study.

Chapter 5 (Results) presents the results of this study.

Chapter 6 (Discussion) discusses the results of this study.

Chapter 7 (Conclusion) presents the conclusions of this study.

2

Background

This chapter will provide the background knowledge necessary to understand the topics of this thesis. This section will first look into the concept of software requirements, and then it will dive into software testing. Furthermore, this chapter provides a comprehensive overview of CI as a methodology. After that, a brief overview of ML and natural language processing (NLP) will be presented.

2.1 Software Requirements

Software requirements refer to the needs and constraints a software needs to meet to satisfy real-world applications. These requirements play an essential role in the software development process. Many regard the most challenging part of building a software system as deciding what to develop. Moreover, starting with incorrect or incomplete software requirements can be difficult to address later in the development cycle [20].

One way to categorize software requirements is based on scope and purpose. Three distinct levels of software requirements within scope and purpose exist. They are business, user, and functional requirements. In addition to these three levels of software requirements, every system has non-functional requirements, which are different types of quality attributes that the system must adhere to (e.g., performance constraints) [20].

Another way software requirements can be categorized into are product requirements and project requirements. Product requirements describe the characteristics or properties of the software that will be built. On the other hand, project requirements are associated with the software development project itself and include resource requirements, staff training needs, compliance requirements, and customer service-level agreements [20].

For this thesis, the focus will be on product requirements, notably functional and non-functional product requirements.

2.1.1 Functional Requirements

Functional requirements are meant to capture the intended behavior of a system. These behaviors can be in the form of services, tasks, or functions [21, 22, 23]. In other words, functional requirements define the necessary tasks, actions, or activities that a software should be able to accomplish [23, 24, 25]. Examples of functional requirements could be that the software should be able to modulate a signal or format some text. Functional requirements can sometimes be referred to as capabilities [25].

There are several forms that functional requirements can be captured. A common practice for capturing functional requirements is use-cases [21, 26]. A use-case defines a goal that an actor wants to accomplish when interacting with the software. In theory, a complete set of use cases should specify all desired functionalities that a software should have [21].

2.1.2 Non-functional Requirements

Unlike functional requirements, there is no consensus in research on the exact definition of non-functional requirements [23, 27, 28]. A frequently used definition of non-functional requirement is that non-functional requirements are all requirements that are not related to functional requirements [25].

Common ways to define non-functional requirements include terms such as properties, attributes, qualities, constraints, and performance. These definitions usually revolve around the idea that the particular software should fulfill and meet certain constraints on *how* it operates [23, 27]. Among common non-functional requirements are performance, reliability, security, and capacity [21, 23, 25].

Non-functional requirements can act as constraints on a system. This happens when functional requirements must be sacrificed to satisfy a non-functional requirement such as performance [25].

2.2 Software Testing

Software testing is the act of executing a given program with the goal of either finding errors or ensuring that no errors exist [20]. In other words, software testing ensures that the software works as intended and that the expected requirements are met. However, it must be noted that software testing can only prove the presence of errors and unintended behavior and not the absence of those because it is impossible to test all possible inputs for any system of reasonable complexity [29]. Testing can be performed at multiple levels of granularity in a system, including unit, integration system testing, and acceptance testing [3, 30]. The different levels of software testing will be discussed later in the study.

Software testing is an activity that is often overlooked in software development, although it makes up a large share of the total effort needed in a software development project. The generally accepted belief is that software testing constitutes between

30% to 50% of the total work in a given software development project [3].

2.3 Levels of Software Testing

2.3.1 Unit Testing

Unit testing revolves around testing individual units or groups of related units, usually testing a minor separable component of a system [31]. This is called the "unit" in unit testing. An example of a unit in an object-oriented program is an individual method or a class [29, 32]. In practice, unit testing is performed by executing a part of the code separately and comparing the actual outcome with the intended outcome of the code [29]. The actual outcome being the same as the expected outcome means that the code has passed the unit test, and all other cases mean it has failed. Even though unit testing has received criticism for the time it requires and the perceived costs of thorough implementation, it is still considered an effective tool for two reasons. The first reason is that it is effective in testing for boundary value behavior in separate components, and the second reason is to guarantee that the code has been sufficiently executed [3].

Almost all popular programming languages have their built-in unit testing framework; examples are JUnit for Java and NUnit for C#. These built-in unit testing frameworks enable the use of automatically executed unit tests [33]. Using unit testing and built-in unit testing frameworks is beneficial in two ways. It can potentially improve the design and significantly decrease the time spent on solving mysterious bugs [29].

2.3.2 Integration Testing

Integration testing in the context of software testing is the testing in which software components are combined and tested in order to evaluate the interaction amongst them [34, 35]. In contrast to unit testing, integration testing focuses on testing at a module level instead of at a statement level. This puts a higher degree of emphasis on the interaction between different software components [35]. Since individual components are grouped and tested in tandem, the SUT in integration testing will be more complex than the SUT during unit testing [36]. Integration testing assumes that all components included in the integration test already have been unit tested and thus have guaranteed intended behaviour [37].

Several known approaches or strategies for integration testing exist, e.g., big-bang integration and pairwise integration. Big-bang integration is simply taking all units and testing their interaction simultaneously, while pairwise integration tests the interaction of a particular unit and its adjacent units [37].

2.3.3 System Testing

The last internal level of software testing is called system testing and concerns testing a whole system's compliance with its specifications [38]. In a system test, all components that the SUT is comprised of are involved [39]. Usually, system testing is performed via APIs or a GUI through defined top-level interfaces. System testing can involve different testing activities such as performance testing, where, for instance, a system's resource utilization and response time are tested, and also functional testing, where a system's behavior are tested [40]. Furthermore, system testing presumes that the components going into to system test have passed integration testing [38]. However, in reality, this is only sometimes the case.

System testing has three primary goals. Firstly to reveal bugs and issues that only are present at a system level scope and thus have not been captured during unit and integration testing. Secondly, to ensure that the SUT has met all the specified requirements. In this case, requirements could be capabilities, features, or functions. Furthermore, lastly to help answer the question of whether the program is ready for release or not [38].

2.3.4 Acceptance testing

Acceptance testing is a crucial phase in software development to verify if a system or software meets the predefined requirements and is ready for delivery to end-users or clients. This type of testing is typically conducted by customers or end-users themselves, making it an integral part of the overall testing process [38, 41].

Acceptance testing is the final stage in the testing process after the completion of development and preceding testing phases. Its primary objective is to ensure that the software aligns with the expectations and needs of the intended users. Customers or end-users evaluate the software's functionality, usability, performance, and compliance by subjecting it to realistic usage scenarios [38, 41].

2.4 Software Integration

2.4.1 Traditional Integration

Traditional waterfall-like development processes have a distinct "integration and test" phase after the implementation phase. Once all the software modules are implemented, they are combined and tested to identify any remaining issues despite planning. This approach causes issues, especially in agile processes with minimal upfront design. Late integration in a project or iteration leads to the late detection of certain faults, such as technical problems, interface mismatches, and defects due to incorrect assumptions and misunderstandings between developers. This makes the integration phase challenging to predict and leads to project delays, as the later the defects are found, the costlier they become [42].

2.4.2 Continuous Integration (CI)

Continuous Integration (CI) is a widely adopted software development practice involving frequent and automated integration of code changes from multiple developers into a shared codebase. This practice has gained popularity due to its ability to ensure high code quality and promote efficient collaboration among team members.Within the realm of software development, the significance of CI stems from its capability to detect and rectify issues at the earliest stages of the development process, resulting in accelerated development cycles and, ultimately, an enhanced software development procedure. [11, 43].

CI entails committing changes daily to the mainline code repository. The updated codebase is rebuilt and tested by an automated test suite as part of a "pipeline" or "workflow". A CI pipeline or workflow refers to a streamlined process that guides the progression of building, testing, and deploying code [44, 16, 8, 9]. A pipeline encompasses a range of tasks, such as code compilation, unit testing, code analysis, security checks, and binary generation. In containerized environments, the pipeline would also encompass bundling the code into a container image that can be deployed across hybrid clouds [44].

Any problems are resolved immediately, ensuring that the code is always in a releasable state. To implement CI effectively, it is important to have a well-designed automated test suite covering all relevant codebase areas. The tests should run quickly and be easily maintainable, allowing developers to make changes and updates as needed. Additionally, the CI process should be well-documented and easily repeatable so that developers can quickly identify and resolve any issues that may arise [11, 43].

By committing changes daily to the mainline, developers can ensure that the codebase is always up-to-date and that everyone is working with the latest code. To further enhance this process, an automated build on an integration server can be implemented to avoid machine-specific errors that can occur when developers work on different machines. This ensures that the code is being built and tested in a consistent environment, reducing the risk of errors or bugs arising from different development setups [11, 43].

Another key component of CI is the use of automated builds of the software. The build can be run automatically whenever changes are committed, helping to identify any issues or bugs that may have been introduced. By using a automated build, developers can save time and ensure that all code changes are thoroughly tested. If any build fails, the problem is resolved immediately, ensuring that the code base remains in a releasable state at all times [11].

2.4.2.1 Benefits of CI

One of the primary benefits of CI is that it enables early detection of defects. By integrating code changes frequently, developers can identify issues early in the development cycle before they become more complex and expensive to fix. This reduces

the overall cost of development and ensures that defects are addressed before they reach production [11].

CI has proven these benefits by reducing development costs and improving deployment frequency for various companies. For instance, at HP, CI contributed to a cost reduction of 78% within a specific product group. Similarly, Flickr, a prominent online photo-sharing platform, experienced remarkable improvements in its deployment process through CI. By adopting CI methodologies, Flickr could deploy production code more than ten times daily. This accelerated deployment frequency demonstrates the agility and streamlined workflow that CI brings to software development organizations [13]. Furthermore, a Microsoft case study concluded that CI reduces the check-in overhead by at least 40% [7].

Another benefit associated with CI is the possibility of getting a more transparent overview of the progress of a project. When CI is not adapted, it is often difficult for managers to estimate the time needed for the final integration, imposing the risk of delivery delays. Using CI makes it easier for managers to forecast the progress and release of products and features [11].

Furthermore, CI removes the most significant barriers to frequent deployment. By frequent deployment, users can get new features quicker, provide feedback on the feature, and be a more integrated part of the development process [11].

2.4.2.2 Challenges of CI

CI is, as previously discussed, associated with several benefits. The implementation and use of CI can, however, be challenging. Seven categories of challenges related to the use of CI have been identified in academic research [13];

1. **Mindset:** For an efficient use of CI, the mindset of the software developers plays a crucial role. The openness to changes and a clear understanding of the benefits of CI are pinned down as a vital aspect when implementing CI.

2. **Tools and Infrastructure:** Organizations that adapt CI often suffer from insufficient code review and integration tools. Furthermore, the build and execution time of regression tests is challenging, given the risk of long feedback loops.

3. **Testing:** Unstable test cases have been highlighted as a critical challenge associated with CI (i.e., tests that are likely to break).

4. **Domain applicability:** The suitability of CI depending on the domain has been discussed in the literature, and concerns that the CI is not suitable for complex products where code is merged to multiple branches have been identified. However, some believe that the concerns are related to the confidence of teams and the tools at their disposal rather than CI itself.

5. **Understanding:** Dissonance of the interpretation of the concepts and objec-

tives of CI between development teams and management has been identified, posing a challenge when adapting CI. The expectations and visualized goals differ between members of the adapting organization, which leads to organizational challenges when adapting CI.

6. **Code dependencies:** Work that would be benefited from being developed and integrated at a particular planned point may be split up into several integrations when the CI methodology is adapted, which can impose inefficiencies in the development process.

7. **Software requirements:** Given the frequent integration in CI, it becomes necessary to break down requirements that were previously integrated on a sprint basis into smaller subparts that can be independently integrated. However, this process can introduce unnecessary overhead work and present challenges in identifying suitable subparts of the appropriate size that can be easily integrated.

2.4.2.3 CI Configuration

There are many definitions of CI configurations in academic research. One definition is that the CI configuration outlines the necessary steps for building the code and executing a test suite. The CI configuration establishes build conditions, such as the operating system, disk size, compiler flags to utilize, required library dependencies, and other analogous properties [15].

In this study, CI configuration refers to a set of specific settings and parameters for a CI pipeline. These configurations are influenced by various factors that affect the cost and complexity of the pipeline, including the execution environment, contents of the test suite, build scheduling, and testing activities performed.

The contents of the test suite determine which tests will be executed during a given build, while the build scheduling specifies how often builds are run and how commits may be bundled into one or more builds. The execution environment, whether centralized or decentralized, determines whether microservices are tested independently or together, allowing for controlled testing of their interactions. Testing activities refer to the breakdown of testing activities based on whether they occur before or after code changes are merged into the main branch of the version control system.

Some forms of requirements (functional versus non-functional) may only be able to be verified, and some testing levels may only be able to be targeted, in certain CI configurations. More details about the specific CI configurations used in this study can be found in Section 4.2.1.

Incorrect CI configurations affect the software development process negatively, as incorrect CI configurations directly cause CI build failures. The consequences of CI build failures are longer development times and thus delayed product release dates [15]. Moreover, CI configurations are often the root of many bottlenecks in the CI pipelines [16].

2.4.2.4 Scalability Issues of CI

As a codebase grows in size and complexity, scalability issues within the CI process begin to emerge. One of the primary challenges is the increase in build time. Longer build times can significantly impact the development workflow, reducing the frequency of commits and impeding the team's overall productivity. Developers often face frustrating delays, waiting for the build process to complete before receiving feedback on their changes [14].

To counteract these scalability issues, tailored CI configurations have been introduced. These configurations aim to optimize the build process by selectively testing only the parts of the codebase affected by the changes. By focusing on specific areas and including only relevant test levels, teams can reduce the build time without compromising the quality of the software. This approach helps mitigate the impact of scalability issues and keeps the development workflow smooth [14].

2.5 Machine Learning

ML is a branch of artificial intelligence that focuses on developing programs, models, and computers that can replicate specific behaviors found in the data used to train them [45]. ML applications are many; some examples are hand gesture recognition, semiconductor fault detection, and mobile advertising [46]. ML also plays a crucial role in several subfields of computer science, including computer vision, natural language processing (NLP), and speech recognition. Moreover, the impact of ML has also spread to other industries with data-intensive issues, such as consumer goods and logistics [45].

2.5.1 Supervised Learning

Supervised learning is a type of ML that entails training a program, model, or computer a desired behavior using input and output variables [47]. The goal is to use this desired behavior to make predictions on data that the program, model, or computer previously has not seen during training [47]. The output variables in supervised learning are so-called target outputs [48].

The ML algorithms this thesis uses are described in the sections below.

2.5.1.1 Linear Support Vector Machine

A linear support vector machine (SVM) is an algorithm utilized in ML for classification and regression tasks. It aims to discover a hyperplane in a high-dimensional space that separates data points into various classes. This hyperplane is chosen to maximize the margin between the classes, which is the distance between the hyperplane and the closest data points from each class [49, 50, 51, 52].

The linear SVM algorithm functions by taking a group of labeled training examples and identifying the hyperplane that best segregates the samples into their respective

classes. To achieve this, the algorithm repeatedly modifies the parameters of the hyperplane until it discovers the one that maximizes the margin [49, 50].

In a binary classification issue, where only two classes exist, the hyperplane is a straight line that splits the feature space into two sections [51, 52]. For multi-class classification, the algorithm builds a series of hyperplanes, each dividing a pair of classes, and then selects the hyperplane that maximizes the minimum distance to the closest data point [52].

One of the critical advantages of linear SVM is its ability to deal with high-dimensional data and large feature spaces [53].

2.5.1.2 Random Forest

Random forest is an ensemble-based ML algorithm for classification and regression tasks. The random forest consists of several decision trees, all making individual predictions. These predictions are then used to make the final random forest prediction in the form of a voting scheme. The decision trees are trained using different subsets of the training data and features of the training data, leading to reduced variance and greater generalization [54, 55].

Random forests offer several advantages over other ML algorithms. They are simple to implement and can be applied to a wide range of datasets [54]. Moreover, they are resilient to noisy and outlier-laden data, making them a robust algorithm [56].

When working with random forests, several parameters can be tuned. Some of the most important ones are *n_estimators* and *max_depth*. The parameter *n_estimators* refers to how many decision trees should be included in the forest. And *max_depth* decides the maximum depth of the decision trees in the forest, i.e., how many nodes each decision tree can have [57].

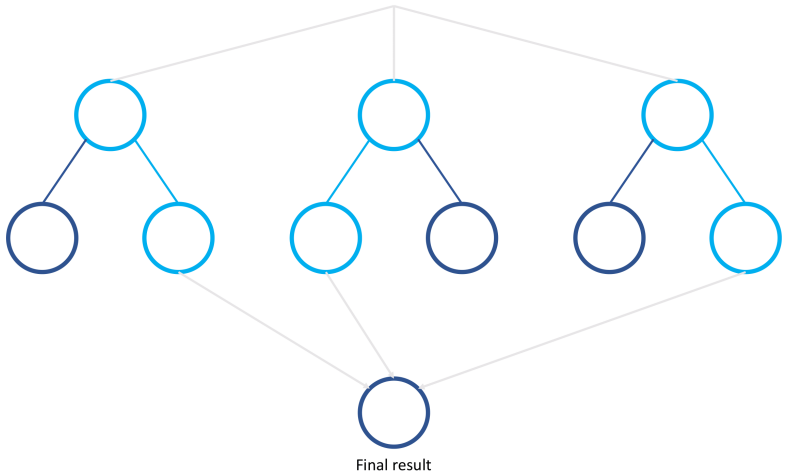

Figure 2.1: An overview of the random forest

2.5.1.3 Bayesian Inference

Bayesian inference involves analyzing data using Bayes' theorem. This approach involves updating prior knowledge about parameters in a statistical model with observed data. The prior knowledge is represented as a distribution and combined with the likelihood function of the observed data to derive the posterior distribution. The posterior distribution can be used to predict future events [58]. The most essential laws of Bayesian Inference, the Law of Total Probability and Bayes' theorem, are depicted in Equations 2.1 and 2.2.

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)} \tag{2.1}$$

where A, B = events, $P(A|B)$ = Probability of A given B, $P(B|A)$ = Probability of B given A, $P(A)$ and $P(B)$ = Independent probabilities of A and B (see [59]).

$$P(A) = \sum_{n} P(A|B_n)P(B_n) \tag{2.2}$$

where, A, B = events, $P(A|B_n)$ = Probability of A given B_n, $P(B_n)$ = Independent probability of and B_n (see: [59]).

2.5.2 Unsupervised Learning

Unsupervised learning involves no target outputs of any kind. Instead, a system that uses unsupervised learning will learn input patterns that can be found in the statistical structure of the input data [47].

2.6 Natural Language Processing (NLP)

NLP is the interdisciplinary subfield of linguistics and artificial intelligence [60]. It focuses on how computers can be used to understand or manipulate natural language in speech or text [61]. A goal of NLP is for a computer to achieve human-like language processing, including paraphrasing a given input text, translating, and answering questions regarding the context of the given input text. Other practical goals of NLP depend on the domain or field that it is implemented in [62].

2.6.1 Applications of NLP

All applications that involve text in any form have the potential to utilize NLP. Information extraction is one use-case of NLP and involves identifying, tagging, and extracting key elements of important information from an input text [62]. These fundamental elements can then be used for several purposes, such as summarizing text and identifying keywords and phrases for information retrieval. Additionally, information extraction is also helpful for text categorization [61].

Another application of NLP is machine translation. This application is perhaps the most commonly associated with NLP and involves translating a text from one natural language to another [61, 62]. There are several different approaches to machine translation, ranging from a more simple word-based approach to more complex ones involving a higher level of analysis [62].

2.6.2 Levels of NLP

There are several levels of language when it comes to NLP. Generally, the more levels an NLP system can utilize, the more sophisticated and capable the NLP system is [62]. In total, there are seven levels of language. However, the first level, phonology, deals with the speech sound of words and is thus not applicable to this thesis and will not be covered in depth.

The second level is the morphology level and focuses on the fact that words can be divided into small components morphenes [62]. These components convey meaning and can therefore be used to derive the meaning of the full word [63]. For instance, the word *preorder* could be morphologically separated into the prefix *pre* and the root *order*. These morphemes have a meaning of their own, which can be used to understand the meaning of the complete word they make up. Adding another morpheme to the word would change the meaning of the word, sometimes in a predictable manner. Taking a word that is a verb such as *preorder* and adding the suffix *-ed* would signal that something happened in the past already.

The next level is the lexical level which focuses on the meaning and interpretation of individual words [61, 62, 64]. The first step is to assign part-of-speech tags to each word in an input text [62]. This means that each word is categorized as a verb, pronoun, or noun. After that, words that only have one possible meaning can be switched out with their semantic representation. This can then be used to form interpretations of the meaning of a sentence [62]. Lexical processing does not capture the meaning conveyed in the ordering of words, leading to the issue of two sentences being treated the same as long as they have the same group of words, regardless of word order [64].

The syntactic level builds upon the lexical level, adding structural information in the sense of the positioning of words to determine the meaning of a sentence [62, 63]. This means that the NLP system would use syntactic structures in grammar to understand what is being said. This prevents the issue introduced in the previous level that two sentences with identical words but different ordering would be interpreted the same way[62, 64].

The semantic level finds the appropriate meaning of words through analysis of the rest of the sentence to determine the meaning of a whole sentence [63, 62]. Instead of focusing on individual words or phrases of a sentence that is done at previous levels, the semantic level focuses on the meaning of the whole sentence [64]. It also relates syntactic features for this purpose.

The discourse level of NLP uses the fact that different texts and documents, such as scientific reports or newspapers, are written in a particular structure to extract additional meaning [63]. This is typically called discourse structure recognition and is done by deconstructing a document into discourse components [62]. The NLP system would then use these different components to determine the role of that specific information [63]. Another type of discourse processing is anaphora resolution, which replaces semantically vacant words, such as pronouns, with the entity they refer to [62, 64].

At the highest level, there is the pragmatic level which deals with using real-world knowledge in the determination of the meaning of a text [61, 62, 64]. This means that additional knowledge can be conveyed without being encoded in the document's actual text through the contextual dimension [62, 64].

2.6.3 Text Embeddener

A word embedder is a technique used in natural language processing (NLP) to represent words numerically, typically as a high-dimensional vector of real numbers. The goal of word embedding is to capture the semantic meaning of words, and the relationships between them, which can then be used for various downstream NLP tasks, such as sentiment analysis, text classification, and machine translation [65].

There are different algorithms for generating word embeddings, such as Word2Vec and FastText. These algorithms typically use large amounts of text data to learn the embedding by modeling the co-occurrence statistics of words in a corpus [66].

Once the word embedding model is trained, it can be used to generate a numerical representation of any word in the vocabulary, which can then be used as input to ML models. This is often more efficient and effective than using the raw text of the words themselves, as the numerical embeddings capture meaningful semantic relationships between words that are difficult to capture otherwise [66, 65].

2.6.3.1 FastText

FastText is a popular open-source library for generating word embeddings in natural language processing (NLP). It was developed by Facebook's AI Research team and is based on the Word2Vec algorithm [67].

FastText is unique because it generates embeddings for individual words and character n-grams (sub-sequences of characters of length n) in words. As a result, it can effectively handle out-of-vocabulary (OOV) words and capture the semantics of words that might not be available in the training data [67].

The mechanism behind FastText involves training a neural network on extensive text data utilizing a technique known as skip-gram, wherein the model endeavors to predict context words (i.e., words in close proximity) based on a given target word. The embeddings learned from this process represent each word as a vector of real numbers, resulting in words with similar meanings being placed close together in

the embedding space tasks [67, 68].

FastText's key benefits include its high speed and efficiency, enabling it to process massive datasets using minimal computational resources. Additionally, it offers pre-trained models in various languages that can be adjusted to suit specific NLP [69].

3

Related Work

This section provides an overview of related work relevant to the research topics. The section begins by summarizing the existing literature and patents on optimizing CI configurations. After that, the section presents factors that are important when determining a domain's suitability for ML-methods.

3.1 CI Configuration Optimization

Academic research has explored the optimization of CI configurations as a way of improving CI performance. One approach was proposed by Santolucito et al. [15], who developed a system that acts as a static analysis tool capable of detecting CI configuration errors at the code level. The system uses a neural network to filter out constraints with a low likelihood of being the root cause of CI configuration errors. The approach taken by Santolucito et al. [15] for CI configuration optimization differs from our approach, as their focus is on identifying errors within the configurations. In contrast, our approach involves selecting the most appropriate CI configuration from a pre-defined set for a given test case.

Another approach to detecting errors in the CI configuration was proposed by Vassallo et al. [70] attempt to automatically identify smells in the configuration file. The study proposes a CD-Linter, a semantic liter that detects CI smells and evaluates its usefulness. The smell was detected by searching for specific parameters and evaluating if smell exists using a rule-based approach [70]. Contrary to our study, this approach detects CI configuration smell in existing CI pipelines, while our study aims to classify test specifications to pre-defined CI configurations.

In their study, Medvedev and Aksyonov [16] propose a simulation model that facilitates the optimization of CI/CD pipelines by providing a testing environment for exploring various factors that impact the performance of CI/CD pipelines. The simulation model is based on a queuing system that employs a multi-agent approach and allows for simulation using different test settings and scenarios. Medvedev and Aksyonov [16] specifically use the model to determine the optimal relationship between the number of service channels and the number of events per period. This approach has the advantage of being cost-effective and enables tracking of the factors that cause errors. Our approach has a distinct advantage over that of Medvedev

and Aksyonov [16] in that we focus on identifying the optimal CI configuration from a pre-defined set for each unique test case. This approach allows for a more targeted and customized optimization of CI performance compared to the simulation model used in their study.

Spieker et al. [19] present a reinforcement learning approach of selecting and prioritizing test cases in CI called RETECS to minimize the time between code commits and developer feedback when a test case fails. The reinforcement learning model Spieker et al. [19] propose makes choices based on the test cases' duration, previous last execution, and their failure history. Their approach can learn to prioritize error-prone test cases. While Spieker et al. [19] use a reinforcement learning approach, this study employs a supervised learning method, representing a key difference between the two works.

Bregman and Mattar [71] propose a method of using the characterization of CI/CD pipelines to select the set of execution platforms from the pool for assignment to the CI/CD pipeline. This method works by providing a definition of the CI/CD pipeline to a processing device that uses this definition to obtain a characterization of the CI/CD pipeline. After that, the processing device analyzes a pool of execution platforms to find a set of candidate execution platforms that are suitable for executing the CI/CD pipeline. Lastly, the processing device assigns the selected subset of execution platforms to the CI/CD pipeline for execution. To learn which combinations of attributes are optimal for executing the CI/CD pipeline, it is possible to use machine learning techniques. The main drawback of this method is that it requires a processing device as input which our method does not require.

The patent *Dynamic automation of pipeline workpiece selection* [72] discloses an AI platform that improves the efficiency and deployment time of CI/CD pipelines. The approach described uses application artifacts to generate one or several dependency graphs. Afterward, ML models are used to find the relationship between the components in these generated dependencies graphs and CI/CD artifacts. These identified relationships are then used to determine the impact of changes made to the CI/CD pipeline artifacts. The CI/CD pipeline can thus be optimized, knowing the effect of changes.

In their patent, Bergman and Gersht [73] disclose a method of executing a CI/CD pipeline using a container imagine file on another computer than the one associated with the CI/CD pipeline. This is achieved by first receiving a definition of a CI/CD pipeline through a processing device. Then the same processing device converts the received definition of the CI/CD pipeline into a container image file. This, in turn, allows for the processing device to implement a container executing the CI/CD pipeline on a second computer system. A drawback of their method is that it has scalability issues.

3.2 Applicability of ML-methods

Relating to **RQ3**, the characteristics that indicate whether a domain that is well suited for adaptation of ML are of interest. The applicability of machine learning is highly dependent on the underlying domain. Based on [74, 75], six characteristics have been identified as important factors in determining whether a domain is well-suited for machine learning.

1. **Tolerance to errors:** It is crucial to acknowledge that achieving 100 percent accuracy with machine learning models is virtually impossible. Therefore, a machine learning-friendly domain should be capable of tolerating a certain degree of error in the predictions or outcomes generated by the model. While minimizing errors is always desirable, having some flexibility in accepting inaccuracies and incorrect predictions is a characteristic of domains that are suitable for machine learning techniques [75].

2. **Inapplicability of conventional approaches:** Some domains may have complex problems that are not easily solvable using traditional models or algorithms. These domains require new approaches, and machine learning can often provide solutions where conventional methods fall short. Therefore, a machine learning-friendly domain is one in which conventional approaches lack effectiveness or efficiency, making machine learning techniques a more suitable alternative [75].

3. **Low interpretability requirement:** Machine learning models are often called black-box models because they operate by learning patterns and relationships from data without explicitly explaining their decisions. In certain domains, the interpretability of the model's internal workings may not be a critical requirement. As long as the model produces acceptable results, understanding how it arrived at them may not be necessary. Thus, a machine learning-friendly domain can tolerate solutions that are derived from black-box models [75].

4. **Mathematical expressibility of the objective function:** In machine learning, models are trained to optimize an objective function, quantifying the desired outcome or performance measure. In a machine learning-friendly domain, the objective function should be mathematically expressible. This enables mathematical optimization techniques to train the model and fine-tune its parameters. When the objective function can be precisely defined mathematically, designing and implementing machine learning algorithms becomes easier [75].

5. **Stability of the objective function:** A machine learning-friendly domain typically exhibits stability in its objective function over a sufficiently long period. This means the underlying problem or task for which the machine learning model is being developed does not undergo frequent or significant changes. Stability allows for the development of reliable and robust models,

as the objective function remains consistent, and the models can adapt and generalize well to new data within that domain [75].

6. **Availability of sufficient training data:** Machine learning models rely heavily on large amounts of labeled training data to learn patterns and make accurate predictions. A machine learning-friendly domain should have access to sufficiently large and representative training datasets. These datasets must encompass various scenarios, variations, and outcomes relevant to the domain. Adequate data availability ensures that the models can effectively learn from the data and generalize their knowledge to unseen instances [74, 75].

We hypothesize that the problem of assigning CI configurations is suitable for machine learning techniques. A simple mathematical expression for the objective function can be expressed as the difference between the predicted classification and the correct label. Furthermore, the objective function is stable, as long as the set of CI configurations does not change frequently.

Additionally, many companies log the test specifications and CI configurations in a central test management system, allowing for the gathering of sufficient training data from many organizations. This abundance of data makes it feasible to train an ML model effectively.

Currently, the best approach to assigning CI configurations is a manual process conducted by a senior test manager, resulting in significant costs. Given the high expense associated with the conventional method, we propose that implementing an ML model to perform this task would lead to substantial cost savings, rendering the traditional approach inefficient.

However, we acknowledge that we must fully know the domain's interpretability requirements and error tolerance. Nonetheless, we hypothesize that both interpretability and error tolerance can be addressed by employing appropriate ML methods.

4

Methodology

This chapter will describe this study's methodology and present the partner company, Ericsson. The methodology aims to find answers to the following research questions:

- **RQ1:** What performance can supervised ML achieve when classifying test specifications into the optimal pre-defined CI configurations?
- **RQ2:** What are the critical features for classifying test specifications into the optimal pre-defined CI configurations?
- **RQ3:** What is the applicability of ML methods for classifying test specifications to pre-defined CI configurations in practice?

Our methodology for answering the research questions is illustrated in Figure 4.1. By answering **RQ1**, more knowledge about the level of performance that can be achieved by supervised ML in assigning CI configurations to test specifications will be gained. This knowledge is crucial for assessing the applicability of ML methods to real-world testing processes and, thus, helps to answer **RQ3**.

To answer **RQ1**, we developed an ML model based on an ensemble architecture with three sub-models. The three sub-models classify based on the following factors:

- Semantic analysis of the test instruction in the natural language of the manual test specifications
- Analysis of the word distribution in test instruction text of the manual test specifications
- Categorical features concerning the manual test specifications

To answer **RQ2**, we looked into the ML model and the data used for training to gain insights into the factors contributing to the model's predictions. This sheds light on the model's criteria for predicting CI configuration.

To answer **RQ3**, we analyzed the performance of the models developed in **RQ1**. Moreover, a decision support system was developed based on the ML models de-

veloped in **RQ1**. This decision support system exemplifies how ML methods can be applied to this specific domain. Furthermore, we conducted a survey study, including this decision support system, answered by experts at the partner company Ericsson to see what applications are feasible given the performance of the models.

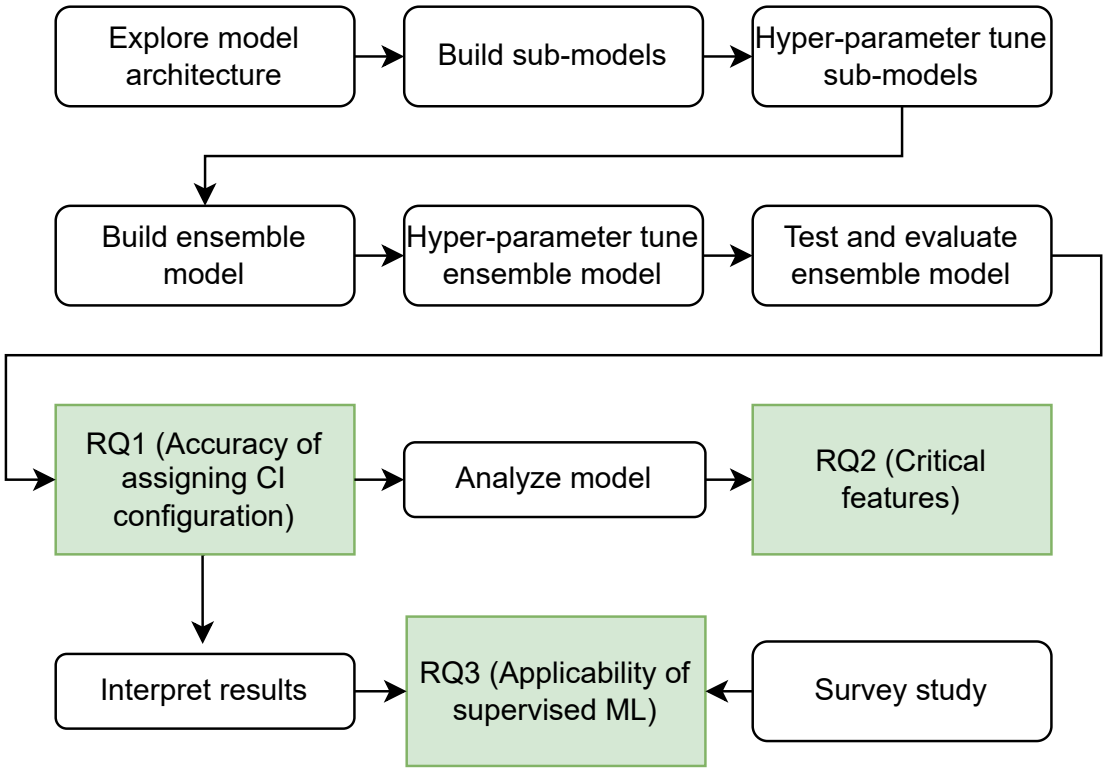

Figure 4.1: Overview of the case study process. Activities are white, and research questions are green

4.1 Research Design

This study employs a case study research design. To assess the performance of ML models in assigning CI configuration to test specifications, data from Ericsson CloudRAN will be used. The research also aims to identify critical features that aid in making informed decisions by analyzing these models. Additionally, a survey study on test managers at Ericsson will be conducted. While the study utilizes Ericsson CloudRAN data for model training and evaluation and Ericsson employees for the survey study, the findings are expected to have broader applicability beyond the telecom industry. The guidelines provided by Runesson and Höst [76] have been used to formulate the case study. An overview can be found in Table 4.1.

Objective:	Explore the performance of supervised ML for assigning CI configurations to test specifications, find the critical criteria in making that decision, and investigate the applicability of ML models for this purpose in practice
Case:	Assigning CI configurations to test specifications
Theory:	Classification models and CI
Research Questions:	RQ1, RQ2, RQ3
Methods:	Tool evaluation and Survey study
Selection Strategy:	Data provided by partner company and convenience sampling

Table 4.1: Overview of case study plan

4.2 Case Study Context

This study is conducted in collaboration with Ericsson, a Swedish telecom equipment supplier. The research focuses on the department at Ericsson responsible for the software product CloudRAN. CloudRAN is a radio access network architecture that leverages cloud computing to centralize and virtualize the baseband processing functions of a mobile network.

The selection of Ericsson's CloudRAN as a case study for this thesis is particularly advantageous due to its well-defined continuous integration workflow with distinct configurations. Moreover, Ericsson has accumulated relevant data over an extended period, resulting in a large annotated dataset. Furthermore, as a leader in software development, Ericsson's domain experts provide invaluable expertise and insights that significantly contribute to the academic research being conducted.

4.2.1 Case CI Configurations

At Ericsson, the CloudRAN team utilizes pre-defined CI workflows encompassing various configurations. Within the organization, the epic driver, in collaboration with developers, formulates test specifications in a semi-structured manner. These test specifications are subsequently manually assigned by a test manager to one of the pre-defined CI workflows, based on their specific characteristics. This assignment ensures that the test specifications are executed using appropriate CI configurations.

The CI configurations employed by Ericsson and utilized in this study are as follows:

Configuration 1:

- **Execution Environment**: Configuration 1 uses a decentralized environment where microservices execute in isolation.

- **Test Suite**: Due to its decentralized environment, Configuration 1 is suited for covering the lowest test levels, such as unit testing. Test cases focus on verifying functional requirements, and the test suite also ensures the validation of legacy functionalities.
- **Build Scheduling**: Builds are scheduled for execution after every commit.

Configuration 2:

- **Execution Environment**: Configuration 2 uses a centralized environment where microservices are tested simultaneously and interact.
- **Test Suite**: Configuration 2 covers tests regarding fulfillment of functional requirements related to basic and life cycle management functionality. Because this environment utilizes a centralized environment, unit, integration, and system tests can be included in a test suite.
- **Build Scheduling**: Configuration 2 can be executed in two different scopes, either on a single commit or overnight, where all commits since the previous build are bundled. The testing process for a single commit should be short (under one hour) to ensure a quick feedback loop for development teams and stakeholders.

Configuration 3:

- **Execution Environment**: Configuration 3 uses a centralized environment.
- **Test Suite**: Configuration 3 is used to execute comprehensive test suites, including test cases that verify non-functional requirements, allowing for testing more complex scenarios and functionalities. The testing aims to verify the application at an overall system level. This includes extended tests, trying to emulate customer-like environments. However, this increases the cost of using this configuration.
- **Build Scheduling**: Configuration 3 is executed weekly.

In current practice, a test manager manually assigns one of these three CI configurations to each test specification. The models developed in this study will be integrated into an internal decision support system at Ericsson, which will aid test managers in deciding which CI configuration should be assigned to each test specification.

4.3 Data

This section presents details related to the data used. It covers data contents, data quality, and data pre-processing.

Test instruction	**System under test**	**Test configuration**	...	**CI configuration**
Instruction A	System A	Test configuration A	...	CI configuration 1
Instruction B	System B	Test configuration B	...	CI configuration 2
Instruction C	System C	Test configuration C	...	CI configuration 3
Instruction D	System D	Test configuration D	...	CI configuration 1

Table 4.2: Examples of test specifications

4.3.1 Data Contents

In this study, the data was obtained from Ericsson in the form of test specifications. The data consisted of 32 columns, including categorical data such as the system-under-test, test configuration, targeted microservices, and natural language data such as test instructions. Table 4.2 provides examples of what a test specification looks like for illustration purposes. The data contained both active and deleted test specifications, with a total of 2143 data points. Of these, 1834 data points corresponded to active test specifications, which are still employed in Ericsson's software testing processes. The remaining 309 data points corresponded to deleted test specifications no longer used.

4.3.2 Data Quality

Several columns contained many invalid data points, which could impact the validity of the findings. However, some critical columns, including targeted microservice, quality area, and system-under-test, had a relatively higher proportion of valid data points. This suggests that these columns could be reliable predictors of the test specifications' CI configuration. In addition to the challenges mentioned earlier, many data points contained placeholder default values. This issue was addressed by filtering out data points with such default values to ensure they did not negatively impact the model's reliability and performance. The data completeness for the columns is depicted in Figure 4.2 The complete data pre-processing procedure is described thoroughly in section 4.3.3 of this study.

One challenge that was encountered with the dataset was the limited amount of valid data points in the target label. Out of all the data points, 487 were valid for the CI configuration label.

4.3.3 Data Pre-processing

Data pre-processing is a crucial step in ensuring the accuracy and reliability of the dataset used for training the model. This study encountered several data pre-processing challenges that needed to be addressed before further analysis. This

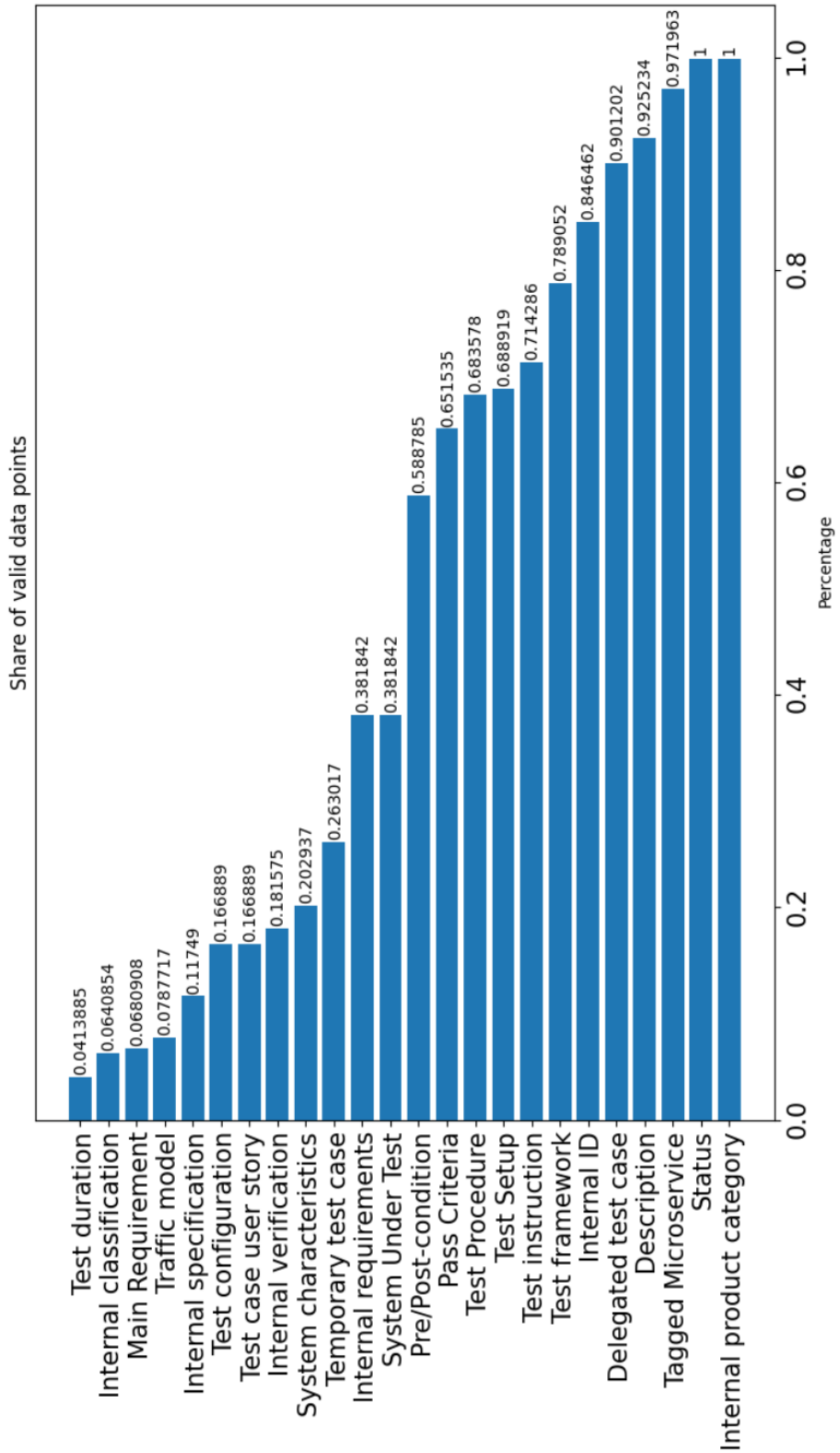

Figure 4.2: The percentage of test specifications with valid data points in the following columns after data processing

section outlines the steps that were taken to pre-process the dataset.

Firstly, it was noticed that some columns containing categorical data had different data but meant the same thing. This issue was addressed by replacing the data that meant the same thing with the same value to ensure consistency.

Secondly, data points that contained invalid test instructions were removed. Some of the test instructions were dummy or example tests that could not be used in the training or evaluation of the models. Additionally, all "N/A" was replaced with "-" to create a constant invalid data value across the dataset. Some data had their test instructions written in the wrong column, which had to be fixed manually. Furthermore, cells that were not filled in were filled in with a default value.

Thirdly, the test instructions were divided into four parts: test setup, pre/post-conditions, test procedure, and pass criteria. A script that identified keywords indicating each part's beginning and end was developed to achieve this. The script then divided the test instructions into four parts, and each part was put into its own column in the dataset.

Lastly, the categorical data was handled by applying a one-hot encoding method on the columns containing categorical data. This technique involves creating a new binary column for each unique category of categorical data. A value of 1 indicates that the data point had that feature, and 0 indicates that it did not.

In the data pre-processing pipeline, several steps to ensure the dataset's quality and to remove redundant test specifications were performed. Firstly, all test specifications with a default value in the test instruction column were removed, resulting in a dataset with 1482 test specifications. Then, duplicate test specifications were identified and dropped, reducing the dataset to 1249 test specifications.

A cosine similarity comparison of the vectorized test instructions using TF-IDF was performed to eliminate redundancy further. This method allowed identifying test specifications that had very similar test instructions. If the cosine similarity of the vectorized test instructions was one, test specifications were deemed very similar, and the copies were thus removed from the dataset. Following this process, we were left with 1237 test specifications in the final dataset. After that, invalid CI configuration tags were removed, resulting in a final dataset of 487 data points.

4.4 Machine Learning Models

This section presents this study's details related to ML models. The methodology covers model architecture, selection, hyper-parameter tuning, and training and testing.

4.4.1 Model Architecture

This section presents a comprehensive overview of the model architectures utilized in this study and the rationale behind the design decisions.

4.4.1.1 Ensemble Model for Assigning CI configurations

The chosen model architecture is an ensemble consisting of three sub-models for assigning CI configurations to test specifications. The reasoning for this decision is that the model needs to encapsulate various types of information, which can only be achieved using several sub-models. The three submodels in the ensemble are the categorically-, semantic- and word distribution-based models.

The three submodels cover different parts of a test specification. The first sub-model of the ensemble uses one-hot-encoded categorical data from the dataset to make predictions. The second sub-model is based on semantic analysis of the test instruction, while the third sub-model utilizes the word distribution of the test instruction and Bayesian inference. Combining the outputs of these three sub-models, the ensemble model aims to provide a more accurate and robust assignments of the CI configuration. This approach also allows the model to capture different aspects of the data, including categorical information and the semantic properties of the test instructions.

The ensemble model utilized in this study has a random forest as its outer layer. The random forest takes in the submodels' predictions and generates its prediction based on them.

The ensemble model's fit function takes in the training data and labels as parameters. These are then forwarded to the submodels, which are trained separately as described in their respective sections. The submodels create predictions using the training data, resulting in vectors with three elements, one for each CI configuration. These vectors are concatenated into a 27-element array, which is then used to train the random forest along with its correct labels.

When the ensemble model predicts the CI configuration for a test specification, it sends the specification to the three submodels. These submodels generate predictions, which are then concatenated into a 27-element vector. Finally, this vector is passed to the ensemble's random forest, producing the final prediction.

The ensemble model incorporates a rule-based component derived from an analysis of the dataset. Upon examination, it was discovered that all test specifications with a pre/post-condition section and pass criteria section exceeding 800 and 1000 characters, respectively, belonged to CI configuration 3. As a result, this rule-based component was integrated into the ensemble. To validate this rule, it was reviewed by domain experts at Ericsson, who confirmed its domain-specific relevance.

An overview of the ensemble model's architecture is depicted in Figure 4.3.

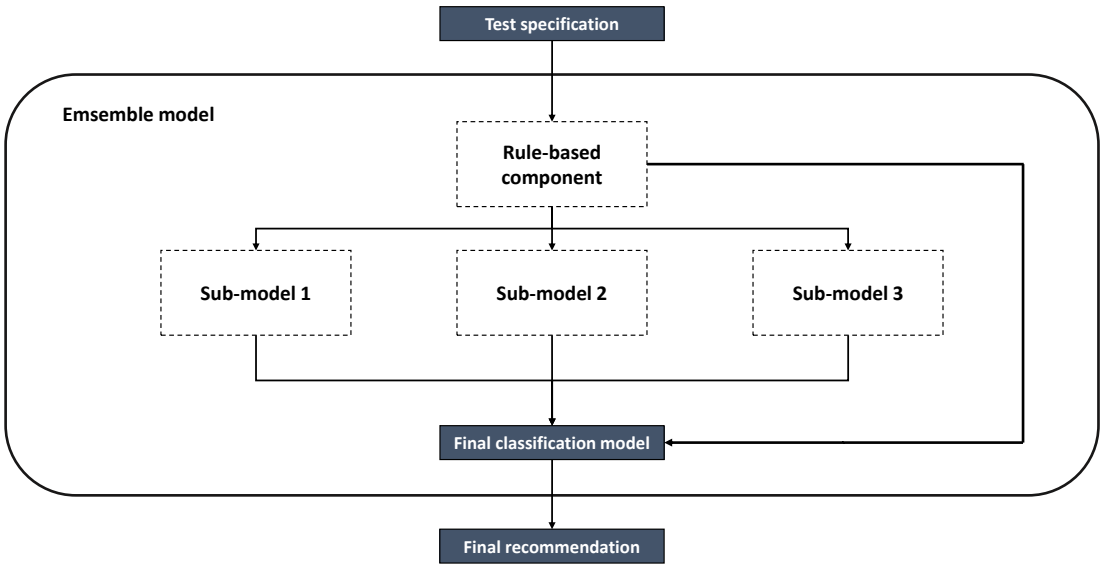

Figure 4.3: An overview of the architecture of the ensemble model

4.4.1.2 Categorical Model for Assigning CI Configurations

The categorical model is an ML model that makes predictions based on categorical features in a given test specification. To achieve this, the model first one-hot encodes the categorical data in the test specification. The resulting one-hot encoded test specification is an array with 271 elements, where each element corresponds to whether or not a categorical value is present in the given test specification.

In the categorical model, scikit-learn's built-in one-hot encoder called "OneHotEncoder" is used. It is initialized in the model's constructor function and fitted using the whole available dataset. The training data is inputted to the fit function, which transforms it into its corresponding one-hot encoded array using the previously initialized one-hot encoder. This array is then used to train a random forest classifier.

When it comes to predicting, the predict function takes in a test specification and transforms it using the one-hot encoder. The transformed test specification is then fed into the previously trained random forest classifier, which predicts the probability of the test specification belonging to each of the three CI configurations: Configuration 1, Configuration 2, and Configuration 3. Thus, making the output of the model a three-element array, which represents the predicted probabilities of the input test specification belonging to each of the three CI configurations.

4.4.1.3 Semantic Model for Assigning CI Configurations

The semantic model is designed to classify test specifications based on the semantics of the test instruction written in natural language. The test instruction is preprocessed and divided into four parts: test setup, pre/post-conditions, test procedure, and pass criteria. Each of these four parts is handled by a separate sub-model,

which makes a prediction based on the part of the test instruction it is responsible for.

The semantic model utilizes pre-trained FastText word embeddings, and the pre-trained weights are loaded during model initialization. During the fitting process, the model vectorizes the split test instruction of the inputted test specification using the pre-trained FastText weights. This is done by tokenizing the sentence and finding the vector representation of each word in the sentence, with each vector having 300 elements. The model then takes the average of all word vectors in a sentence to represent the sentence in vector format.

The vectorized sentences are used to train a random forest. During prediction, a test specification is inputted into the model, and its test instruction is extracted and divided into four parts. The four parts are vectorized using FastText, and the resulting vectors are inputted into the trained random forest to make the prediction. Since the semantic model has four sub-models, it outputs four three-element arrays, where each element corresponds to the probability of the test specifications belonging to each of the three CI configurations.

4.4.1.4 Word Distribution Model for Assigning CI Configurations

The ensemble model comprises multiple sub-models, the last of which is the word distribution model. This model uses Bayesian inference to make predictions using the test specification's test instruction, which is divided into four parts: test setup, pre/post-conditions, test procedure, and pass criteria. The word distribution model takes a test instruction as input and outputs four three-element vectors, each representing the probability of the test specifications belonging to the three CI configurations. Each three-element vector is the prediction for each part of the test instruction.

The fit function builds dictionaries to store statistics from the training data to train the model. These dictionaries contain information regarding the conditional probability that a specific word occurs, given a CI configuration. It also calculates and stores the probability of each word occurring in the training data and the probability of each CI configuration occurring in the training data.

During the prediction phase, the model calculates the conditional probability of a word bag given a particular CI configuration. It does this by taking the product of all the conditional probabilities for the words in the word bag given that particular CI configuration, which were calculated and stored in the fit function. Although this method assumes that the words are independent of each other, which is not always true in reality, the impact of this assumption on the performance of the model is assumed to be negligible. The conditional probability of the word bag given a CI configuration is calculated for all three CI configurations. The conditional probability of a CI configuration given a word bag is then given by the quotient of the conditional probability of the word bag given a CI configuration divided by the sum of the conditional probability of the word bag given a CI configuration,

where the sum loops through all possible CI configurations. These three conditional probabilities are the probabilities that the model outputs.

If a word that has not been seen during the training phase occurs during prediction, the model will assign the probability for that particular word to a very small number.

4.4.2 Model Selection

This section outlines the approach to selecting the ensemble model and its sub-models. The rationale and decision-making process for choosing each model are explained in detail.

4.4.2.1 Categorical Model for Assigning CI Configurations

Several models were considered for the categorical model for predicting CI configurations. These models included random forest, support vector machine, and logistic regression. The decision to consider these models was based on their simplicity, making them suitable for the small dataset used in this study. Additionally, speed and simplicity were prioritized over complexity, given that the models were intended to be implemented in a decision support system.

To determine the most suitable model, the performance of these models was compared against each other using accuracy and F1-score. After the evaluation, it was found that random forest had the best performance, and it was selected as the model of choice for the categorical model for the study.

4.4.2.2 Semantic Model for Assigning CI configurations

The word embedding techniques FastText, BERT, and Doc2Vec, were tried for the text classification task. Pre-trained FastText and BERT embeddings were used, while Doc2Vec was trained on the entire dataset. Doc2Vec performed poorly compared to the other two approaches, and FastText was ultimately selected due to its faster speed and comparable performance to BERT. Another benefit of FastText is that it can handle words that it has not been trained on, which is crucial when dealing with a niched dataset such as the one in this study. To embed the test specifications into vectors, a pre-trained FastText model was utilized as previously discussed. The vectors obtained were then utilized for training three different models: random forest, decision tree, and logistic regression. Evaluation of the models was based on accuracy and F1 score. Random forest was determined to be the most effective model based on accuracy and F1 score.

4.4.2.3 Word Distribution Model for Assigning CI configurations

The word distribution model leverages Bayesian inference to predict the CI configuration for each test specification. To encapsulate more information from the test instruction in natural language, this sub-model is included in addition to the semantic model and rule-based feature of the ensemble since this model will provide a lexical aspect that is not is not in focus in the other sub-models.

4.4.2.4 Ensemble Model for Assigning CI configurations

Due to limited data availability, simpler ML models were evaluated for the ensemble model. The three models considered were random forest, decision tree, and logistic regression. The evaluation was based on accuracy and F1 score, and the random forest model was determined to be the most effective.

4.4.3 Bootstrapping

Due to the relatively small size of the dataset, the study utilized bootstrapping to enhance the models' performance. Random sampling with replacement was applied to the original dataset, and the resulting samples were added. The number of bootstrap samples was a variable parameter, and various values were explored to identify the optimal balance between benefits and drawbacks.

When selecting the number of bootstrap samples, balancing over- and under-sampling is crucial. Over-reliance on randomness might lead to increased bias if too few bootstrap samples are used while using too many bootstrap samples can increase computational costs. The number of bootstrap samples was determined by systematically evaluating model performance across various possible values. It should be noted that bootstrapping was employed in conjunction with the tuning of other hyper-parameters, and a detailed description of this process is provided in the following section of this study.

4.4.4 Hyper-parameter Tuning

Three hyper-parameters have been tuned in this study: n_estimators, max_depth, and the number of bootstrap samples. The ranges considered in the tuning process for each hyper-parameter were: max_depth (5 to 45), n_estimators (5 to 500), and the number of bootstrap samples (0 to 50). These ranges are chosen to balance computational constraints and the need for a comprehensive search.

An exhaustive search is conducted over all possible combinations of the three hyper-parameters mentioned above to find the best set of hyper-parameters. The F1 score is used as the evaluation metric for each hyper-parameter configuration. The reason for testing combinations of hyper-parameters instead of finding an optimal value one by one is that the hyper-parameters depend on each other. The hyper-parameter combination was chosen using a greedy approach, where the combination that resulted in the highest F1 score was selected.

In this study, hyper-parameter tuning was performed on three models: the categorical, semantic, and ensemble models. The tuning process was conducted sequentially, where the categorical and semantic models were tuned first. Their respective optimized hyper-parameters were then used to train the corresponding sub-models of the ensemble model. Finally, the ensemble model was also tuned.

The results of the hyper-parameter tuning process for each model are presented in Figures B.1 - B.6. The categorical and semantic models were tuned individually, with

the aim of optimizing their respective performances. The ensemble model, which combines the outputs of the categorical and semantic models, was then fine-tuned based on the optimized hyper-parameters of the individual models.

Table 4.3 presents the hyper-parameter settings selected for each model.

Model	n_estimators	max_depth	Bootstrap
Categorical	250	45	0
Semantic (Test setup)	500	35	2
Semantic (Pre/post-conditions)	250	20	40
Semantic (Test procedure)	500	45	5
Semantic (Pass criteria)	250	15	20
Ensemble	500	10	10

Table 4.3: Hyper-parameter settings selected for each model

4.4.5 Training and Testing

In this study, the pre-processed dataset was used for training and testing. To split the dataset, we employed scikit-learn's built-in train-test split function with an 80/20 ratio, where 80% of the data was used for training and 20% for testing. The training data was then passed to the ensemble model for training, which was further forwarded to the sub-models and bootstrapped with the number of samples determined during hyper-parameter tuning.

After training, the model was evaluated using the reserved testing data. For each run, the F1 score and accuracy were recorded. This process was repeated 100 times to ensure the robustness of the results. The final performance of the model was determined based on the mean values of the F1 score and accuracy over the 100 runs. We included both accuracy and F1 score as evaluation metrics because the dataset was not 100% balanced, and using only accuracy might have produced skewed results.

4.5 Decision Support System

In this study, we created a web portal with an integrated graphical user interface that serves as a decision support system. This system helps test managers create new test specifications and receive real-time recommendations for the most suitable CI configuration for the given test specification.

In the future, it is planned to integrate this system into Ericsson CloudRAN's test management system to streamline the testing process, reduce the number of misconfigurations, and improve overall efficiency. The goal is to provide a reliable and efficient tool that test managers can use to make informed decisions and optimize the testing process.

The web portal is an illustrative example of how supervised ML can enhance the assignment of CI configurations. This example will provide valuable guidance and assist domain experts in evaluating the applicability of supervised ML in this specific domain.

Figure 4.4 shows the interface of the decision support system.

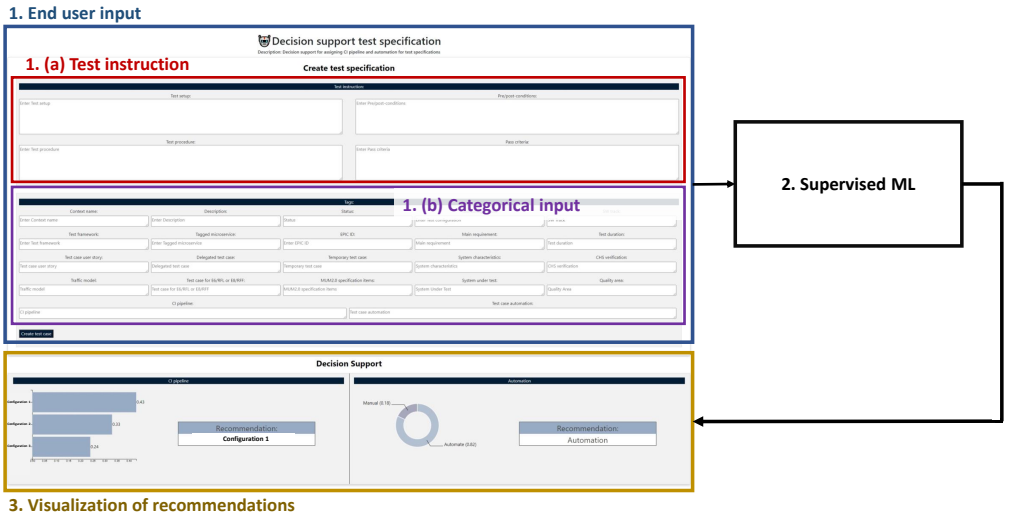

Figure 4.4: An overview of the interface of the decision support system

Test instruction (end user input): In the first section of the GUI shown in Figure 4.4, users can input data related to the test instruction for the feature to be tested. Although the user can enter this information as free-form text, the GUI guides them to provide the information in a structured manner. The GUI prompts users to enter the subparts of the test instruction in separate text boxes: test setup, pre/post-conditions, test procedure, and pass criteria.

Categorical (end user input): In the second section of the GUI shown in Figure 4.4, end users are required to input categorical data. To ensure the user-friendliness of the GUI, some categories are selected by the end-user from a predetermined dropdown bar. However, some categories have a large set of possible options or are dynamic, where the categories are subject to change over time. For such categories, the user is required to input data in free text without any limitations.

Visualization of recommendations: In the third section of the GUI shown in Figure 4.4, recommendations generated by the AI model are displayed, which are determined based on the test specification inputted by the user. The recommendation of CI configuration is visualized through a bar chart, where the confidence of the AI model for each CI configuration is indicated. The CI configuration with the highest confidence is selected as the recommended CI configuration.

A domain expert makes the ultimate decision on the CI configuration. The decision provided by the domain expert is entered into the web portal and then propagated to the test management system used by Ericsson CloudRAN.

4.6 Survey Study

A survey study was conducted to answer research question **RQ3**, regarding the applicability of supervised ML for assigning CI configurations. The survey primarily concentrated on the decision support system presented in Section 4.5. Nevertheless, many of the questions can be generalized to other use cases of ML models in assigning CI configurations. The survey took the form of an online form, including both closed and open-ended questions. The choice of the online survey was motivated by its low cost and time efficiency, compared to many other types of surveys, such as face-to-face interviews [77]. The population targeted for the survey are people with domain knowledge of software testing in the context of Ericsson. Convenience sampling was used as the sampling technique for this survey study. Test managers at Ericsson CloudRAN in Sweden and Canada were approached through our contact person at Ericsson and invited to participate in the survey study. A total of 10 test managers took part in the survey study.

The survey comprised 27 questions divided into six categories: correctness, speed, value, usability, interpretability, and stability. These categories are inspired by the characteristics presented in 3.2. In Appendix A, in Tables A.1 - A.6, the questions are presented. The correctness section of the questionnaire aims to explore the performance requirements for an ML model to be applicable in the given domain. Conversely, the speed-related questions assess the necessary speed of the ML model to serve as a useful decision support tool. The value section focuses on investigating the investment case for implementing a decision support system similar to the one proposed in this study. The usability questions delve into the end users' perspective and examine such a system's usability. Interpretability questions consider both the need for high interpretability of ML systems in the given domain and evaluate the level of interpretability exhibited by the proposed ML models in this study. Lastly, the stability section investigates the stability of the underlying domain and how it may impact the applicability of ML models for the given task.

The survey responses then underwent thematic analysis, a qualitative research method that identifies patterns and themes within the data. This involves familiarizing with the responses, coding relevant concepts, generating initial themes, reviewing and refining them, and finally defining and naming them. The thematic analysis enables a systematic and comprehensive exploration of the survey data to uncover meaningful insights and understand underlying meanings. In addition, we use descriptive statistics to analyze the quantitative data.

5

Results

This chapter will present an assessment of the performance of the ML models, an in-depth analysis of the ML models, and the results from the survey study conducted with domain experts at Ericsson regarding the applicability of AI methods in assigning CI configurations. Note that, in this chapter, we focus on providing the results. In Chapter 6, we will discuss the results and provide answers for the research questions.

5.1 RQ1—Performance of Models for Assigning CI Configurations

This section presents the level of performance that can be obtained using ML models to assign CI configurations for test specifications. This data is used to answer **RQ1**. The performance of all sub-models and the ensemble model will be discussed.

5.1.1 Performance of Categorical Model

The performance of the categorical model for assigning CI configurations is summarized in Table 5.1, which includes key metrics such as F1 score, accuracy, recall, and precision. These metrics have been calculated as averages over 100 runs.

Metric	Value
F1 score	0.8949
Accuracy	0.9030
Recall	0.8957
Precision	0.8976

Table 5.1: Performance metrics of the categorical model

5.1.2 Performance of Semantic Model

Table 5.2 presents the performance of the semantic model's sub-models in assigning CI configurations, with key metrics such as F1 score, accuracy, recall, and precision. These metrics are calculated as averages over 100 runs and are presented for each subpart of the test instruction.

Metric	Test setup	Pre/Post-conditions	Test procedure	Pass criteria
F1 score	0.8199	0.7801	0.7642	0.7877
Accuracy	0.8335	0.7936	0.7816	0.8017
Recall	0.8176	0.7817	0.7610	0.7849
Precision	0.8307	0.8163	0.7794	0.8000

Table 5.2: Performance metrics for the semantic model

5.1.3 Performance of Word Distribution Model

The performance of the word distribution model for assigning CI configurations can be found in Table 5.3. The table includes key metrics such as F1 score, accuracy, recall, and precision, which have been calculated as averages over 100. The performance metrics are calculated for each subpart of the test instruction.

Metric	Test setup	Pre/Post-conditions	Test procedure	Pass criteria
F1 score	0.7853	0.7680	0.7753	0.6965
Accuracy	0.7944	0.7800	0.7917	0.7218
Recall	0.7900	0.7633	0.7600	0.6924
Precision	0.8040	0.8005	0.8432	0.7381

Table 5.3: Performance metrics of the word distribution model.

5.1.4 Performance of Ensemble Models

The outcomes were obtained from 100 model runs, and the average F1 score, accuracy, precision, and recall were calculated for analysis. Table 5.4 showcases the performance achieved by executing the model with the optimal combination, as identified during hyper-parameter tuning (see Section 4.4.4). The ensemble model, on average, generates recommendations in 0.0037 seconds (measured over 370 runs).

Metric	Value
F1 score	0.9075
Accuracy	0.9139
Recall	0.9104
Precision	0.9086

Table 5.4: Performance metrics of the ensemble model

5.2 RQ2—Critical Features for Assigning CI Configurations

The following section analyzes the ML models in order to answer **RQ2**.

5.2.1 Analysis of Categorical Model

As the categorical model for assigning CI configurations is based on a random forest, it is possible to calculate feature importance to determine which tags have the highest impact on assigning a test specification's CI configuration. However, since the model uses one-hot encoded data, it is necessary to aggregate the feature importance of each specific tag value. This requires adding the scores of all tag values that belong to the same tag. The resulting feature importance scores are visualized in Figure 5.1.

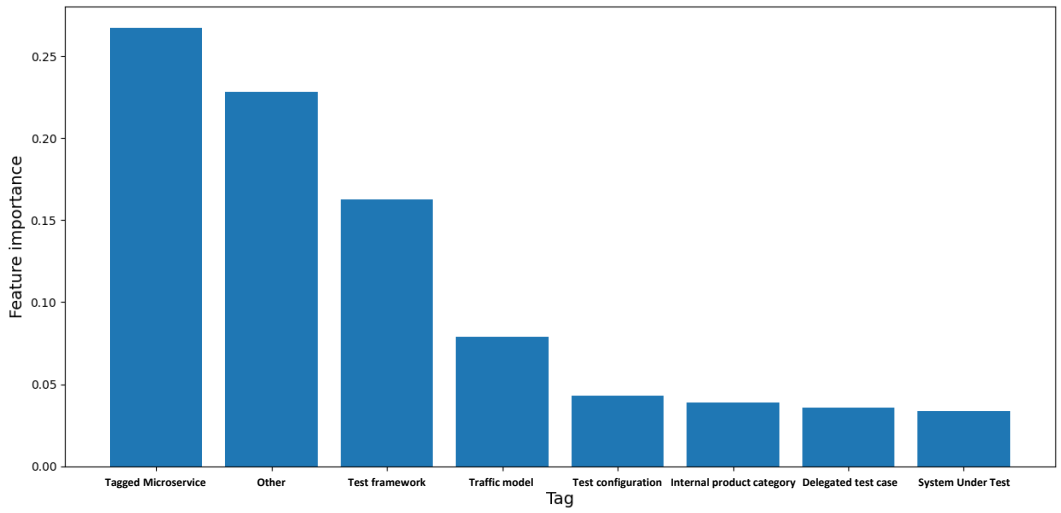

Figure 5.1: Feature importance in the categorical model

5.2.2 Analysis of Word Embeddings used in Semantic Model

The word embeddings used in the semantic models consist of 300-dimensional vectors. To visualize and therefore gain insights from the word embeddings, it is possible to apply dimensionality reduction techniques to reduce the dimensions of these vectors to two dimensions. This makes it possible to plot the test specifications as data points in a scatter plot to see potential clusters and gain insights. This has been done using t-SNE and is visualized in Figure 5.2.

5.2.3 Analysis of Word Distribution Model

The word distribution model for assigning CI configurations is analyzed by calculating the significance score of a specific word in a particular CI configuration. This involves calculating the conditional probability of a test specification belonging to a CI configuration given the presence of the word and then calculating the conditional probability of a test specification belonging to another CI configuration given the word. The quotient of these probabilities yields the significance score. This analysis is performed separately for each of the four parts into which a test instruction is

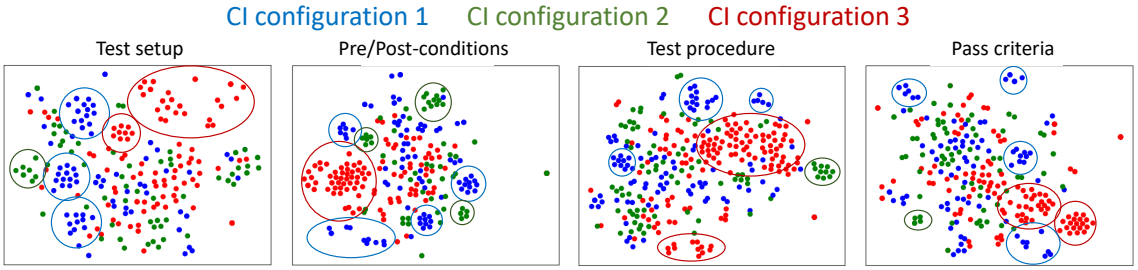

Figure 5.2: Scatter plot of dimensionality reduced test instructions

divided. Table 5.5 shows the five words with the highest significant scores for the three CI configurations, considering different parts of the test instruction.

Test setup		
CI configuration 1	**CI configuration 2**	**CI configuration 3**
ready	"Internal Ericsson information"	setuplb
"Internal Ericsson information"	"Internal Ericsson information"	nsa,nr
cu-up	setupthe	set
default	top	same
setup	ctx	setuplitmus
Pre/Post-conditions		
CI configuration 1	**CI configuration 2**	**CI configuration 3**
download	coordinator	logon
deploy	msrbs	shown
white	c2cci	pre-post
later	post-condition	rbs
ppt	online	fhgw/lte
Test procedure		
CI configuration 1	**CI configuration 2**	**CI configuration 3**
indicates	moi	impacted
permission	operationalstate	rbs
equal	availabilitystate	associated
it	read-only	released
fail	present	service
Pass criteria		
CI configuration 1	**CI configuration 2**	**CI configuration 3**
list	string	logon
commands	validation	shown
kvdb-rs	logs	rbs
passes	present	vcu
user	if	pre-post

Table 5.5: Significant words in the sub parts of test instruction

5.2.4 Analysis of Ensemble Model

Feature importance analysis can be used to assess how the ensemble model makes its decisions. This analysis allows us to identify the sub-models within the ensemble that hold the highest importance in making assignments. Figure 5.3 presents the feature importance scores for each sub-model.

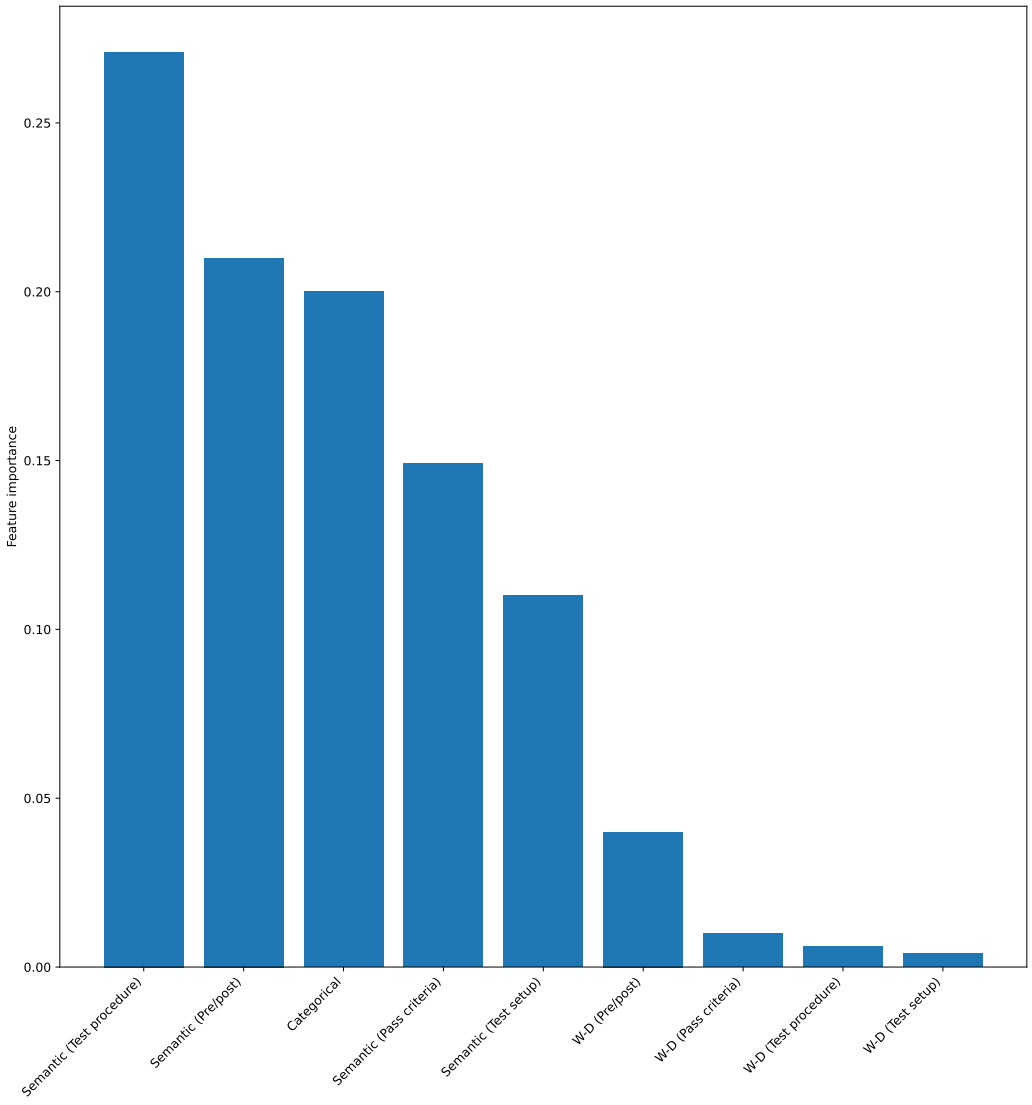

Figure 5.3: Feature importance of ensemble model for assigning CI configurations

5.3 RQ3—Applicability of Supervised ML

In this section, we present the findings of the survey study, conducted among Ericsson employees to investigate the potential application of ML in software testing.

We have categorized the survey responses and synthesized the results accordingly.

5.3.1 Correctness

The survey respondents agree that for a machine learning model to be used as a fully automated system, it must have a higher performance level than if it is used as a decision support tool. However, there is no consensus on the specific performance level that must be achieved for either use case. Figures 5.4 and 5.5 present the distribution of survey responses. The data indicates that the accuracy performance necessary for reliable decision support is distributed across multiple divisions between 70–99 percent, highlighting significant discrepancies among domain experts' judgments. The median accuracy needed to not negatively affecting the use of the model as a decision support stands at 85 percent.

Regarding the automated assignment of CI configurations, the performance requirements exhibit a higher frequency of responses concentrated in the upper range of accuracy. The median accuracy performance requirement obtained from the survey for automated CI configuration assignment was 90 percent. A greater level of accuracy is needed to trust a tool to make decisions without human involvement.

Furthermore, the respondents assessed the consequences of misassignment as significant, with a average score of 3.9 on a scale from 1 to 5, where 5 represents "very significant." Interestingly, the survey suggests a disagreement among domain experts regarding whether the consequences of misalignment of CI configurations are uniform across all configurations. Sixty percent of the respondents claimed that the consequences were the same, while forty percent stated that they varied. Those who believed that the consequences differed argued that misassignment of CI configurations occurring late in the development process and involving high test levels incurred higher costs.

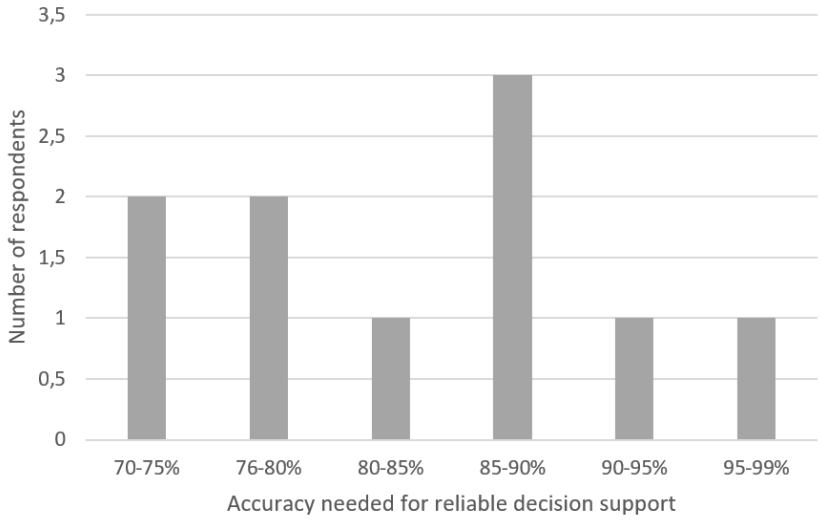

Figure 5.4: Survey result regarding the needed accuracy for decision support.

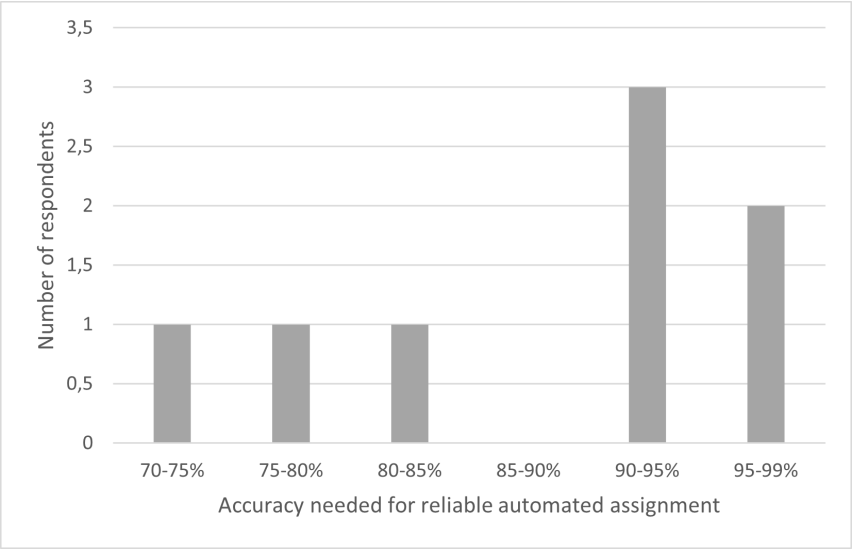

Figure 5.5: Survey result regarding the needed accuracy for automated CI configuration assignment.

5.3.2 Speed

The majority of respondents expressed the opinion that the ML model should generate predictions within a few minutes or in a few seconds to serve as a decision support tool without negatively impacting user willingness to utilize the tool. Four respondents believe the recommendation should be generated in a few seconds, while four other respondents think that the recommendation can be generated in a few minutes without negatively impacting usability.

However, some variations in opinions do exist. For instance, one respondent expressed satisfaction with receiving recommendations within a few hours, while another highlighted the importance of an instant generation of recommendations to enhance the users' sense of awareness and trust in the ML models.

Ninety percent of the respondents believed that sacrificing some speed in the current ML model to improve the accuracy performance would be beneficial. Some respondents express a strong preference for a more accurate model at the expense of the speed at which the recommendations can be generated. Other respondents point out that it would be worth trading off some speed to increase accuracy performance until the generation time of the model is approximately one second.

5.3.3 Value

The survey respondents acknowledge the difficulty of accurately estimating the time saved through the use of the proposed tool for assigning CI configurations. However, there is a consensus among respondents that significant time savings can indeed be achieved. For instance, one respondent mentioned potential savings of a couple of hours per test specification, while another suggested a saving of one hour per test specification. Moreover, yet another respondent emphasized that the savings would

be “huge.”

Furthermore, the survey results highlight that the implementation of the proposed ML models could effectively reduce coordination efforts within Ericsson. As a result, decisions could be made days faster compared to scenarios where ML models are not utilized.

The survey results indicate a disagreement among respondents regarding the extent to which misassignment of CI configurations will be reduced when the proposed tool is implemented. Figure 5.6 displays the distribution of the responses. Upon analyzing the answers, it becomes apparent that the majority of respondents are optimistic about the potential of the proposed tool to significantly reduce misassignments. However, two respondents appear to be skeptical, believing that the reduction will be below 10 percent. The median estimate for the reduced number of misassignments stands at 60%.

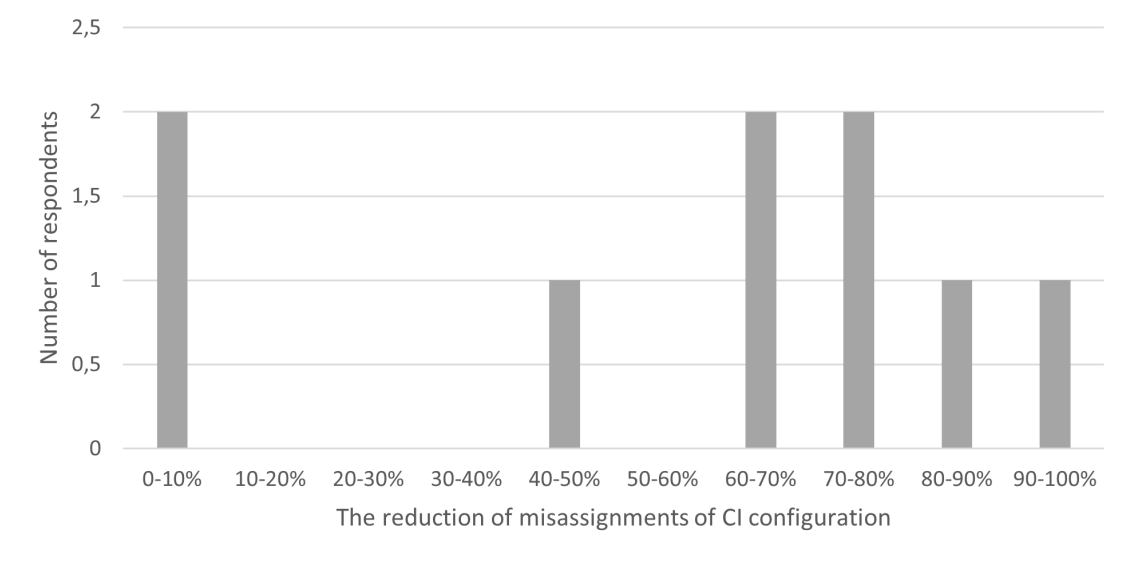

Figure 5.6: Survey result regarding the estimated reduction of misassignment of CI configurations

5.3.4 Usability

Based on the survey responses, assigning CI configurations to test specifications can be moderately to highly challenging, with a mean score of 3.6 out of 5, where 5 represents a high level of difficulty. However, the respondents suggested that having access to a decision support system could significantly alleviate this difficulty, with a mean score of 1.9 for the level of difficulty in assigning CI configurations to test specifications with the help of a decision support system. The majority of respondents indicated that they would actively use the system if it were implemented and were confident in its compatibility with their current workflow.

Those who answered that they would not use the decision support system either did not find the task of assigning CI configurations to be difficult or felt that the system

did not integrate well into their workflow. Overall, the respondents demonstrated a strong trust in AI-based systems and their practical applications in the workplace.

5.3.5 Interpretability

The survey reveals that the software testing domain highly values interpretability. Regarding the importance of interpretability, the respondents answered with a mean answer of 4.5 on a scale of 1 to 5, where 5 indicates that interpretability is very important. This means that decision support system users in this domain are highly interested in understanding how ML models arrive at their recommendations. However, the respondents' opinions on the level of interpretability of the proposed decision support system were scattered. Most responses fell within the range of 2 to 4 on a scale of 1 to 5, where 5 represents "strongly agree" and mean of 3.5 that the system adequately addresses interpretability concerns.

5.3.6 Stability

According to the survey results, the respondents generally agree that the distribution of CI configurations will change in the future. The mean answer was 3.8 on a scale of 1 to 5, where 5 indicates that it is very likely that the CI configuration will change in the future. However, the respondents were less certain about the stability of test instruction. The mean of their responses was 3.6 on a scale of 1 to 5, where 5 indicates that it is very likely that the structure of test instruction will change in the future.

Regarding potential modifications to the CI configurations, the respondents had varying opinions. 50% of the respondents rated the likelihood of changes to the CI configurations as 3 out of 5 on the scale, where 5 represents a high probability of changes occurring. The remaining respondents overwhelmingly perceived a high likelihood of changes to the CI configurations.

6

Discussion

6.1 RQ1—Performance of Models for Assigning CI Configurations

The machine learning models developed in this study achieved a final accuracy of 0.9139 and an F1 score of 0.9075 for assigning CI configurations, which is generally considered to be a good performance. Furthermore, the categorical-based sub-model outperformed the other sub-models in terms of F1 score and accuracy, followed by the semantic model and the word distribution model.

6.1.1 Discrepancy Between Sub-models

The discrepancy in performance between the sub-models in the ensemble is notable. The categorical model performs noticeably better than the semantic and word distribution models. Additionally, the semantic model generally outperformed the word distribution model, although certain sub-models within each model did not follow this pattern. Specifically, the semantic model was better at predicting using test setup and pass criteria, while both models performed similarly for pre/post-conditions and test procedure.

One possible explanation for the higher performance of the categorical model compared to the other sub-models is that there simply is more relevant information in the tags of the test specification in comparison to the natural language in the test instruction. Alternatively, the sub-models using the test instructions may not be appropriately designed to capture the underlying meaning of the instructions. Different NLP techniques are suitable for various NLP problems. In this case, the semantic representation of test instructions in the form of word embeddings may be more appropriate than analyzing the statistical distribution of words. Choosing an alternate way of interpreting the test instruction could result in higher performance. Another possibility is that the categorical data used by the categorical model has higher quality than the test instructions used by the semantic and word distribution models. However, a deeper analysis of the dataset is needed to determine whether this is the case, which is beyond the scope of this thesis.

6.1.2 Discrepancy Between Parts of Test Instruction

In addition to the discrepancy between sub-models, there is also a noticeable difference between the models that use different subparts of the test instruction. The test setup subpart appears to contain the most relevant information, as both the semantic and word distribution models perform better when using the test setup compared to other subparts of the test instruction. This suggests that the test setup is more effective in assigning CI configuration than the other subparts when using the specific NLP techniques used in this thesis. However, it is important to ensure that the different subparts of the test instructions are written with equal effort to draw this conclusion. Suppose the test manager writing the test instruction spends more effort describing the test setup than the pass criteria, for instance. In that case, it is unsurprising that the test setup contains more useful information and vice versa.

The semantic model outperforms the word distribution model in assigning CI configuration using the subpart pass criteria. However, both models perform similarly well in utilizing the subparts pre/post-conditions and test procedure. Since the models are evaluated using the same test specifications and, consequently, the same pass criteria, it can be concluded that the semantic representation of pass criteria contains more useful information than the statistical distribution of words.

6.1.3 Size and Quality of the Dataset

Various factors can affect the validity of the model's performance. This includes the size of the dataset. The dataset being too small could potentially result in overfitting of the model and, in turn, lower performance. Nonetheless, this issue does not significantly impact the model, as the F1 score and accuracy are stable. A too-small dataset could also potentially result in underfitting as the dataset is too small to include the complexity of the real-world problem. This is not a problem either due to the high performance of the models.

The performance of the different sub-models may vary due to the size of the data, as it reflects the complexity of the real-world domain that is captured by the dataset. Various machine learning techniques have varying requirements regarding the size of the dataset and complexity, which may explain why the different sub-models perform differently.

The quality of the dataset also impacts the performance. It is reasonable to think that having better coverage of the categorical data in the test specification could improve the overall performance of the ensemble model, especially considering that the sub-model that performed the best was the categorical model. Assuming that the categorical model performed the best due to underlying structural reasons, and not simply due to the categorical data being of better quality than the natural language data, suggests that improvements in size and quality of the categorical data could result in better performance of the ensemble model.

An unbalanced dataset usually impacts machine learning models negatively and

must be addressed. In this study, although the dataset is not perfectly balanced, it only suffers from a mild case of class imbalance with a class imbalance ratio of 1.71. To determine if the model is affected negatively by this imbalance, it is essential to evaluate the accuracy and recall metrics, as shown in Table 5.4. A common indicator that an ML model is affected negatively by class imbalance is that it has a low recall but high precision. However, both performance metrics are similar in this case, and the recall is even slightly higher than precision. This is likely due to the fact that the class imbalance is very mild in this instance.

RQ1: Supervised ML can achieve an accuracy of 0.9139 and an F1 score of 0.9075 for assigning CI configurations to test specifications.

6.2 RQ2—Critical Features for Assigning CI Configurations

6.2.1 Rule-based Component

The rule-based component evaluates the length of the subparts pre/post-conditions and pass criteria, as discussed in Section 4.4.1.1. The length of the subparts of the instruction text is a feature that seems to be a good predictor of CI Configuration 3.

Figure 6.1 shows the metric of the probability that a CI configuration has the given length normalized, so that the total of the probabilities of all CI configurations adds to one for every given length. This metric is independent of the prior, resulting in a clearer representation of how the length of the subparts differs for the CI configurations.

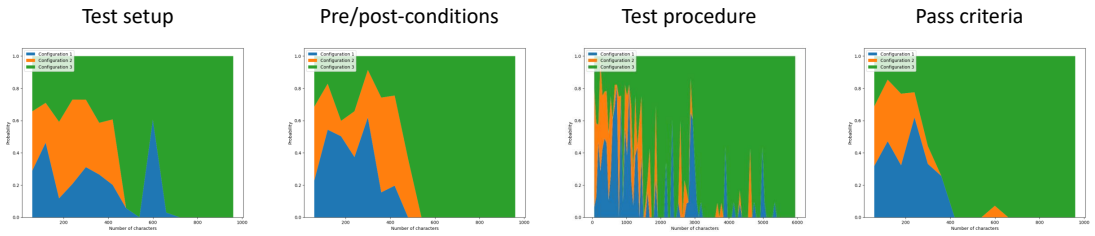

Figure 6.1: Visualisation of how the subpart of the test instruction differs for the CI configurations (Configuration 1 = Blue, Configuration 2 = Orange, Configuration 3 = Green).

The analysis reveals that Configuration 3 (in green) becomes increasingly prominent as the length of the subparts' texts increases. This trend is particularly noticeable in the subparts test setup, pre/post-conditions, and pass criteria. However, it is essential to note that the subpart test setup has limited data points for long text,

which may skew the interpolated results, giving the impression of a stronger correlation between long test setups and Configuration 3 than can actually be motivated by the data.

6.2.2 Submodel—Semantic

Examining Table 5.2, it becomes evident that the machine learning model excels in learning the test instruction subpart "Test Setup." This suggests that the Test Setup likely contains crucial criteria for accurately classifying test specifications into CI configurations. However, drawing this conclusion with absolute certainty based on the available data is challenging. One plausible explanation for the results presented in Table 5.2, which does not necessarily support the aforementioned conclusion, is that the Test Setup section of the test instructions is written in a manner that is inherently more accessible for the machine learning model to learn from. In such a scenario, it is possible that Test Setup contains an equal amount of critical criteria used for classifying test specifications to CI configurations, despite the models performing significantly better when utilizing that particular subpart of the test instructions.

An aspect that contradicts the notion that Test Setup contains more crucial information for classifying CI configurations can be observed in Figure 5.2. Upon examining the dimensionally-reduced text embeddings, it becomes clear that the test specifications cluster together more prominently, and notably, they are more separated for the subpart Test Setup compared to the other subparts. The greater separations in the test specifications for Test Setup makes it easier for the submodel to learn and subsequently classify the CI configurations. This indicates that the improved performance of the submodel does not necessarily imply that the Test Setup inherently contains more crucial criteria regarding CI configuration.

6.2.3 Submodel—Word Distribution

Table 5.3 supports the hypothesis that Test Setup contains crucial information for classifying test specifications into CI configurations. This is evident from the highest performance the word distribution model achieves when the subpart Test Setup is utilized. However, it is important to note that the performance differences among the various subparts of the test instruction are significantly smaller when employing the word distribution model.

A pattern emerges through analysis of the identified key words in Table 5.5. Words associated with lower abstraction levels of the code, such as "list", "pathfinder", and "loops", are more frequently significant for CI Configuration 1 compared to the others. CI Configuration 1, which focused on executing microservices in isolation, is used to perform lower-level forms of testing, such as unit testing. Regarding CI configuration 2 and 3, it is more challenging to discern a clear pattern of abstraction level. However, both CI configurations contain key words that are associated with integration and system testing, such as "node", "available", and "disturbance". Furthermore, does configuration 2 and 3 contain key words that are associated with

non-functional requirements such as “robustness“.

Initially, we anticipated a gradual progression towards higher levels of system abstraction when comparing CI configurations 1 to 3. It was expected that CI configuration 1 would be characterized by lower-level, code-centric terminologies, which aligns with our observations. However, we were surprised to find that there was no distinct pattern between CI configurations 2 and 3. One possible explanation for this result is that both Configuration 2 and 3 involve the execution of multiple microservices. The key difference lies in the speed of execution, with configuration 2 prioritizing quick execution, while configuration 3 allows for longer execution times due to the inclusion of more intricate tests.

6.2.4 Submodel—Categorical

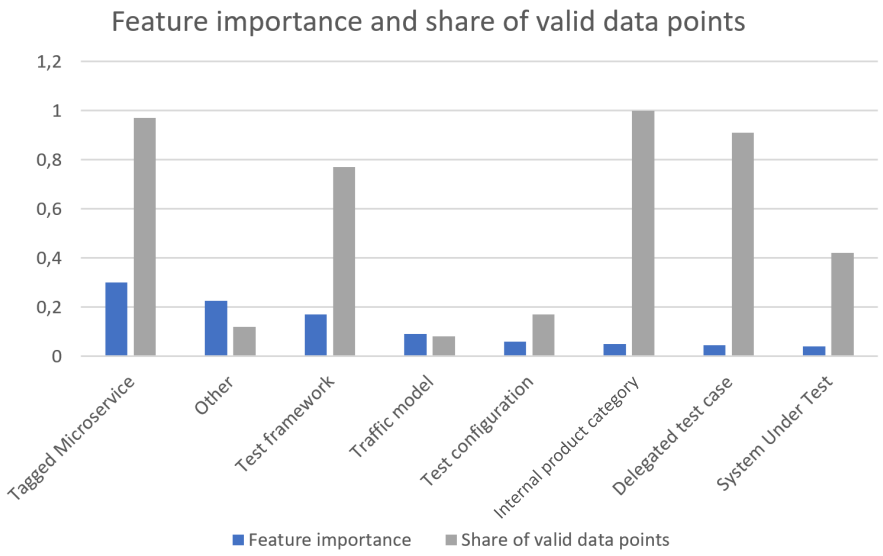

Figure 6.2: The feature importance of the categorical sub-model in relation to the share of valid data points of the features.

By analyzing the feature importance of the categorical submodel, interesting insights regarding the important characteristics for classifying test specifications into CI configurations can be identified. In Figure 5.3, it is depicted that approximately 75% of the model’s decision is based on features such as “Tagged Microservice”, “Test framework”, “Traffic model”, “Test configuration”, “Internal product category track”, “Delegated test cases”, and “System Under Test”. There are justifications that these features do have an important role. For example, the presence of tagged microservice might be significant as it relates to the execution environment, and the test framework might also be an important indicator, as different configurations may require different test frameworks.

Notably, there appears to be a strong correlation between feature importance and the share of valid data points for those features, as shown in Figure 6.2. This raises

the possibility that the features with the most significant impact on the machine learning model are not necessarily the most important when a person classifies test specifications into CI configurations. The model may place more emphasis on these features due to the larger quantity of valid data available.

An alternative approach to identifying the most important features is to consider the relationship between feature importance and the share of valid data points. If a feature has a high impact on the machine learning model but a small share of valid data points, it would indicate its true significance. Analyzing Figure 6.2, it becomes evident that "Traffic model" and "Test configuration" possess these characteristics and are, therefore, likely significant factors to consider when assigning CI configuration to test specifications. Notably, the feature labeled "Other" also appears to have high feature importance but a low share of valid data points. However, the feature "Other" bundles 12 features within its feature importance calculation, while the share of valid data points represents an average for all the bundled features. Therefore, the individual features within the "Other" bundle are unlikely to hold significant importance.

6.2.5 Ensemble Model

Upon observing Figure 5.3, it appears that the Semantic model utilizing the subparts Test Procedure and Pre/post Conditions holds the highest feature importance for the final classification of the ensemble. This finding is surprising, considering that other models individually exhibit significantly better performance. For instance, the categorical model achieves an accuracy of 0.89, and the Semantic model utilizing Test Setup attains an accuracy of 0.83, both of which are considerably higher than when both Test Procedure and Pre/post Conditions are used in the Semantic model, (0.76 and 0.78). The underlying reason for this discrepancy is unclear, indicating the need for further research to shed light on the underlying factors.

RQ2: The most critical features for assigning CI configurations are the test instruction, more specifically the subpart "Test Setup" of the test instruction. Furthermore, the categorical data bears significance, especially the features "Test Framework", "Traffic Model", "Test Configuration", "Internal product catagory", "Delegated Test Cases", and "System Under Test."

6.3 RQ3—Applicability of Supervised ML

The applicability of ML methods in a given domain depends on various factors, such as error tolerance, interpretability requirements, and domain stability. In this section, we will discuss the applicability of ML methods for this specific domain given the performance of the ML models (Section 5.1) and the results of the survey study (Section 5.3).

6.3.1 Correctness

Achievable performance is a crucial factor in determining the suitability of ML methods for a specific domain. If the model can achieve high accuracy consistently, it may be suitable for automating processes. On the other hand, if the model's accuracy is limited and cannot be relied upon for all cases, it may be better suited as a decision support system. The severity of misclassifications is also crucial in determining the suitability of ML methods in a domain. If misclassifications have severe consequences, automating processes entirely using ML may be less appropriate. In such cases, a decision support system may be more appropriate, where a domain expert can oversee the process and reduce misclassifications in the final decision.

As explained in Section 5.1, the ensemble model obtained an F1 score of 0.9075 and an accuracy of 0.9139, surpassing the median accuracy required for not negatively affecting the use as a decision support system (85%, as derived from the survey). This minimum threshold was established by domain experts who participated in the survey, with most experts considering an accuracy of up to 90% to be sufficient for decision support system implementation.

However, in general, a higher level of performance is required for fully automated systems. This is because, in such systems, there is no ability to verify the model's recommendations, thus necessitating a higher level of performance. According to the survey, the performance of the ensemble model meets the median performance requirement for implementation as a fully automated system, although it does so with a narrow margin. This proximity to the performance threshold raises questions about trust in the system. While some domain experts argue that an accuracy of 90% is sufficient, it does not necessarily imply their acceptance of a system that merely surpasses this threshold. Moreover, certain domain experts demand an even higher level of performance, ranging from 95% to 99%. Consequently, these experts could distrust the system and actively oppose its implementation.

6.3.2 Speed

As presented in Section 5.3, the survey indicates that the speed at which ML models generate recommendations must be within a "couple of minutes" to ensure the usability of the system is not negatively affected. As shown in Section 5.1, the ML model generates recommendations in just 0.0037 seconds on average, well below the threshold for negatively impacting usability. Considering that the proposed model also meets the performance metrics for accuracy, it appears that supervised ML models can indeed meet the speed constraints in the given domain.

Interestingly, domain experts emphasize that the speed of recommendation generation could be traded for better recommendations. Given the significant gap between the speed of the model proposed in this study and the maximum threshold indicated by the survey, a more complex model than the one presented could prove beneficial. As a practitioner planning to implement a similar ML-based system for assigning CI configuration, an ensemble model with additional sub-models could be of interest.

Furthermore, increasing the complexity of the proposed sub-models might also be beneficial. For instance, incorporating neural networks as prediction heads could potentially improve the accuracy performance of the models, albeit at the expense of generation speed. However, using more complex models usually requires much training data which could be an issue in the software testing domain. In the case of Ericsson CloudRAN, using a more complex model—such as a neural network—was not possible due to the mentioned constraints.

6.3.3 Value

To determine the applicability of ML methods in any domain, evaluating the value they bring in terms of time and effort saved and financial benefits is necessary. If implementing an ML model results in minimal savings while requiring high costs, then its value is diminished, making ML less relevant and thus less applicable.

As seen in Section 5.3, domain experts believe that the proposed decision support system brings benefits in terms of time saved when assigning CI configurations to test specifications. However, there is uncertainty regarding the exact magnitude of the time savings that can be achieved. This uncertainty could be attributed to the varying levels of the perceived difficulty in assigning CI configurations to test specifications among domain experts. Those who view the task as more challenging may likely see greater benefits in using the decision support system, while those who find it less challenging may perceive the benefits as less significant.

The survey reveals that most domain experts hold a favorable view regarding the proposed decision support system's impact on reducing CI configuration misassignments. The median estimate suggests a significant reduction of approximately 60% in misassignments. This finding indicates that the decision support system possesses considerable business value, as fewer misassignments translate into reduced feature delays and faster delivery to customers.

Moreover, it is essential to note that the domain experts' responses regarding the reduced need for domain expertise to perform the task may be biased. It is in their interest to maintain their expertise's relevance; thus, they may not fully acknowledge the potential benefits of reducing the need for domain expertise. Therefore, it is necessary to consider these biases when interpreting the domain experts' responses and to conduct further research to determine the actual time savings and benefits that can be achieved using the proposed decision support system.

The financial savings estimated in Section 5.3 underscore the potentially significant value of employing ML models as decision support for assigning CI configurations. However, it should be noted that the cost of developing the ML model was not explicitly evaluated. Nonetheless, considering that the proposed system in this study was developed by two students in approximately two months, a high return on investment (ROI) can be inferred.

Nevertheless, the certainty of the estimated savings derived from the proposed tool

is subject to debate. The survey sample size was limited to only 10 respondents, which poses certain limitations on the generalizability of the results. Additionally, some of the respondents indicated that they were unsure about certain questions, further reducing the number of reliable data points available for analysis.

6.3.4 Usability

Based on the responses from domain experts, it becomes evident that a decision support system brings significant value by simplifying the task of assigning CI configurations to test specifications. The mean difficulty rating decreased noticeably from 3.6 out of 5 to 1.9 out of 5, indicating a substantial improvement in the perceived complexity of the task. Additionally, most domain experts expressed their willingness to actively utilize the decision support system if it were to be implemented. This overwhelming response further reinforces the notion that the decision support system would benefit domain experts in performing this task.

However, it is worth noting that a few outliers indicated their reluctance to use the decision support system. This can be attributed to two potential reasons. Firstly, these individuals might not consider the task of assigning CI configurations to test specifications as challenging to begin with, thereby diminishing the perceived value of such a system for them. Secondly, their lack of trust in AI-based systems in the workplace could also contribute to their hesitancy to adopt the decision support system.

Overall, the survey results highlight the overall positive impact of the decision support system on reducing task difficulty and the strong inclination of most domain experts to utilize it while acknowledging the presence of a minority with different perspectives based on task perception and trust in AI systems.

6.3.5 Interpretability

The level of interpretability required in a domain is another crucial factor in determining the applicability of ML methods. Applying ML methods is comparatively more straightforward in domains with minimal to no interpretability requirements. However, in domains with a high level of interpretability required, applying ML methods is more challenging as interpretability becomes an added decision factor [75]. The domain experts believe that interpretability is very important for a decision support system for assigning CI configurations to test specifications. Moreover, they also think the proposed decision support system meets the interpretability requirements. Connecting this to existing research, the results from the survey study regarding interpretability indicate that ML is applicable in this domain. This is supported by the belief of domain experts that the proposed decision support system for assigning CI configurations to test specifications meets the high interpretability requirements necessary for this domain.

6.3.6 Stability

As discussed earlier, the stability of the underlying domain is an essential characteristic of a domain suitable for machine learning methods. The training data shows a relatively balanced distribution among the CI configurations, which the sub-model word distribution model learns. However, if the distribution of test specifications belonging to the CI configurations were to change, the word distribution model's performance might be negatively affected due to the altered priors. If the stability of the CI configuration distribution is not maintained, more than simply retraining, the model may be required to adapt to the new conditions. This is because the current data may have a significantly different distribution compared to the entire dataset over time. It is worth mentioning that word distribution is the only sub-model impacted by an unstable distribution of the CI configurations since no other submodel depends on the estimated priors of the CI configurations.

Consequently, the ensemble model should be relatively robust when exposed to this instability. The domain experts believe that the distribution of CI configurations will change. This lowers the applicability of ML models highly dependent on this prior, such as the word-distribution model in this thesis. Knowing that the prior is susceptible to change, it is beneficial to focus on ML models that are less dependent on the prior distribution, such as the semantic and categorical models.

Another potential area of instability in this domain relates to the formulation and writing of test instructions. Frequent changes in the testing habits of employees at Ericsson or the lack of established best practices for test specifications formulation can threaten the applicability of machine learning models in this context. According to domain experts, there is a considerable likelihood that the structure of test specifications will change in the future, as reflected by a mean score of 3.6 out of 5. A change in the structure of the test instruction implies a corresponding change in the input to the decision support system, increasing the need for maintaining the system. This increased maintenance requirement can lead to higher costs for keeping the system running, lowering its overall applicability.

Significant trends in CI and the emergence of new techniques and CI platforms can threaten the longevity of machine learning models. Moreover, the stability of the domain relies heavily on the stability of the predefined CI configurations. Any alterations to these predefined CI configurations could render the training data obsolete, thus diminishing the domain's suitability for machine learning. According to the survey, no domain experts believed that it was unlikely for changes to occur in the CI configurations in the future. Instead, most respondents thought that it was either very likely or had a moderate likelihood. Given the moderate to high likelihood of changes in the CI configurations, the applicability of machine learning is reduced since the availability of relevant training data is at risk, which in turn can negatively impact the performance of the models, lowering the overall applicability.

RQ3: Supervised ML appears to be well-suited for the given task. The proposed ML model fulfills the correctness and speed requirements expressed by the domain experts. Furthermore, it adds significant business value and is highly likely to be used in operations if implemented. Nevertheless, there are some concerns about the stability of the domain, which may impact the maintenance requirements of supervised ML models.

6.4 Comparison to Related Work

This study successfully demonstrates the feasibility of utilizing supervised machine learning to assign test cases to pre-defined CI configurations, a task that has not been previously explored. Furthermore, it addresses the research gap of focusing on CI configuration assignment rather than solely identifying CI configuration errors at the code level, as discussed in Santolucito et al. [15]. While the study showcases the ability to assign test cases to pre-defined CI configurations, it is essential to note that it is possible that non of the pre-determined configurations is optimal for that specific test case. In such cases, the solution proposed by Vasallo et al. [70] offers a more practical approach. For future research, it would be intriguing to combine Vasallo et al.'s [70] method with ours, resulting in a system that can not only select appropriate CI configurations from a pre-defined set but also provide insights on the level of suitability for the existing pre-determined configurations.

6.5 Threats to Validity

This section discusses potential internal and external threats to the study's validity, which may affect the findings' accuracy, reliability, or generalizability.

6.5.1 Internal Validity

This study faces several potential internal threats to its validity. One significant concern is the inherent randomness in evaluating machine learning models, which can introduce variability in the results. To mitigate this issue, we took measures to increase the reliability of our findings by running the models 100 times and averaging the results.

Furthermore, the risk of faults in our implementation poses another potential threat. However, we minimized this risk by leveraging reliable pre-built components from trusted sources like Sci-Kit and Facebook.

Another potential threat arises from the limited scope of hyperparameter tuning, which was constrained by computational limitations. This constraint may have led to suboptimal model performance.

Lastly, the limited data availability constrained our exploration of other relevant machine learning techniques, such as neural networks, which could have enriched

the study.

6.5.2 External Validity

The external validation threat to the study's findings can be considered high due to the exclusive focus on data from the telecom industry. Moreover, the survey in the study does rely solely on responses from a small number of employees at Ericsson, which further increases the risk of low external validity.

6.5.3 Generalizability

The concept of test specification and its formulation can be considered relatively generic, suggesting that the findings in this study may have broader applicability beyond the telecom industry. Organizations that employ the same methodology of assigning test specifications to pre-determined CI configurations are likely to achieve similar results, regardless of the industry. One argument supporting the generalizability of these findings beyond the telecom industry is that the model considered few industry-specific aspects. Even if the test specifications in other domains differ slightly from those examined in this thesis, there are likely to be similarities in certain aspects. For example, if the test specifications consist solely of natural language test instructions, similar semantic models can be utilized while omitting the categorical models. On the other hand, if the test specifications solely comprise categorical data, the natural language-based models can be omitted in favor of the categorical model. However, it would be beneficial to conduct further research to explore the possibility of generalizing the findings of this study.

7

Conclusion

This study presents a novel supervised machine learning (ML) model for accurately assigning CI configurations to test specifications. Our model utilizes an ensemble architecture comprising three sub-models and a rule-based component. Each sub-model focuses on a specific aspect of the problem: capturing the semantic meaning of test instructions in free natural language, applying Bayesian inference to analyze the words in test instructions, and handling categorical data in test specifications.

To gain deeper insights into the assignment process, we extensively analyzed our model, identifying the key features that play a crucial role in this task. Based on our findings, we developed a decision support system that leverages the power of our ML model to evaluate the applicability of supervised ML in assigning CI configurations to test specifications.

To validate the effectiveness of our approach, we conducted a survey study involving domain experts from Ericsson, who provided valuable feedback on the proposed decision support system. This study explored the applicability of ML models and showcased our system's potential benefits in real-world scenarios.

We have shown that it is possible to achieve an accuracy of 0.9139 and an F1 score of 0.9075 using supervised ML, which surpasses the performance requirements established by most domain experts. Nevertheless, it is essential to acknowledge that a more robust ML model could have been attained with better data quality and increased quantity.

An analysis of the ensemble model has indicated that certain parts of the test specifications play a crucial role in determining the assignment outcome. Specifically, features such as "Tagged Microservice", "Test Framework", "Traffic Model", "Test Configuration", "SW Track", "Delegated Test Cases", and "System Under Test" appear to be highly influential. However, it is important to note that drawing definitive conclusions is challenging due to incomplete data.

We have observed that supervised ML can meet the performance standards defined by domain experts, encompassing factors such as the accuracy and speed of the ML model. This becomes particularly evident when implementing it as a decision support system. However, when considering its implementation as a fully automated

system, it may not be immediately apparent whether it can be effectively applied. This is because the performance of the ML model closely aligns with the expected performance of domain experts, which could impact the trust of certain users.

Implementing supervised ML in this domain brings significant potential business value, manifesting in several tangible benefits. Firstly, it could lead to a notable reduction in misassignments, ensuring more accurate assignment of CI configurations. This improvement could minimize errors and enhance the overall quality of the testing process. Secondly, the use of supervised ML could result in a considerable reduction in the time required for assigning CI configurations. By semi-automating this process, the testing process becomes more efficient and allows engineers to focus their efforts on other critical tasks, thereby increasing productivity. Furthermore, applying supervised ML could reduce fault slip through, meaning potential issues or faults are identified and addressed promptly. This proactive approach mitigates the risks of shipping bugs and other defects to customers.

Further research into the applicability of implementing a fully automated CI configuration assignment tool would be of interest. We propose conducting a comprehensive case study that implements a fully automated system in collaboration with a company, as this would provide real-world insights and practical implications. Moreover, considering the speed of our current model and the narrow margin of accuracy over achievement compared to the required for full automation, it is worth exploring more complex machine learning (ML) models to enhance performance. One intriguing avenue to investigate is using neural networks, which have demonstrated remarkable capabilities in various domains. By leveraging the power of neural networks, we can potentially achieve even higher accuracy in the CI configuration assignment process making it suitable for fully automated adaptation.

Bibliography

[1] S. Tahvili and L. Hatvani, *Artificial Intelligence Methods for Optimization of the Software Testing Process With Practical Examples and Exercises*. Elsevier, July 2022.

[2] S. Ahamed, "Studying the feasibility and importance of software testing: An analysis," *arXiv preprint arXiv:1001.4193*, 2010.

[3] M. Ellims, J. Bridges, and D. C. Ince, "The economics of unit testing," *Empirical Software Engineering*, vol. 11, pp. 5–31, 2006.

[4] M. Noorian, E. Bagheri, and W. Du, "Machine learning-based software testing: Towards a classification framework.," in *SEKE*, pp. 225–229, 2011.

[5] D. S. Battina, "Artificial intelligence in software test automation: a systematic literature review," *International Journal of Emerging Technologies and Innovative Research (www. jetir. org| UGC and issn Approved), ISSN*, pp. 2349–5162, 2019.

[6] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, "Software testing techniques: A literature review," in *2016 6th international conference on information and communication technology for the Muslim world (ICT4M)*, pp. 177–182, IEEE, 2016.

[7] A. Miller, "A hundred days of continuous integration," in *Agile 2008 conference*, pp. 289–293, IEEE, 2008.

[8] M. Santolucito, J. Zhang, E. Zhai, and R. Piskac, "Statically verifying continuous integration configurations," *arXiv preprint arXiv:1805.04473*, 2018.

[9] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices," *IEEE access*, vol. 5, pp. 3909–3943, 2017.

[10] G. Pinto, F. Castor, R. Bonifacio, and M. Rebouças, "Work practices and challenges in continuous integration: A survey with travis ci users," *Software: Practice and Experience*, vol. 48, no. 12, pp. 2223–2236, 2018.

[11] M. Fowler and M. Foemmel, "Continuous integration," 2006.

[12] S. Tahvili, *Multi-criteria optimization of system integration testing*. PhD thesis, Mälardalen University, 2018.

[13] A. Debbiche, M. Dienér, and R. Berntsson Svensson, "Challenges when adopting continuous integration: A case study," in *Product-Focused Software Process Improvement: 15th International Conference, PROFES 2014, Helsinki, Finland, December 10-12, 2014. Proceedings 15*, pp. 17–32, Springer, 2014.

[14] R. O. Rogers, "Scaling continuous integration," in *Extreme Programming and Agile Processes in Software Engineering: 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004. Proceedings 5*, pp. 68–76, Springer, 2004.

[15] M. Santolucito, J. Zhang, E. Zhai, J. Cito, and R. Piskac, "Learning ci configuration correctness for early build feedback," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 1006–1017, IEEE, 2022.

[16] D. Medvedev and K. Aksyonov, "The development of a simulation model for assessing the ci/cd pipeline quality in the development of information systems based on a multi-agent approach," in *MATEC Web of Conferences*, vol. 346, p. 03095, EDP Sciences, 2021.

[17] V. H. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. Dias, and M. P. Guimarães, "Machine learning applied to software testing: A systematic mapping study," *IEEE Transactions on Reliability*, vol. 68, no. 3, pp. 1189–1212, 2019.

[18] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*, pp. 426–437, 2016.

[19] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 12–22, 2017.

[20] K. Wiegers and J. Beatty, *Software requirements*. Pearson Education, 2013.

[21] R. Malan, D. Bredemeyer, *et al.*, "Functional requirements and use cases," *Bredemeyer Consulting*, 2001.

[22] R. Fulton and R. Vandermolen, *Airborne Electronic Hardware Design Assurance: A Practitioner's Guide to RTCA/DO-254*. CRC Press, 2017.

[23] M. Glinz, "On non-functional requirements," in *15th IEEE international requirements engineering conference (RE 2007)*, pp. 21–26, IEEE, 2007.

[24] B. Lightsey, "Systems engineering fundamentals," tech. rep., DEFENSE ACQUISITION UNIV FT BELVOIR VA, 2001.

[25] A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. Tripp, "Software engineering body of knowledge," *IEEE Computer Society, Angela Burgess*, vol. 25, 2004.

[26] J. Clarkson and C. Eckert, "Design process improvement: a review of current practice," 2010.

[27] L. Chung and J. C. S. do Prado Leite, "On non-functional requirements in software engineering," *Conceptual modeling: Foundations and applications: Essays in honor of john mylopoulos*, pp. 363–379, 2009.

[28] A. De Lucia and A. Qusef, "Requirements engineering in agile software development," *Journal of emerging technologies in web intelligence*, vol. 2, no. 3, pp. 212–220, 2010.

[29] M. Olan, "Unit testing: test early, test often," *Journal of Computing Sciences in Colleges*, vol. 19, no. 2, pp. 319–328, 2003.

[30] A. De Camargo, I. Salvadori, R. d. S. Mello, and F. Siqueira, "An architecture to automate performance tests on microservices," in *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, pp. 422–429, 2016.

[31] P. Runeson, "A survey of unit testing practices," *IEEE software*, vol. 23, no. 4, pp. 22–29, 2006.

[32] T. Xie, K. Taneja, S. Kale, and D. Marinov, "Towards a framework for differential unit testing of object-oriented programs," in *Second International Workshop on Automation of Software Test (AST'07)*, pp. 5–5, IEEE, 2007.

[33] E. Daka and G. Fraser, "A survey on unit testing practices and problems," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pp. 201–211, IEEE, 2014.

[34] ISO Central Secretary, "Systems and software engineering — Vocabulary," Standard SO/IEC/IEEE 24765:2017, International Organization for Standardization, Geneva, CH, 2017.

[35] H. K. Leung and L. White, "A study of integration testing and software regression at the integration level," in *Proceedings. Conference on Software Maintenance 1990*, pp. 290–301, IEEE, 1990.

[36] H. Jiang, X. Li, Z. Yang, and J. Xuan, "What causes my test alarm? auto-

matic cause analysis for test alarms in system and integration testing," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pp. 712–723, IEEE, 2017.

[37] P. C. Jorgensen and C. Erickson, "Object-oriented integration testing," *Communications of the ACM*, vol. 37, no. 9, pp. 30–38, 1994.

[38] R. Binder, *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Professional, 2000.

[39] R. V. Binder, "Testing object-oriented software: a survey," *Software Testing, Verification and Reliability*, vol. 6, no. 3-4, pp. 125–252, 1996.

[40] L. Briand and Y. Labiche, "A uml-based approach to system testing," *Software and systems modeling*, vol. 1, pp. 10–42, 2002.

[41] R. Miller and C. T. Collins, "Acceptance testing," *Proc. XPUniverse*, vol. 238, 2001.

[42] C. Rehn, "Continuous integration: Aspects in automation and configuration management," *Term Paper, TU Kaiserslautern, Germany*, 2012.

[43] M. Meyer, "Continuous integration and its tools," *IEEE software*, vol. 31, no. 3, pp. 14–16, 2014.

[44] Red Hat, "What is a CI/CD pipeline?," May 2022. `https://www.redhat.com/en/topics/devops/what-cicd-pipeline`, Last accessed on 2023-06-07.

[45] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[46] P. P. Shinde and S. Shah, "A review of machine learning and deep learning applications," in *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*, pp. 1–6, IEEE, 2018.

[47] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," *Machine learning techniques for multimedia: case studies on organization and retrieval*, pp. 21–49, 2008.

[48] P. Dayan, M. Sahani, and G. Deback, "Unsupervised learning," *The MIT encyclopedia of the cognitive sciences*, pp. 857–859, 1999.

[49] W. S. Noble, "What is a support vector machine?," *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[50] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.

[51] D. Boswell, "Introduction to support vector machines," *Departement of Computer Science and Engineering University of California San Diego*, vol. 11, 2002.

[52] N. Cristianini, J. Shawe-Taylor, *et al.*, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[53] P.-H. Chen, C.-J. Lin, and B. Schölkopf, "A tutorial on ν-support vector machines," *Applied Stochastic Models in Business and Industry*, vol. 21, no. 2, pp. 111–136, 2005.

[54] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, pp. 197–227, 2016.

[55] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[56] T. Hengl, M. Nussbaum, M. N. Wright, G. B. Heuvelink, and B. Gräler, "Random forest as a generic framework for predictive modeling of spatial and spatio-temporal variables," *PeerJ*, vol. 6, p. e5518, 2018.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[58] R. van de Schoot, S. Depaoli, R. King, B. Kramer, K. Märtens, M. G. Tadesse, M. Vannucci, A. Gelman, D. Veen, J. Willemsen, *et al.*, "Bayesian statistics and modelling," *Nature Reviews Methods Primers*, vol. 1, no. 1, p. 1, 2021.

[59] L. Hong, "Law of total probability and bayes' theorem in riesz spaces," *arXiv preprint arXiv:1502.00124*, 2015.

[60] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, 2011.

[61] G. G. Chowdhury, "Natural language processing," *Fundamentals of artificial intelligence*, pp. 603–649, 2020.

[62] E. D. Liddy, "Natural language processing," 2001.

[63] S. Feldman, "Nlp meets the jabberwocky: Natural language processing in information retrieval," *ONLINE-WESTON THEN WILTON-*, vol. 23, pp. 62–73, 1999.

[64] V. Tanna, "Introduction to different levels of natural language processing," Mar 2021.

[65] Y. Meng, J. Huang, G. Wang, C. Zhang, H. Zhuang, L. Kaplan, and J. Han, "Spherical text embedding," *Advances in neural information processing systems*, vol. 32, 2019.

[66] J. C. Young and A. Rusli, "Review and visualization of facebook's fasttext pretrained word vector model," in *2019 international conference on engineering, science, and industrial applications (ICESI)*, pp. 1–6, IEEE, 2019.

[67] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[68] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[69] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," *arXiv preprint arXiv:1802.06893*, 2018.

[70] C. Vassallo, S. Proksch, A. Jancso, H. C. Gall, and M. Di Penta, "Configuration smells in continuous delivery pipelines: a linter and a six-month study on gitlab," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 327–337, 2020.

[71] M. Bregman, "Execution platform assignments in ci / cd systems," U.S. Patent 20230009997A1, Jan. 2023.

[72] H. J. B. M. F. C. A. N. SHRIPAD, "Dynamic automation of pipeline workpiece selection," China Patent 115668129A, Jan. 2023.

[73] G. Bregman, "Ci/cd pipeline to container conversion," U.S. Patent 20220391215A1, Dec. 2022.

[74] D. Rafique and L. Velasco, "Machine learning for network automation: overview, architecture, and applications [invited tutorial]," *Journal of Optical Communications and Networking*, vol. 10, no. 10, pp. D126–D143, 2018.

[75] R. M. Morais, "On the suitability, requisites, and challenges of machine learning," *Journal of Optical Communications and Networking*, vol. 13, no. 1, pp. A1–A12, 2021.

[76] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, pp. 131–164, 2009.

[77] L. K. Owens, "Introduction to survey research design," in *SRL fall 2002 seminar series*, vol. 1, 2002.

A

Appendix 1

A.1 Survey Study Questions

Table A.1: Survey Questions (Correctness)

Question number	Question wording	Response options
1	What level of accuracy is necessary for the system to function as a decision support tool? (Answer in the percentage of correct assignments)	Open-ended text field
2	What level of accuracy is necessary for the system to function as a fully automated system? (Answer in the percentage of correct assignments)	Open-ended text field
3	In this domain, how significant are the consequences of misassignment? (5 = very significant, 1 = insignificant)?	1, 2, 3, 4, 5
4	Are the consequences of misassignment the same for all CI configurations?	Yes, No
5	If not, how do the consequences differ across different CI configurations?	Open-ended text field

Table A.2: Survey Questions (Speed)

Question number	Question wording	Response options
6	What is an acceptable time frame for the model to generate a recommendation without affecting your willingness to use the tool?	Open-ended text field
7	Currently, the model generates recommendations with a level of accuracy of 0.9139, an F1-score of 0.9075, and a speed of 0.036 seconds. Would it be advantageous to sacrifice some speed to improve the accuracy/F1 score?	Yes, No
8	If so, how much improvement would be necessary to justify the slower speed? Give an example of a good trade-off between accuracy and speed.	Open-ended text field

Table A.3: Survey Questions (Value)

Question number	Question wording	Response options
9	How much faster will the tool allow you to assign CI configurations to test specifications? What is the estimated time savings per test specifications?	Open-ended text field
10	What is your best estimate of the reduction in misassignments of CI configurations that can be achieved using the proposed tool? (Answer in percentage)	Open-ended text field
11	Approximately how many misassignments of CI configurations are made yearly?	Open-ended text field
12	There is an estimation that a misassignment of a CI configuration leads to an average feature delay of 2 months in delivery. Do you agree with this estimation?	Yes, No
13	Skip if you answered Yes to the previous question. On average, how long of a feature delay do you estimate one misassignment results in?	Open-ended text field
14	What is your best estimate of the cost (in SEK) associated with this feature delay?	Open-ended text field
15	How many fewer fault-slip throughs (i.e., undetected bugs) are expected to occur when using the proposed system compared to the current process?	Open-ended text field
16	What is your best estimate of the average cost (in SEK) associated with a fault-slip through?	Open-ended text field
17	Do you believe that using the proposed tool will reduce the need for domain expertise in assigning CI configurations?	Yes, No

Table A.4: Survey Questions (Usability)

Question number	Question wording	Response options
18	Currently, what level of difficulty is involved in assigning CI configurations to test specifications? (5 = more difficult, 1 = less difficult)	1, 2, 3, 4, 5
19	How difficult would it be to assign CI configurations to test specifications using the proposed tool? (5 = more difficult, 1 = less difficult)	1, 2, 3, 4, 5
20	How well does the proposed system integrate with your existing workflow? (5 = very well, 1 = less well)	1, 2, 3, 4, 5
21	If the proposed system were fully implemented, would you actively use it? (5 = very inclined to using it, 1 = not inclined to using it)	1, 2, 3, 4, 5
22	Overall, to what extent do you place trust in AI-based systems for practical use in the workplace? (5 = high trust, 1 = low trust)	1, 2, 3, 4, 5

Table A.5: Survey Questions (Interpretability)

Question number	Question wording	Response options
23	How important is interpretability, i.e., understanding how the system generates its recommendations to you? (5 = very important, 1 = not important)	1, 2, 3, 4, 5
24	Do you believe that the current method used by the model to present its recommendations adequately addresses interpretability concerns? (5 = strongly agree, 1 = strongly disagree)	1, 2, 3, 4, 5

Table A.6: Survey Questions (Stability)

25	How likely is it that the distribution of the various CI configurations will change in the future? (5 = very likely, 1 = very unlikely)	1, 2, 3, 4, 5
26	How likely is it that the current structure of test specifications is subject to changes in the future? (5 = very likely, 1 = very unlikely)	1, 2, 3, 4, 5
27	How likely are the different CI configurations subject to future modifications? (5 = very likely, 1 = very unlikely)	1, 2, 3, 4, 5

B

Appendix 2

B.1 Hyper-parameter Tuning Results

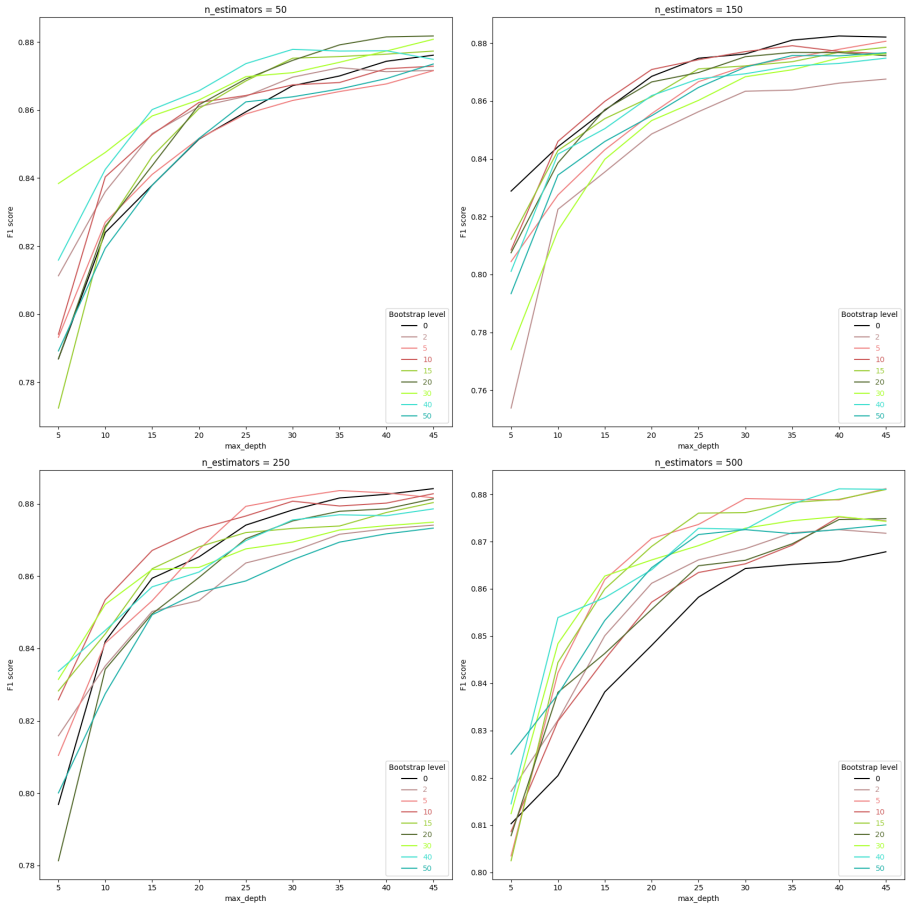

Figure B.1: F1 Scores of Tag-based Model by Bootstrap Level, Maximum Depth, and Number of Estimators

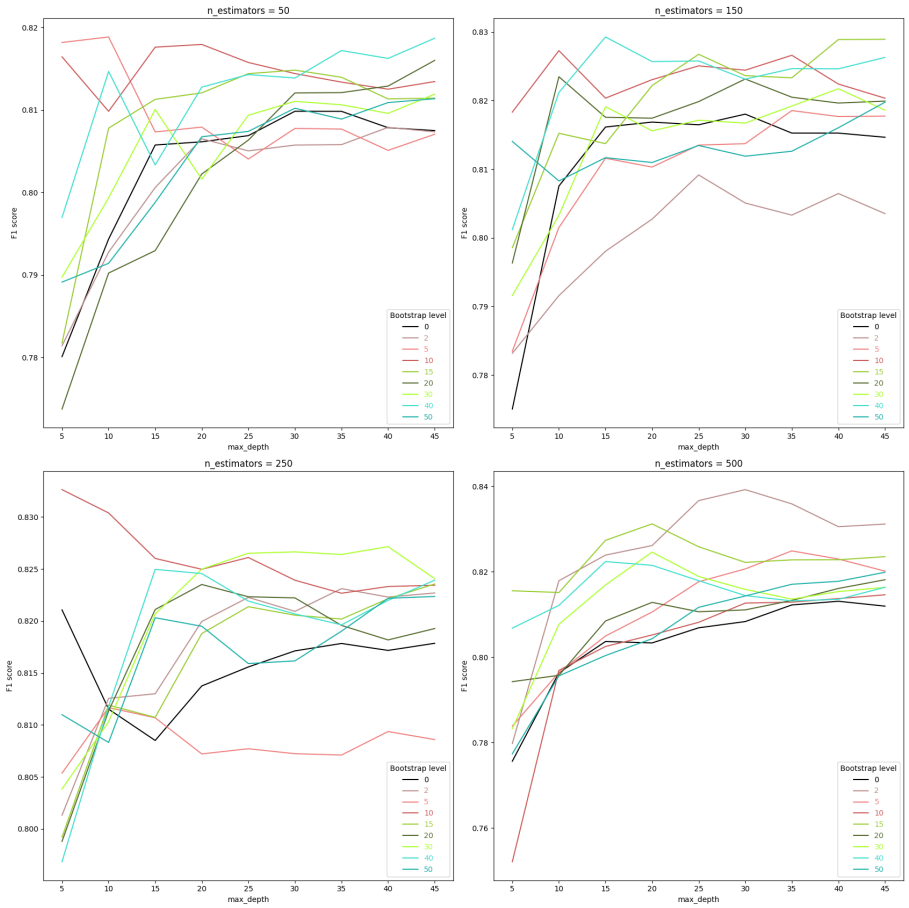

Figure B.2: F1 Scores of Semantic Model Sub-model 1 by Bootstrap Level, Maximum Depth, and Number of Estimators

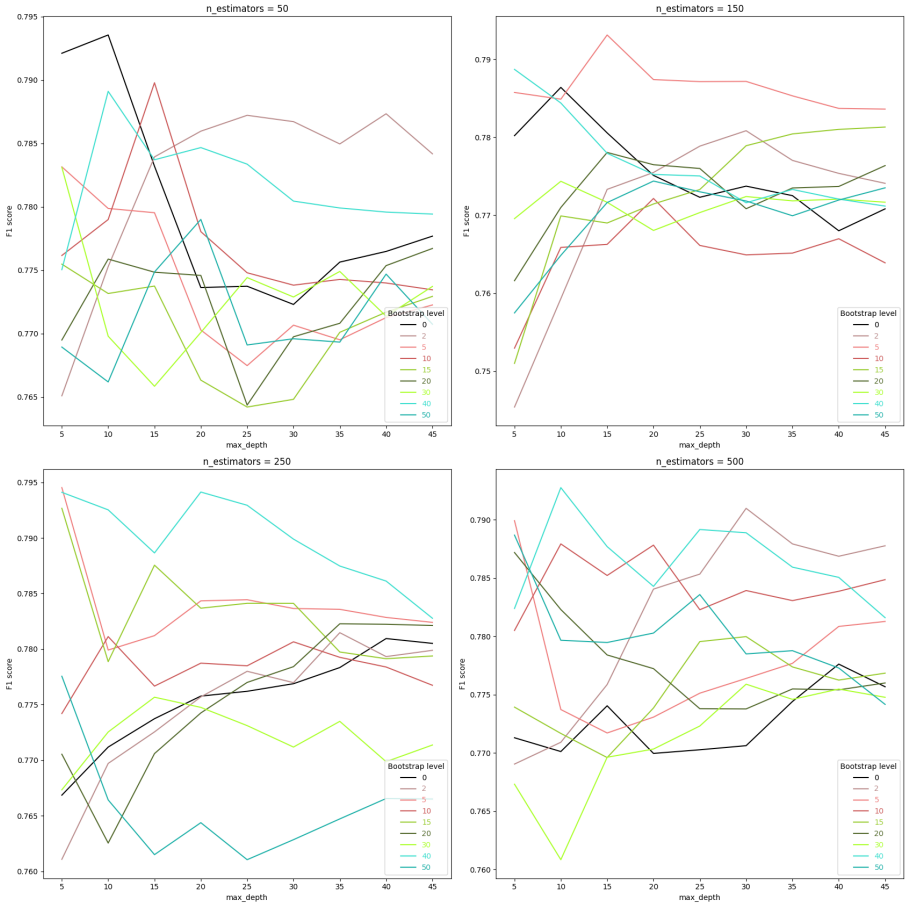

Figure B.3: F1 Scores of Semantic Model Sub-model 2 by Bootstrap Level, Maximum Depth, and Number of Estimators

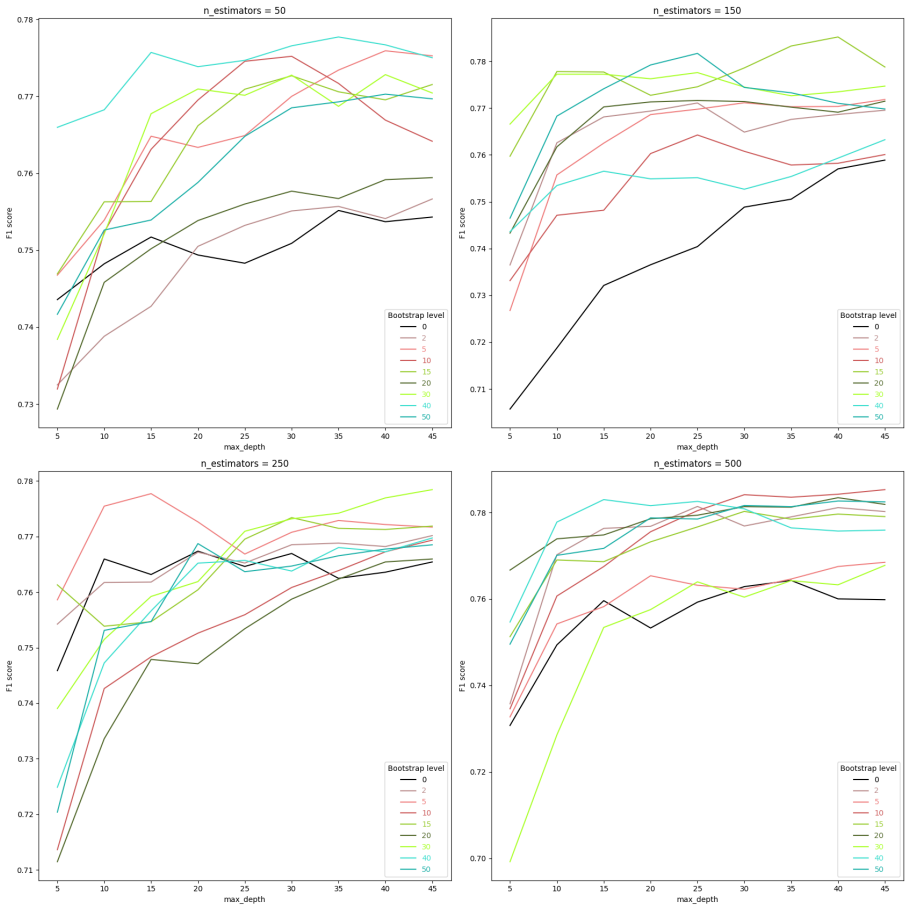

Figure B.4: F1 Scores of Semantic Model Sub-model 3 by Bootstrap Level, Maximum Depth, and Number of Estimators

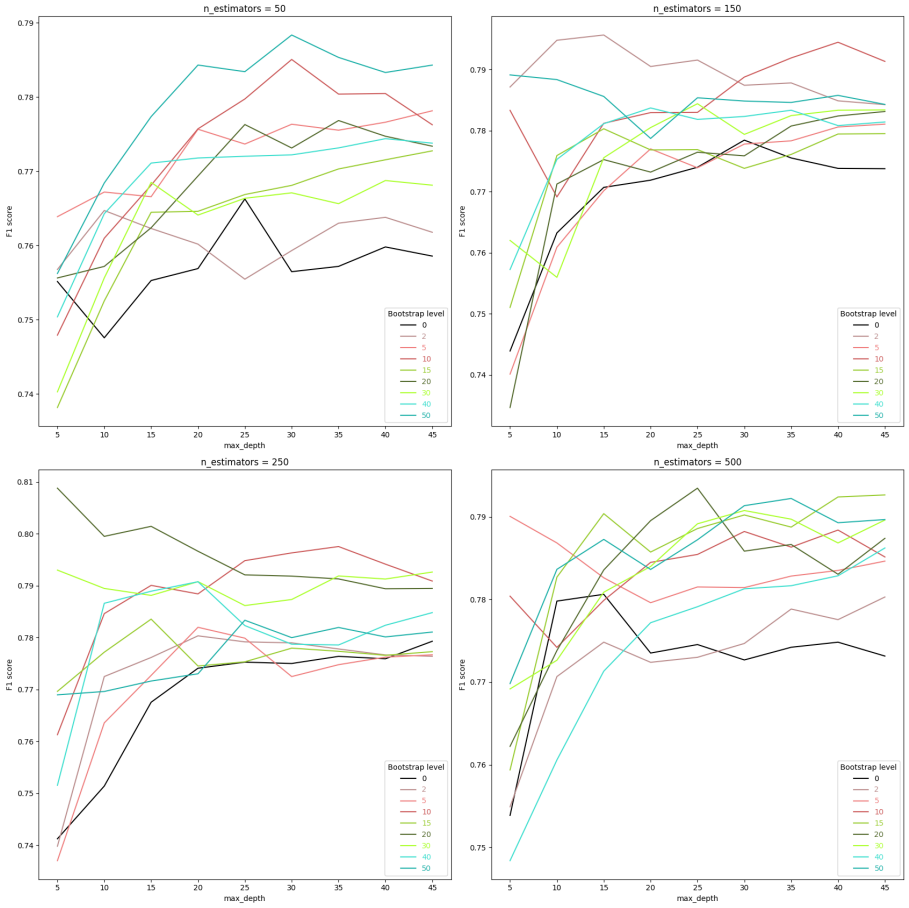

Figure B.5: F1 Scores of Semantic Model Sub-model 4 by Bootstrap Level, Maximum Depth, and Number of Estimators

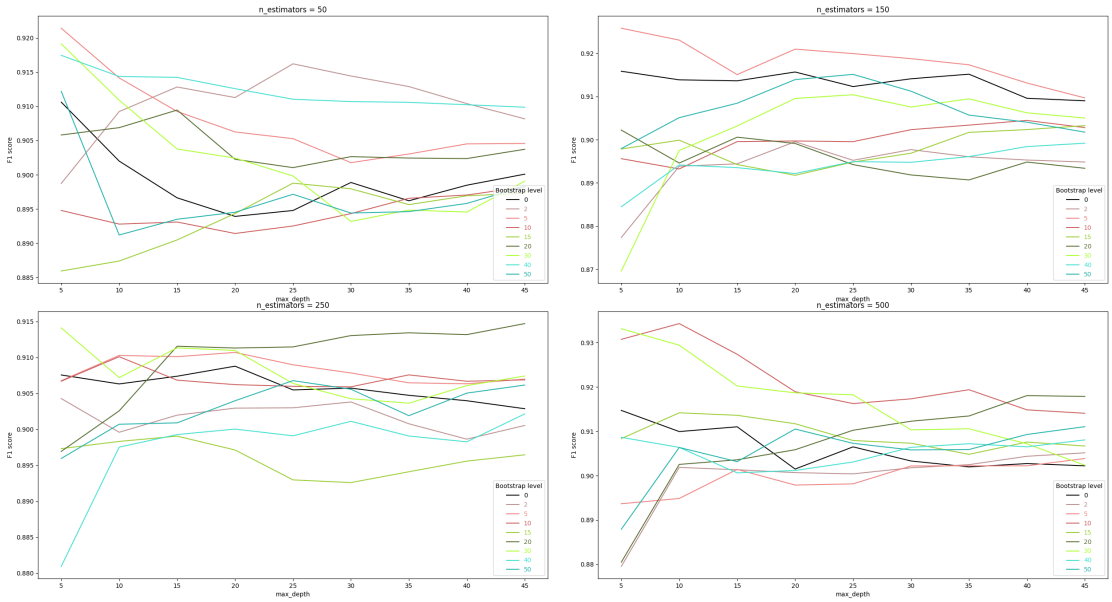

Figure B.6: F1 Scores of Ensemble Model by Bootstrap Level, Maximum Depth, and Number of Estimators